M269

Exam Revision

Contents

1 M269 Exam Revision Agenda & Aims

1. Welcome and introductions
2. Revision strategies
3. M269 Exam — Part 1 has 15 questions 60%
4. M269 Exam — Part 2 has 2 questions 40%
5. M269 Exam — 3 hours, Part 1 100 mins, Part 2 70 mins
6. M269 2014J exam — Part 1
7. M269 2014J exam — Part 2
8. Topics and discussion for each question
9. Exam techniques
10. Two sessions

1.1 Introductions & Revision Strategies

- Introductions

- What other exams are you doing this year ?
- Each give one exam tip to the group

1.2 M269 Exam 2016J

- Not examined this presentation:
- Unit 4, Section 2 String search
- Unit 7, Section 2 Logic Revisited
- Unit 7, Section 4 Beyond the Limits

2 M269 Prsntn 2014J Exam Qs

2.1 M269 2014J Exam Qs

- M269 Algorithms, Data Structures and Computability
- Presentation 2014J Exam
- Date Monday, 8 June 2015 Time 10:00–13:00
- There are **TWO** parts to this examination. You should attempt all questions in **both** parts
- **Part 1** carries **60 marks** — **100 minutes**
- **Part 2** carries **40 marks** — **70 minutes**
- **Note** see the original exam paper for exact wording and formatting — these slides and notes may change some wording and formatting

Go to Exam Soln s

2.2 M269 2014J Exam Q Part1

- Answer every question in this part.
- Answers to questions in this part should be written on this paper in the spaces provided, or in the case of multiple-choice questions you should tick the appropriate box(es).
- If you tick more boxes than indicated for a multiple choice question, you will receive **no** marks for your answer to that question.

Go to Exam Soln Part1

3 Units 1 & 2

3.1 Unit 1 Introduction

- Unit 1 Introduction
- Computation, computable, tractable
- Introducing Python
- What are the three most important concepts in programming ?
    1. Abstraction
    2. Abstraction
    3. Abstraction
- Quote from Paul Hudak (1952–2015)

3.2 M269 2014J Exam Q 1

- **Question 1** Which **two** of the following statements are true ? **(2 marks)**

A. A computational thinker can explain why some computational problems do not have a good computational solution.

B. The main factor when deciding the best algorithm for a given task is the programming language in which the algorithm is to be implemented.

C. A decision problem can be defined as a formally stated problem to which the answer is either yes or no.

D. The data structure chosen to hold data does not affect the efficiency of an algorithm operating on that data.

Go to Exam Soln 1

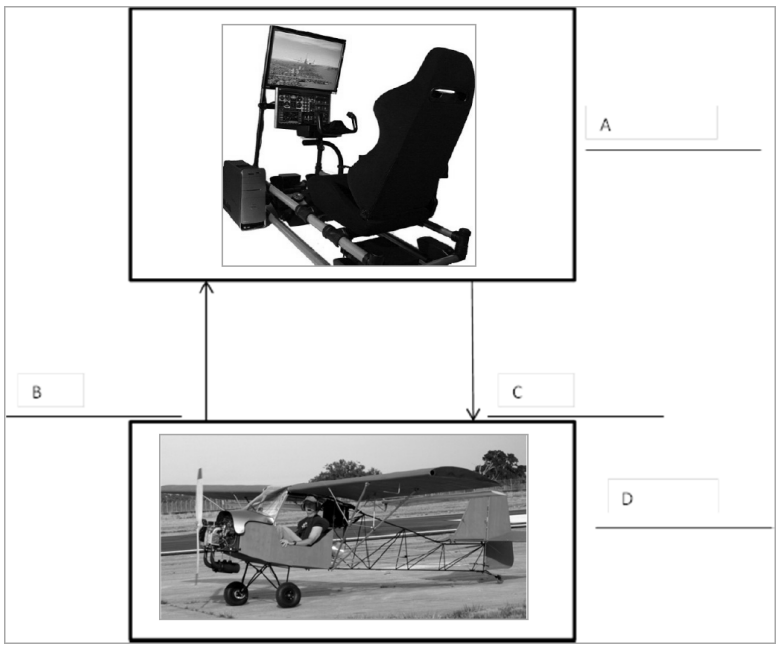
3.3 M269 2014J Exam Soln 1

- A, C

Go to Exam Q 1

3.4 M269 2014J Exam Q 2

- **Question 2** The diagram below shows images of a flight simulator for a single seater aircraft and an actual single seater aircraft. **(2 marks)**



- Complete the diagram above by adding an appropriate label (one of the numbers 1 to 8) in each of the spaces indicated by A, B, C and D. The possible answers are shown as 1 to 8 below.

1 ... represented by

2 ... ignores detail of

3 ... solves

4 ... transforms

5 ... model

6 ... part of reality

7 ... computational problem

8 ... layer

Go to Exam Soln 2

3.5 M269 2014J Exam Soln 2

A 5 model

B 1 represented by

C 2 ignores detail of

D 6 part of reality

- See Unit 1 section 3.2 *Computational thinking and abstraction*

Go to Exam Q 2

3.6 Unit 2 From Problems to Programs

- Unit 2 From Problems to Programs
- Abstract Data Types
- Pre and Post Conditions
- Logic for loops

3.6.1 Example Algorithm Design — Searching

- Given an ordered list (xs) and a value (val), return
  - Position of val in xs or
  - Some indication if val is not present
- *Simple strategy:* check each value in the list in turn
- *Better strategy:* use the ordered property of the list to reduce the range of the list to be searched each turn
  - Set a range of the list
  - If val equals the mid point of the list, return the mid point
  - Otherwise half the range to search
  - If the range becomes negative, report not present (return some distinguished value)

Binary Search Iterative

```
1  def binarySearchIter(xs,val):
2    lo = 0
3    hi = len(xs) - 1
4
5    while lo <= hi:
6      mid = (lo + hi) // 2
7      guess = xs[mid]
8
9      if val == guess:
10       return mid
11     elif val < guess:
12       hi = mid - 1
13     else:
14       lo = mid + 1
15
16   return None
```

Binary Search Recursive

```
1  def binarySearchRec(xs,val,lo=0,hi=-1):
2    if (hi == -1):
3      hi = len(xs) - 1
4
5    mid = (lo + hi) // 2
6
7    if hi < lo:
8      return None
9    else:
10     guess = xs[mid]
```

```
11      if val == guess:
12        return mid
13      elif val < guess:
14        return binarySearchRec(xs,val,lo,mid-1)
15      else:
16        return binarySearchRec(xs,val,mid+1,hi)
```

Binary Search Recursive — Solution

```
xs = [12,16,17,24,41,49,51,62,67,69,75,80,89,97,101]
binarySearchRec(xs, 67)
xs = [12,16,17,24,41,49,51,62,67,69,75,80,89,97,101]
binarySearchRec(xs,67,8,14) by line 15
xs = [12,16,17,24,41,49,51,62,67,69,75,80,89,97,101]
binarySearchRec(xs,67,8,10) by line 13
xs = [12,16,17,24,41,49,51,62,67,69,75,80,89,97,101]
binarySearchRec(xs,67,8,8) by line 13
xs = [12,16,17,24,41,49,51,62,67,69,75,80,89,97,101]
Return value: 8 by line 11
```

Binary Search Iterative — Miller & Ranum

```
1   def binarySearchIterMR(alist, item):
2     first = 0
3     last = len(alist)-1
4     found = False

6     while first<=last and not found:
7       midpoint = (first + last)//2
8       if alist[midpoint] == item:
9         found = True
10      else:
11        if item < alist[midpoint]:
12          last = midpoint-1
13        else:
14          first = midpoint+1

16    return found
```

Miller and Ranum (2011, page 192)

Binary Search Recursive — Miller & Ranum

```
1   def binarySearchRecMR(alist, item):
2     if len(alist) == 0:
3       return False
4     else:
5       midpoint = len(alist)//2
6       if alist[midpoint]==item:
7         return True
8       else:
9         if item<alist[midpoint]:
10          return binarySearchRecMR(alist[:midpoint],item)
11        else:
12          return binarySearchRecMR(alist[midpoint+1:],item)
```

Miller and Ranum (2011, page 193)

3.7 M269 2014J Exam Q 3

- **Question 3** An insertion sort is being carried out on the list of integers shown below, so as to arrange the list in ascending numerical order: **(4 marks)**

start array	54	26	93	17	77	44	55	31

- For the first four passes of the algorithm (after assuming that a list with one item is already sorted), show the order of the items in the list after that pass:

(Pass 1)								
(Pass 2)								
(Pass 3)								
(Pass 4)								

Go to Exam Soln 3

3.8 M269 2014J Exam Soln 3

- The sorted part of the list is filled in pale blue with the next item to be inserted in pale red

(Pass 1)	26	54	93	17	77	44	55	31
(Pass 2)	26	54	93	17	77	44	55	31
(Pass 3)	17	26	54	93	77	44	55	31
(Pass 4)	17	26	54	77	93	44	55	31

Go to Exam Q 3

3.9 M269 2014J Exam Q 4

- **Question 4** Consider the guard in the following Python `while` loop header: **(4 marks)**

```
while (b > 8) and not(a < 6 or b <= 8):
```

(a) Make the following substitutions:

P represents a < 6

Q represents b > 8

complete the following truth table:

P	Q	¬Q	P ∨ ¬Q	¬(P ∨ ¬Q)	Q ∧ ¬(P ∨ ¬Q)
T	T				
T	F				
F	T				
F	F				

(b) Use the results from your truth table to choose which one of the following expressions could be used as a simpler equivalent to the above guard.

A. `(b > 9 and a < 6)`

B. `not (a < 6 or b > 8)`

C. `(a >= 6 and b <= 8)`

D. `not (a < 6 or b <= 8)`

E. `(a < 6 and b <= 8)`

Go to Exam Soln 4

3.10 M269 2014J Exam Soln 4

(a) Truth table

P	Q	¬Q	P ∨ ¬Q	¬(P ∨ ¬Q)	Q ∧ ¬(P ∨ ¬Q)
T	T	F	T	F	F
T	F	T	T	F	F
F	T	F	F	T	T
F	F	T	T	F	F

(b) The only row that has True in the final column for the guard is

P = F and Q = T

- This is equivalent to

  ¬P ∧ Q

- → ¬(P ∨ ¬Q) hence answer D

Go to Exam Q 4

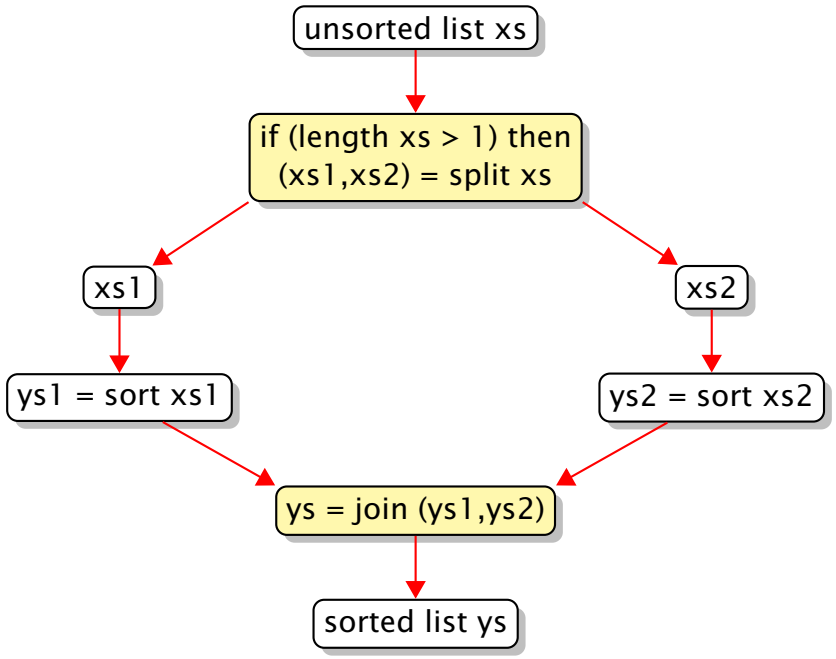
4 Units 3, 4 & 5

4.1 Unit 3 Sorting

- Unit 3 Sorting
- Elementary methods: Bubble sort, Selection sort, Insertion sort

- Recursion — base case(s) and recursive case(s) on smaller data
- Quicksort, Merge sort
- Sorting with data structures: Tree sort, Heap sort
- See sorting notes for abstract sorting algorithm

Abstract Sorting Algorithm



Sorting Algorithms

Using the *Abstract sorting algorithm*, describe the *split* and *join* for:

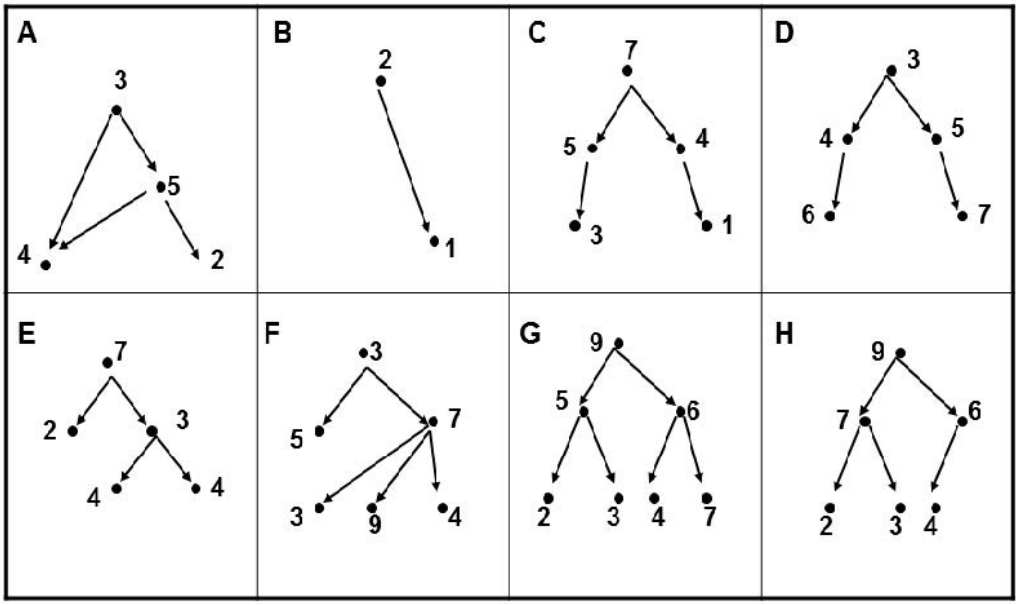
- Insertion sort
- Selection sort
- Merge sort
- Quicksort
- Bubble sort (the odd one out)

4.2 Unit 4 Searching

- Unit 4 Searching
- String searching: Quick search Sunday algorithm, Knuth-Morris-Pratt algorithm
- Hashing and hash tables
- Search trees: Binary Search Trees
- Search trees: Height balanced trees: AVL trees

4.3 M269 2014J Exam Q 5

- **Question 5** Consider the diagrams in A–H. In these diagrams, nodes are represented by black dots and edges by arrows. The numbers are the keys for the corresponding nodes. **(4 marks)**



- Answer the following questions. Write your answer on the line that follows each question. In each case there is at least one diagram in the answer but there may be more than one. Explanations are not required.

(a) Which of A, B, C and D **do not** show trees ?

(b) Which of E, F, G and H are binary trees ?

(c) Which of C, D, G and H are complete binary trees ?

(d) Which of C, D, G and H are binary heaps ?

Go to Exam Soln 5

4.4 M269 2014J Exam Soln 5

(a) In a tree, there is a unique path from the root to each node (graph theory version)

- A is not a tree since 4 can be reached 3–4 or 3–5–4
- B, C, D are trees

(b) In a binary tree, each node has at most two children (graph theory version) Note also that in a binary tree each child node is either a left or a right child. The inductive definition of a binary tree: a binary tree is either an empty tree or a node with two subtrees

- E, G, H are binary trees

(c) In a complete binary tree, every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

- G, H are complete trees.

(d) Binary heaps are complete binary trees that satisfy the heap order property. This property requires that the key of a node is smaller or equal to the key of its children for min heaps or greater or equal for max heap

- H is a binary heap

Go to Exam Q 5

4.5 M269 2014J Exam Q 6

- **Question 6** Consider the following function, which takes an integer n as an argument. You can assume that n is positive. **(4 marks)**

```
1  def twiddle(n):
2      a = 5
3      for i in range(n):
4          for j in range(n):
5              x = i * i
6              y = j * j
7          for k in range(n):
8              z=i*k
9      ans = (x + y + z) * a
10     return ans
```

- From the five options below, select the **one** that represents the correct combination of T(n) and Big-O complexity for this function. You may assume that a step (i.e. the basic unit of computation) is the assignment statement.

A. $T(n) = n^2 + 3n + 2$ and $O(n^2)$

B. $T(n) = 2n^3 + n^2 + 2$ and $O(n^3)$

C. $T(n) = 2n^2 + n + 2$ and $O(2n^2)$

D. $T(n) = 3n^2 + 2$ and $O(n^2)$

E. $T(n) = 2n + 5$ and $O(n)$

- Now explain how you obtained T(n) and the Big-O complexity.

Go to Exam Soln 6

4.6 M269 2014J Exam Soln 6

- D $T(n) = 3n^2 + 2$ and $O(n^2)$
- Explanation:
- 2 assignment statements outside loops
- 3 assignment statements inside two nested loops which are each executed n times ($3n^2$)

Go to Exam Q 6

4.7 M269 2014J Exam Q 7

(a) Given an alphabet of ACGT and the target string CAGAGAG, select the option below that represents the shift table that would be used by the Sunday string search algorithm.

A.

A	C	G	T
1	6	0	8

B.

A	C	G	T
2	1	3	0

C.

A	C	G	T
1	0	2	7

D.

A	C	G	T
2	7	1	8

(b) Assuming a hash table with 20 slots, using the folding method, with 2 as the size of each part, what would be the hash value of the item 1459862913 ?

- In the box below give the hash value and indicate how you calculated your answer, showing all steps.

Go to Exam Soln 7

4.8 M269 2014J Exam Soln 7

(a) D

- The shift table for the Sunday Quick Search algorithm:
  - If the character does not appear in the target string *T*, the shift distance is one more than the length of *T*
  - If the character does appear in *T* the shift distance is the first position at which it appears, counting from right to left and starting at 1

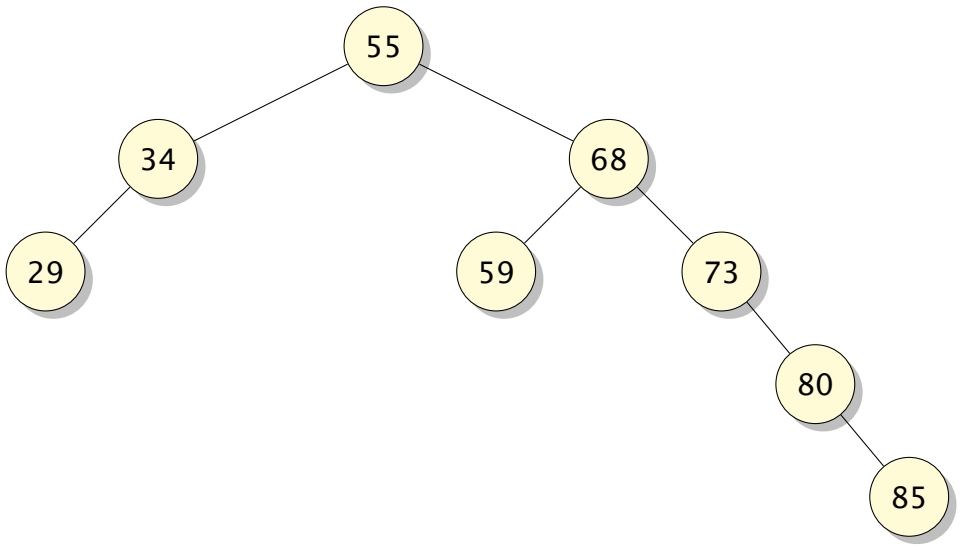
(b) Hash value = 1

since (14 + 59 + 86 + 29 + 13) % 20 = 1

Go to Exam Q 7

4.9 M269 2014J Exam Q 8

- Consider the following Binary Search Tree. **(4 marks)**



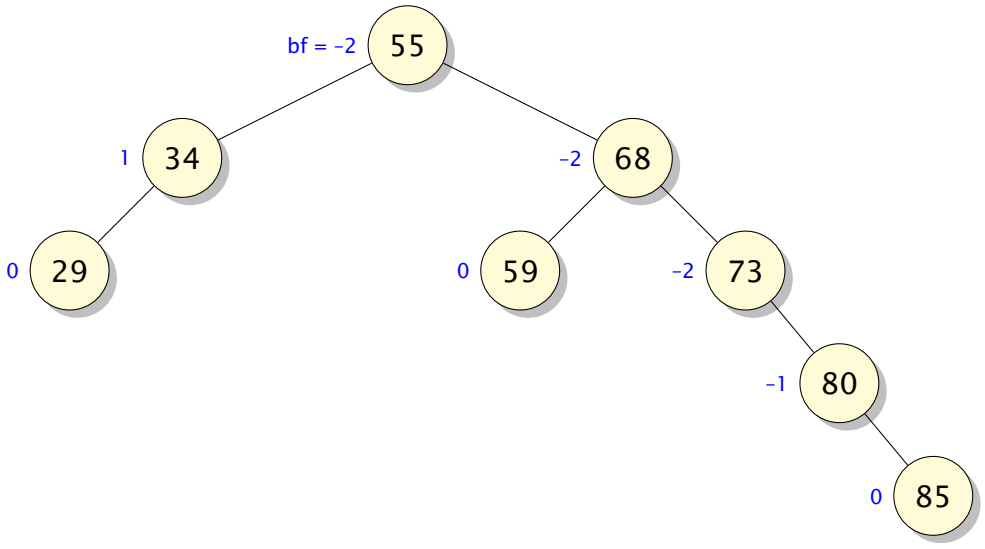
(a) Calculate the balance factors of each node in the above tree and modify the diagram to show these balance factors.

(b) Given your calculated balance factors, would this tree need to be rebalanced to be a valid AVL tree? Give your answer and a brief explanation for your answer in the box provided below.

Go to Exam Soln 8

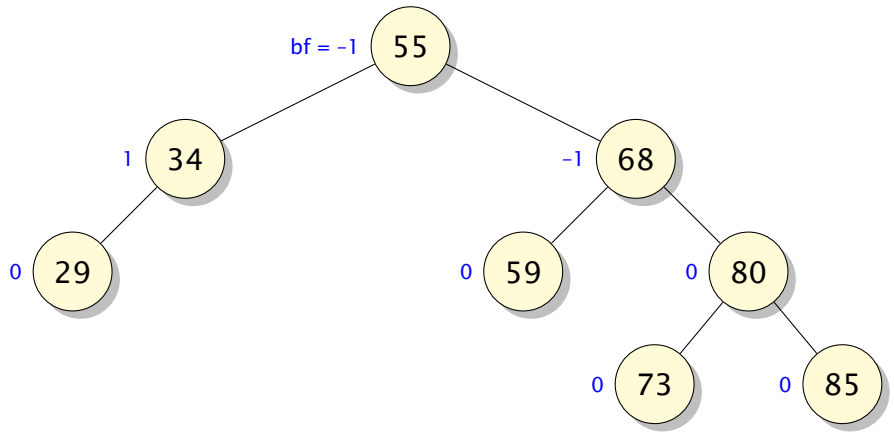
4.10 M269 2014J Exam Soln 8

(a) Tree with balance factors



(b) The tree is not a valid AVL tree since some of the balance factors are outside the range $[-1, 0, +1]$

- The tree must have become unbalanced as a result of inserting 85
- Rebalancing would mean a left rotation around the node 73 as below (this diagram was not required in the exam)



Go to Exam Q 8

4.11 Unit 5 Optimisation

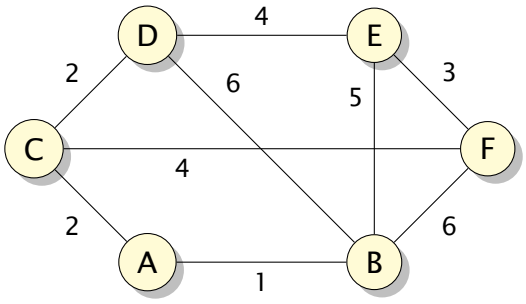
- Unit 5 Optimisation
- Graphs searching: DFS, BFS
- Distance: Dijkstra's algorithm
- Greedy algorithms: Minimum spanning trees, Prim's algorithm
- Dynamic programming: Knapsack problem, Edit distance

4.12 M269 2014J Exam Q 9

- **Question 9** Recall that the structured English for Dijkstra's algorithm is: **(5 marks)**

```
create priority~queue
set dist to 0 for v and dist to infinity
  for all other vertices
add all vertices to priority~queue
ITERATE while priority~queue is not empty
  remove u from the front of the queue
  ITERATE over w in the neighbours of u
    set new~distance to
      dist u + length of edge from u to w
    IF new~distance is less than dist w
      set dist w to new~distance
      change priority(w, new~distance)
```

- Now consider the following weighted graph:



- **Starting from vertex A**, the following table represents the distances from each vertex to A after the second line of structured English is executed for the graph given above (using the convention that the character ∞ represents infinity):

Vertex	A	B	C	D	E	F
Distance	0	∞	∞	∞	∞	∞

- *Note that neither the table above nor the subsequent tables represent the priority queue.*
- Now, complete the appropriate boxes in the next table to show the distances after the first and second iterations of the `while` loop of the algorithm.

Vertex	A	B	C	D	E	F	
Distance	0						First iteration
Distance	0						Second iteration

Go to Exam Soln 9

4.13 M269 2014J Exam Soln 9

- The completed table

Vertex	A	B	C	D	E	F	
Distance	0	1	2	∞	∞	∞	First iteration
Distance	0	1	2	7	6	7	Second iteration

- The node being processed has a red background
- Nodes with a final label have a green background
- Neighbours of the node being processed have blue text

Go to Exam Q 9

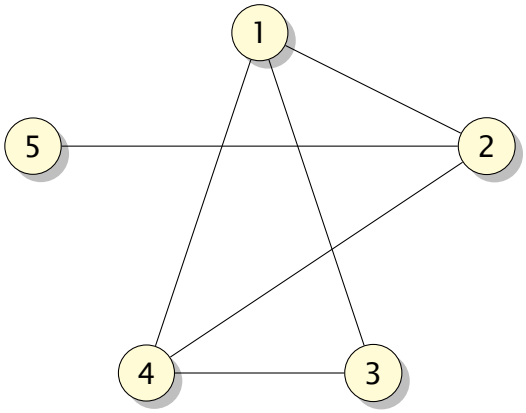
- The complete iterations

Vertex	A	B	C	D	E	F	
Distance	0	1	2	∞	∞	∞	First iteration
Distance	0	1	2	7	6	7	Second iteration
Distance	0	1	2	4	6	6	Third iteration
Distance	0	1	2	4	6	6	Fourth iteration
Distance	0	1	2	4	6	6	Fifth iteration
Distance	0	1	2	4	6	6	Sixth iteration

Go to Exam Q 9

4.14 M269 2014J Exam Q 10

- **Question 10** Consider the following graph: **(4 marks)**



- From the options below, select the **two** which show possible orders in which the vertices of the above graph could be visited in a *Breadth First Search* (BFS) starting at vertex 2:

A.

Vertex	2	3	4	5	1

B.

Vertex	2	1	4	5	3

C.

Vertex	2	1	3	4	5

D.

Vertex	2	5	1	4	3

E.

Vertex	2	5	4	3	1

F.

Vertex	2	1	3	4	5

Go to Exam Soln 10

4.15 M269 2014J Exam Soln 10

- B, D
- A *Breadth First Search* (BFS) from 2 must visit its neighbours in some order first

Go to Exam Q 10

5 Units 6 & 7

5.1 Propositional Logic

M269 Specimen Exam Q11 Topics

- Unit 6
- Sets
- Propositional Logic
- Truth tables
- Valid arguments
- Infinite sets

5.2 M269 2014J Exam Q 11

- **Question 11** **(4 marks)**

(a) What does it mean to say that a well-formed formula (WFF) is a *contradiction* ? Use the space below for your answer.

(b) Is the following WFF a contradiction ?

$(P \wedge (Q \vee \neg Q))$

- Explain how you arrived at your answer in the space below:

Go to Exam Soln 11

5.3 M269 2014J Exam Soln 11

(a) A WFF is a *contradiction* if it is *False* in every *interpretation* — an interpretation is an assignment of meaning to the symbols of a formal language — here it is assignment of truth values to P and Q

(b) Truth table

P	Q	¬Q	Q ∨ ¬Q	(P ∧ (Q ∨ ¬Q))
T	T	F	T	T
T	F	T	T	T
F	T	F	T	F
F	F	T	T	F

- The statement is not a contradiction — it is satisfiable — it has the same truth value as P

Go to Exam Q 11

5.4 Predicate Logic

- Unit 6
- Predicate Logic
- Translation to/from English
- Interpretations

5.5 M269 2014J Exam Q 12

- **Question 12** A particular interpretation of predicate logic allows facts to be expressed about cities and people, in particular, facts about who visited and/or liked which cities. In this interpretation, we will make use of the following two sets:

  (6 marks)

  cities = {Adelaide, San Francisco, Mumbai}

  persons = {Lin, Derren, Gabi}

- Some of the assignments in the interpretation are given below (where the symbol $\mathcal{I}$ is used to show assignment). The interpretation assigns Lin, Derren and Gabi to the constants *lin*, *derren* and *gabi*.
- $\mathcal{I}(lin)$ = Lin $\mathcal{I}(derren)$ = Derren $\mathcal{I}(gabi)$ = Gabi
- The predicates *has_visited* and likes are assigned to binary relations. The comprehensions of the relations are:

  $\mathcal{I}(has\_visited)$ = {(A, B) : the person A has visited the city B}

  $\mathcal{I}(likes)$ = {(A, B) : the person A likes the city B}

- The enumerations of the relations are:

  $\mathcal{I}(has\_visited)$ = {(Lin, Adelaide), (Derren, San Francisco), (Gabi, Mumbai), (Gabi, San Francisco)}

  $\mathcal{I}(likes)$ = {(Lin, Mumbai), (Lin, San Francisco), (Derren, Mumbai), (Derren, San Francisco), (Gabi, Adelaide)}

- You will find parts (a) and (b) of this question on the next page, whilst the interpretation is reproduced on the other side of this page.
- In both parts, you are given a sentence of predicate logic and asked to provide an English translation of the sentence in the box immediately following it.
- You also need to state whether the sentence is TRUE or FALSE in the interpretation that is provided on this page, and give an explanation of your answer.
- In your explanation you need to consider any relevant values for the variables, and *show, using the interpretation above, whether they make the quantified expression TRUE or FALSE*.
- When your explanation refers to the interpretation, make sure that you use formal notation.
- So instead of saying that *Lin likes Mumbai according to the interpretation*, write:

  (Lin, Mumbai) $\in \mathcal{I}(likes)$.
- Similarly, instead of *Lin doesn't like Mumbai* you would need to write:

  (Lin, Mumbai) $\notin \mathcal{I}(likes)$.
- **Interpretation to be used for answering Question 12**
- cities = {Adelaide, San Francisco, Mumbai}
- persons = {Lin, Derren, Gabi}
- $\mathcal{I}(lin)$ = Lin $\mathcal{I}(derren)$ = Derren $\mathcal{I}(gabi)$ = Gabi
- $\mathcal{I}(has\_visited)$ = {(Lin, Adelaide), (Derren, San Francisco), (Gabi, Mumbai), (Gabi, San Francisco)}
- $\mathcal{I}(likes)$ = {(Lin, Mumbai), (Lin, San Francisco), (Derren, Mumbai), (Derren, San Francisco), (Gabi, Adelaide)}

(a) $\exists X.\neg(likes(lin, X) \wedge likes(derren, X))$

can be translated into English as:

- This sentence is ____ (choose from TRUE/FALSE), because:

(b) $\forall X.(likes(gabi, X) \vee has\_visited(gabi, X))$

can be translated into English as:

- This sentence is ____ (choose from TRUE/FALSE), because:

Go to Exam Soln 12

5.6 M269 2014J Exam Soln 12

(a) $\exists X.\neg(likes(lin, X) \wedge likes(derren, X))$

- There is a city that is not liked by both Lin and Derren
- *True* since X could be *Adelaide*

(b) $\forall X.(likes(gabi, X) \vee has\_visited(gabi, X))$

- *Gabi* either likes or has visited all cities
- *True* since *Gabi* likes *Adelaide* and has visited the others.

Go to Exam Q 12

5.7 SQL Queries

M269 Specimen Exam Q13 Topics

- Unit 6
- SQL queries

5.8 M269 2014J Exam Q 13

- **Question 13** A database contains the following tables, *production_line* and *product*. **(6 marks)**

production_line	
id	*unit*
Line1	Sportscar
Line2	SUV
Line3	Bus
Line4	Tractor
Line5	Aeroplane

product	
type	*unit_price*
SUV	50000
Sportscar	200000
Bus	250000
Tractor	50000
Aeroplane	30000000

(a) For the following SQL query, give the table returned by the query.

```
SELECT id, unit_price
FROM production_line CROSS JOIN product
WHERE unit = type AND unit_price < 300000;
```

(b) Write an SQL query which answers the question *Which products cost exactly 50000?* The answer should be the following table:

type
SUV
Tractor

Go to Exam Soln 13

5.9 M269 2014J Exam Soln 13

(a) The output table

id	*unit_price*
Line1	200000
Line2	50000
Line3	250000
Line4	50000

(b) SQL query

```
SELECT type
FROM product
WHERE unit_price = 50000;
```

Go to Exam Q 13

5.10 Logic

M269 Exam — Q14 topics

- Unit 7
- Proofs
- Natural deduction

Logicians, Logics, Notations

- A plethora of logics, proof systems, and different notations can be puzzling.
- Martin Davis, Logician When I was a student, even the topologists regarded mathematical logicians as living in outer space. Today the connections between logic and computers are a matter of engineering practice at every level of computer organization
- Various logics, proof systems , were developed well before programming languages and with different motivations,

References: Davis (1995, page 289)

Logic and Programming Languages

- Turing machines, Von Neumann architecture and procedural languages Fortran, C, Java, Perl, Python, JavaScript
- Resolution theorem proving and logic programming — Prolog

- Logic and database query languages — SQL (Structured Query Language) and QBE (Query-By-Example) are syntactic sugar for first order logic
- Lambda calculus and functional programming with Miranda, Haskell, ML, Scala

Reference: Halpern et al. (2001)

Validity and Justification

- There are two ways to model what counts as a logically good argument:
  - the **semantic** view
  - the **syntactic** view
- The notion of a valid argument in propositional logic is rooted in the semantic view.
- It is based on the semantic idea of interpretations: assignments of truth values to the propositional variables in the sentences under discussion.
- A *valid argument* is defined as one that preserves truth from the premises to the conclusions
- The syntactic view focuses on the syntactic form of arguments.
- Arguments which are correct according to this view are called *justified arguments*.

Proof Systems, Soundness, Completeness

- Semantic validity and syntactic justification are different ways of modelling the same intuitive property: whether an argument is logically good.
- A proof system is *sound* if any statement we can prove (justify) is also valid (true)
- A proof system is *adequate* if any valid (true) statement has a proof (justification)
- A proof system that is sound and adequate is said to be *complete*
- Propositional and predicate logic are *complete* — arguments that are valid are also justifiable and vice versa
- Unit 7 section 2.4 describes another logic where there are valid arguments that are not justifiable (provable)

Reference: Chiswell and Hodges (2007, page 86)

Valid arguments

- Unit 6 defines valid arguments with the notation $\begin{array}{l} P_1 \\ \vdots \\ P_n \\ \hline C \end{array}$
- The argument is *valid* if and only if the value of C is *True* in each interpretation for which the value of each premise P_i is *True* for $1 \leq i \leq n$
- In some texts you see the notation $\{P_1, \ldots, P_n\} \models C$
- The expression denotes a *semantic sequent* or *semantic entailment*

- The $\models$ symbol is called the *double turnstile* and is often read as *entails* or *models*
- In LaTeX $\vDash$ and $\models$ are produced from `\vDash` and `\models` — see also the *turnstile* package
- In Unicode $\models$ is called *TRUE* and is U+22A8, HTML ⊨
- The argument $\{\} \models C$ is valid if and only if C is *True* in all interpretations
- That is, if and only if C is a tautology
- Beware different notations that mean the same thing
  - Alternate symbol for empty set: $\varnothing \models C$
  - Null symbol for empty set: $\models C$
  - Original M269 notation with null axiom above the line:

    $\overline{C}$

Justified Arguments and Natural Deduction

- Definition 7.1 An argument $\{P_1, P_2, \ldots, P_n\} \vdash C$ is a justified argument if and only if either the argument is an instance of an axiom or it can be derived by means of an inference rule from one or more other justified arguments.
- Axioms

$$\Gamma \cup \{\mathbf{A}\} \vdash \mathbf{A} \text{ (axiom schema)}$$

- This can be read as: any formula **A** can be derived from the assumption (premise) of {**A**} itself
- The $\vdash$ symbol is called the *turnstile* and is often read as *proves*, denoting *syntactic entailment*
- In LaTeX $\vdash$ is produced from `\vdash`
- In Unicode $\vdash$ is called *RIGHT TACK* and is U+22A2, HTML ⊢

See (Thompson, 1991, Chp 1)

- Section 2.3 of Unit 7 (not the Unit 6, 7 Reader) gives the inference rules for $\rightarrow$, $\wedge$, and $\vee$ — only dealing with *positive propositional logic* so not making use of negation — see List of logic systems
- Usually (Classical logic) have a functionally complete set of logical connectives — that is, every binary Boolean function can be expressed in terms the functions in the set

Inference Rules — Notation

- Inference rule notation:

$$\frac{Argument_1 \quad \ldots \quad Argument_n}{Argument}\ (label)$$

Inference Rules — Conjunction

- $\dfrac{\Gamma \vdash \mathbf{A} \quad \Gamma \vdash \mathbf{B}}{\Gamma \vdash \mathbf{A} \wedge \mathbf{B}}$ ($\wedge$-introduction)
- $\dfrac{\Gamma \vdash \mathbf{A} \wedge \mathbf{B}}{\Gamma \vdash \mathbf{A}}$ ($\wedge$-elimination left)
- $\dfrac{\Gamma \vdash \mathbf{A} \wedge \mathbf{B}}{\Gamma \vdash \mathbf{B}}$ ($\wedge$-elimination right)

Inference Rules — Implication

- $\dfrac{\Gamma \cup \{\mathbf{A}\} \vdash \mathbf{B}}{\Gamma \vdash \mathbf{A} \to \mathbf{B}}$ ($\to$-introduction)
- The above should be read as: If there is a proof (justification, inference) for **B** under the set of premises, Γ, augmented with **A**, then we have a proof (justification. inference) of $\mathbf{A} \to \mathbf{B}$, *under the unaugmented set of premises,* Γ.

  The unaugmented set of premises, Γ may have contained **A** already so we cannot assume

  $$(\Gamma \cup \{\mathbf{A}\}) - \{\mathbf{A}\} \text{ is equal to } \Gamma$$

- $\dfrac{\Gamma \vdash \mathbf{A} \quad \Gamma \vdash \mathbf{A} \to \mathbf{B}}{\Gamma \vdash \mathbf{B}}$ ($\to$-elimination)

Inference Rules — Disjunction

- $\dfrac{\Gamma \vdash \mathbf{A}}{\Gamma \vdash \mathbf{A} \vee \mathbf{B}}$ ($\vee$-introduction left)
- $\dfrac{\Gamma \vdash \mathbf{B}}{\Gamma \vdash \mathbf{A} \vee \mathbf{B}}$ ($\vee$-introduction right)
- Disjunction elimination

$$\frac{\Gamma \vdash \mathbf{A} \vee \mathbf{B} \quad \Gamma \cup \{\mathbf{A}\} \vdash \mathbf{C} \quad \Gamma \cup \{\mathbf{B}\} \vdash \mathbf{C}}{\Gamma \vdash \mathbf{C}} \text{ ($\vee$-elimination)}$$

- The above should be read: if a set of premises Γ justifies the conclusion $\mathbf{A} \vee \mathbf{B}$ and Γ augmented with each of **A** or **B** separately justifies **C**, then Γ justifies **C**

Proofs in Tree Form

- The syntax of proofs is recursive:
- A proof is either an axiom, or the result of applying a rule of inference to one, two or three proofs.
- We can therefore represent a proof by a tree diagram in which each node have one, two or three children
- For example, the proof of $\{P \wedge (P \to Q)\} \vdash Q$ in *Question 4* (in the Logic tutorial notes) can be represented by the following diagram:

$$\dfrac{\dfrac{\{P \wedge (P \to Q)\} \vdash P \wedge (P \to Q)}{\{P \wedge (P \to Q)\} \vdash P} \text{ ($\wedge$-E left)} \quad \dfrac{\{P \wedge (P \to Q)\} \vdash P \wedge (P \to Q)}{\{P \wedge (P \to Q)\} \vdash P \to Q} \text{ ($\wedge$-E right)}}{\{P \wedge (P \to Q)\} \vdash Q} \text{ ($\to$-E)}$$

Self-Assessment activity 7.4 — tree layout

- Let $\Gamma = \{P \to R, Q \to R, P \vee Q\}$
- $$\frac{\Gamma \vdash P \vee Q \quad \Gamma \cup \{P\} \vdash R \quad \Gamma \cup \{Q\} \vdash R}{\Gamma \vdash R}\ (\vee\text{-elimination})$$
- $$\frac{\Gamma \cup \{P\} \vdash P \quad \Gamma \cup \{P\} \vdash P \to R}{\Gamma \cup \{P\} \vdash R}\ (\to\text{-elimination})$$
- $$\frac{\Gamma \cup \{Q\} \vdash Q \quad \Gamma \cup \{Q\} \vdash Q \to R}{\Gamma \cup \{Q\} \vdash R}\ (\to\text{-elimination})$$
- Complete tree layout
- $$\dfrac{\Gamma \vdash P \vee Q \qquad \dfrac{\dfrac{\Gamma \cup \{P\}}{\vdash P} \quad \dfrac{\Gamma \cup \{P\}}{\vdash P \to R}\ (\to\text{-E})}{\Gamma \cup \{P\} \vdash R} \qquad \dfrac{\dfrac{\Gamma \cup \{Q\}}{\vdash Q} \quad \dfrac{\Gamma \cup \{Q\}}{\vdash Q \to R}\ (\to\text{-E})}{\Gamma \cup \{Q\} \vdash R}}{\Gamma \vdash R}\ (\vee\text{-E})$$

Self-assessment activity 7.4 — Linear Layout

1. $\{P \to R, Q \to R, P \vee Q\} \vdash P \vee Q$ [Axiom]
2. $\{P \to R, Q \to R, P \vee Q\} \cup \{P\} \vdash P$ [Axiom]
3. $\{P \to R, Q \to R, P \vee Q\} \cup \{P\} \vdash P \to R$ [Axiom]
4. $\{P \to R, Q \to R, P \vee Q\} \cup \{Q\} \vdash Q$ [Axiom]
5. $\{P \to R, Q \to R, P \vee Q\} \cup \{Q\} \vdash Q \to R$ [Axiom]
6. $\{P \to R, Q \to R, P \vee Q\} \cup \{P\} \vdash R$ [2, 3, $\to$-E]
7. $\{P \to R, Q \to R, P \vee Q\} \cup \{Q\} \vdash R$ [4, 5, $\to$-E]
8. $\{P \to R, Q \to R, P \vee Q\} \vdash R$ [1, 6, 7, $\vee$-E]

5.11 M269 2014J Exam Q 14

- **Question 14** Consider the following axiom schema and rules: **(4 marks)**

Axiom schema	$\{\mathbf{A}\} \vdash \mathbf{A}$
Rules	$\dfrac{\Gamma \vdash \mathbf{A} \wedge \mathbf{B}}{\Gamma \vdash \mathbf{A}}$ ($\wedge$-elimination left)
	$\dfrac{\Gamma \vdash \mathbf{A} \wedge \mathbf{B}}{\Gamma \vdash \mathbf{B}}$ ($\wedge$-elimination right)
	$\dfrac{\Gamma \vdash \mathbf{A} \quad \Gamma \vdash \mathbf{B}}{\Gamma \vdash \mathbf{A} \wedge \mathbf{B}}$ ($\wedge$-introduction)
	$\dfrac{\Gamma \cup \{\mathbf{A}\} \vdash \mathbf{B}}{\Gamma \vdash \mathbf{A} \to \mathbf{B}}$ ($\to$-introduction)
	$\dfrac{\Gamma \vdash \mathbf{A} \quad \Gamma \vdash \mathbf{A} \to \mathbf{B}}{\Gamma \vdash \mathbf{B}}$ ($\to$-elimination)

- Complete the following proof by filling in the two boxes. You can use any of the above as appropriate.

1. $\{(V \wedge W)\} \vdash (V \wedge W)$ [Axiom schema]
2. ?? ?? [1 $\wedge$-elimination right]
3. $\emptyset \vdash (V \wedge W) \to W$?? ??

Go to Exam Soln 14

5.12 M269 2014J Exam Soln 14

- Completed proof
  1. $\{(V \wedge W)\} \vdash (V \wedge W)$ [Axiom schema]
  2. $\{(V \wedge W)\} \vdash W$ [1 $\wedge$-elimination right]
  3. $\emptyset \vdash (V \wedge W) \rightarrow W$ 2 $\rightarrow$-introduction
- Note $\emptyset$ is a symbol for the empty set (in LaTeX `\varnothing`)
- You could also use {} (or even leave a blank, but that would not be good practice)

Go to Exam Q 14

5.13 Computability

M269 Specimen Exam — Q15 Topics

- Unit 7
- Computability and ideas of computation
- Complexity
- P and NP
- NP-complete

Ideas of Computation

- The idea of an algorithm and what is effectively computable
- **Church-Turing thesis** Every function that would naturally be regarded as computable can be computed by a deterministic Turing Machine. (Unit 7 Section 4)
- See Phil Wadler on computability theory performed as part of the Bright Club at The Strand in Edinburgh, Tuesday 28 April 2015

Reducing one problem to another

- To reduce problem P_1 to P_2, invent a construction that converts instances of P_1 to P_2 that have the same answer. That is:
  - any string in the language P_1 is converted to some string in the language P_2
  - any string over the alphabet of P_1 that is not in the language of P_1 is converted to a string that is not in the language P_2
- With this construction we can solve P_1
  - Given an instance of P_1, that is, given a string w that may be in the language P_1, apply the construction algorithm to produce a string x
  - Test whether x is in P_2 and give the same answer for w in P_1

(Hopcroft et al., 2007, page 322)

- The direction of reduction is important
- If we can reduce P_1 to P_2 then (in some sense) P_2 is at least as hard as P_1 (since a solution to P_2 will give us a solution to P_1)
- So, if P_2 is decidable then P_1 is decidable
- To show a problem is undecidable we have to reduce from an known undecidable problem to it
- $\forall x(dp_{P_1}(x) = dp_{P_2}(reduce(x)))$
- Since, if P_1 is undecidable then P_2 is undecidable

Computability — Models of Computation

- In automata theory, a *problem* is the question of deciding whether a given string is a member of some particular language
- If Σ is an alphabet, and L is a language over Σ, that is $L \subseteq \Sigma^*$, where Σ^* is the set of all strings over the alphabet Σ then we have a more formal definition of *decision problem*
- Given a string $w \in \Sigma^*$, decide whether $w \in L$
- Example: Testing for a prime number — can be expressed as the language L_p consisting of all binary strings whose value as a binary number is a prime number (only divisible by 1 or itself)

(Hopcroft et al., 2007, section 1.5.4)

Computability — Church-Turing Thesis

- **Church-Turing thesis** Every function that would naturally be regarded as computable can be computed by a deterministic Turing Machine.
- **physical Church-Turing thesis** Any finite physical system can be simulated (to any degree of approximation) by a Universal Turing Machine.
- **strong Church-Turing thesis** Any finite physical system can be simulated (to any degree of approximation) with polynomial slowdown by a Universal Turing Machine.
- Shor's algorithm (1994) — quantum algorithm for factoring integers — an NP problem that is not known to be P — also not known to be NP-complete and we have no proof that it is not in P

Reference: Section 4 of Unit 6 & 7 Reader

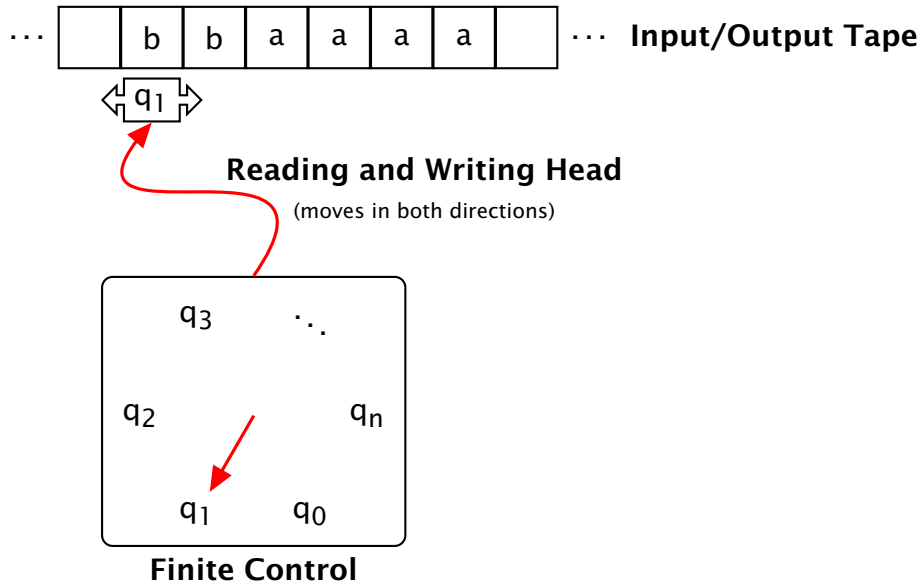
Computability — Turing Machine

- **Finite control** which can be in any of a finite number of *states*
- **Tape** divided into cells, each of which can hold one of a finite number of symbols
- Initially, the **input**, which is a finite-length string of symbols in the *input alphabet*, is placed on the tape

- All other tape cells (extending infinitely left and right) hold a special symbol called *blank*
- A **tape head** which initially is over the leftmost input symbol
- A **move** of the Turing Machine depends on the state and the tape symbol scanned
- A move can change state, write a symbol in the current cell, move left, right or stay

References: Hopcroft et al. (2007, page 326), Unit 6 & 7 Reader (section 5.3)

Turing Machine Diagram



Source: Sebastian Sardina http://www.texample.net/tikz/examples/turing-machine-2/
Date: 18 February 2012 (seen Sunday, 24 August 2014)
Further Source: Partly based on Ludger Humbert's pics of Universal Turing Machine at https://haspe.homeip.net/projekte/ddi/browser/tex/pgf2/turingmaschine-schema.tex (not found) — http://www.texample.net/tikz/examples/turing-machine/

Turing Machine notation

- Q finite set of states of the finite control
- Σ finite set of *input symbols* (M269 S)
- Γ complete set of *tape symbols* $\Sigma \subset \Gamma$
- δ Transition function (M269 instructions, I)
  $\delta :: Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$
  $\delta(q, X) \mapsto (p, Y, D)$
- $\delta(q, X)$ takes a state, q and a tape symbol, X and returns (p, Y, D) where p is a state, Y is a tape symbol to overwrite the current cell, D is a direction, Left, Right or Stay
- q_0 *start state* $q_0 \in Q$
- B *blank symbol* $B \in \Gamma$ and $B \notin \Sigma$
- F set of *final or accepting states* $F \subseteq Q$

Computability — Decidability

- **Decidable** — there is a TM that will halt with yes/no for a decision problem — that is, given a string w over the alphabet of P the TM with halt and return yes.no the string is in the language P (same as *recursive* in Recursion theory — old use of the word)
- **Semi-decidable** — there is a TM will halt with yes if some string is in P but may loop forever on some inputs (same as *recursively enumerable*) — *Halting Problem*
- **Highly-undecidable** — no outcome for any input — *Totality, Equivalence Problems*

Undecidable Problems

- **Halting problem** — the problem of deciding, given a program and an input, whether the program will eventually halt with that input, or will run forever — term first used by Martin Davis 1952
- **Entscheidungsproblem** — the problem of deciding whether a given statement is provable from the axioms using the rules of logic — shown to be undecidable by Turing (1936) by reduction from the *Halting problem* to it
- **Type inference and type checking** in the second-order lambda calculus (important for functional programmers, Haskell, GHC implementation)
- Undecidable problem — see link to list

(Turing, 1936, 1937)

Why undecidable problems must exist

- A *problem* is really membership of a string in some language
- The number of different languages over any alphabet of more than one symbol is uncountable
- Programs are finite strings over a finite alphabet (ASCII or Unicode) and hence countable.
- There must be an infinity (big) of problems more than programs.

Reference: Hopcroft et al. (2007, page 318)

Computability and Terminology

- The idea of an *algorithm* dates back 3000 years to Euclid, Babylonians...
- In the 1930s the idea was made more formal: which *functions* are *computable*?
- A *function* a set of pairs $f = \{(x, f(x)) : x \in X \wedge f(x) \in Y\}$ with the *function property*
- *Function property:* $(a, b) \in f \wedge (a, c) \in f \Rightarrow b == c$
- *Function property:* Same input implies same output
- Note that maths notation is deeply inconsistent here — see Function and History of the function concept

- *What do we mean by computing a function — an algorithm ?*
- In the 1930s three definitions:
- λ-Calculus, simple semantics for computation — Alonzo Church
- General recursive functions — Kurt Gödel
- Universal (Turing) machine — Alan Turing
- Terminology:
  - Recursive, recursively enumerable — Church, Kleene
  - Computable, computably enumerable — Gödel, Turing
  - Decidable, semi-decidable, highly undecidable
  - In the 1930s, *computers* were *human*
  - Unfortunate choice of terminology
- Turing and Church showed that the above three were equivalent
- Church-Turing thesis — function is intuitively computable if and only if Turing machine computable

Sources on Computability Terminology

- Soare (1996) on the history of the terms *computable* and *recursive* meaning *calculable*
- See also Soare (2013, sections 9.9–9.15) in Copeland et al. (2013)

5.14 M269 2014J Exam Q 15

(a) Complete the following sentence in the box below:

The statement *If a computational problem is in NP, then it must be intractable* may be false because

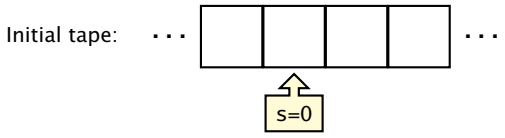
(b) Consider the following Turing Machine:

	$	EMPTY
0		WRITE $ MOVE RIGHT NEXT STATE 1
1	WRITE blank MOVE LEFT NEXT STATE 0	WRITE $ MOVE RIGHT NEXT STATE 0

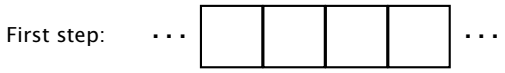
- Assume that the starting state is 0 and that the input tape consists of empty squares.
- One square is marked as the current square with the tape head, shown here as

where n is the current state

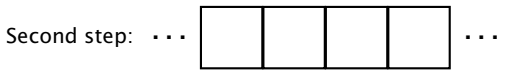
- So, initially the tape looks as follows:



- Note it the original exam the tape cells and head are denoted by ASCII symbols
- Using the same notation, write down what the tape looks like after each of the next two steps of the computation.
- Use the boxes below for this.
- After the first step



- After the second step

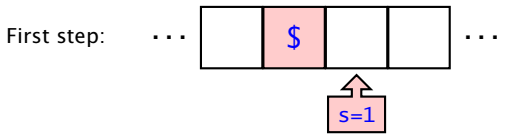


Go to Exam Soln 15

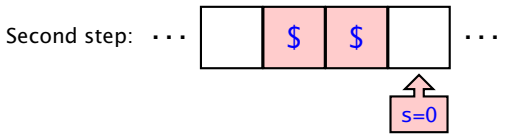
5.15 M269 2014J Exam Soln 15

(a) P is a subset of NP but we do not know if it is a proper subset — so the problem may be in P

(b) After the first step



- After the second step



Go to Exam Q 15

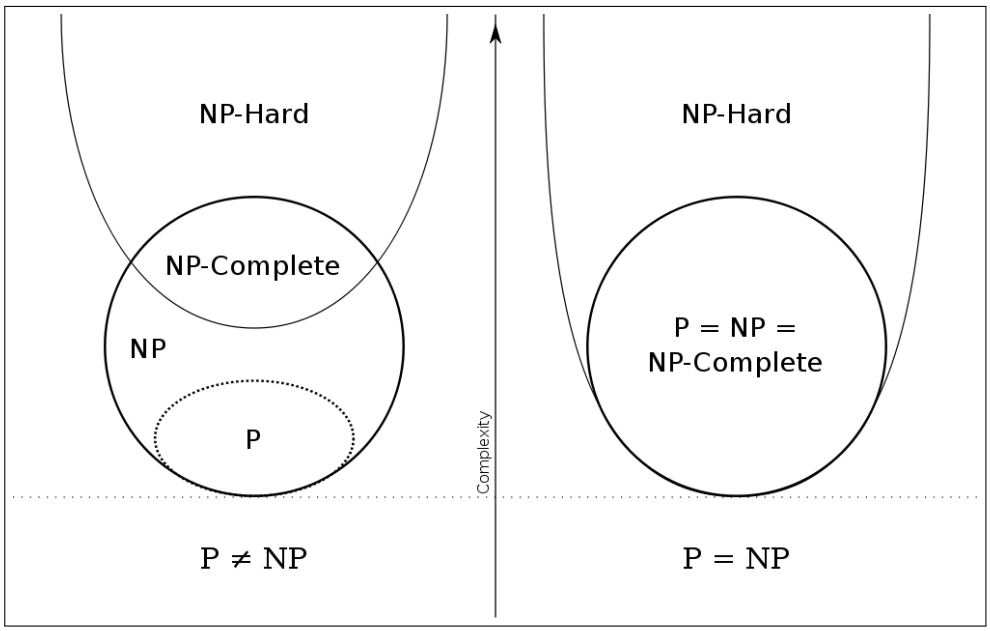
5.16 Complexity

P and NP

- P, the set of all decision problems that can be solved in polynomial time on a deterministic Turing machine
- NP, the set of all decision problems whose solutions can be verified (certificate) in polynomial time

- Equivalently, NP, the set of all decision problems that can be solved in polynomial time on a non-deterministic Turing machine
- A *decision problem*, *dp* is *NP-complete* if
  1. *dp* is in NP and
  2. Every problem in NP is reducible to *dp* in polynomial time
- *NP-hard* — a problem satisfying the second condition, whether or not it satisfies the first condition. Class of problems which are at least as hard as the hardest problems in NP. NP-hard problems do not have to be in NP and may not be decision problems

Euler diagram for P, NP, NP-complete and NP-hard set of problems

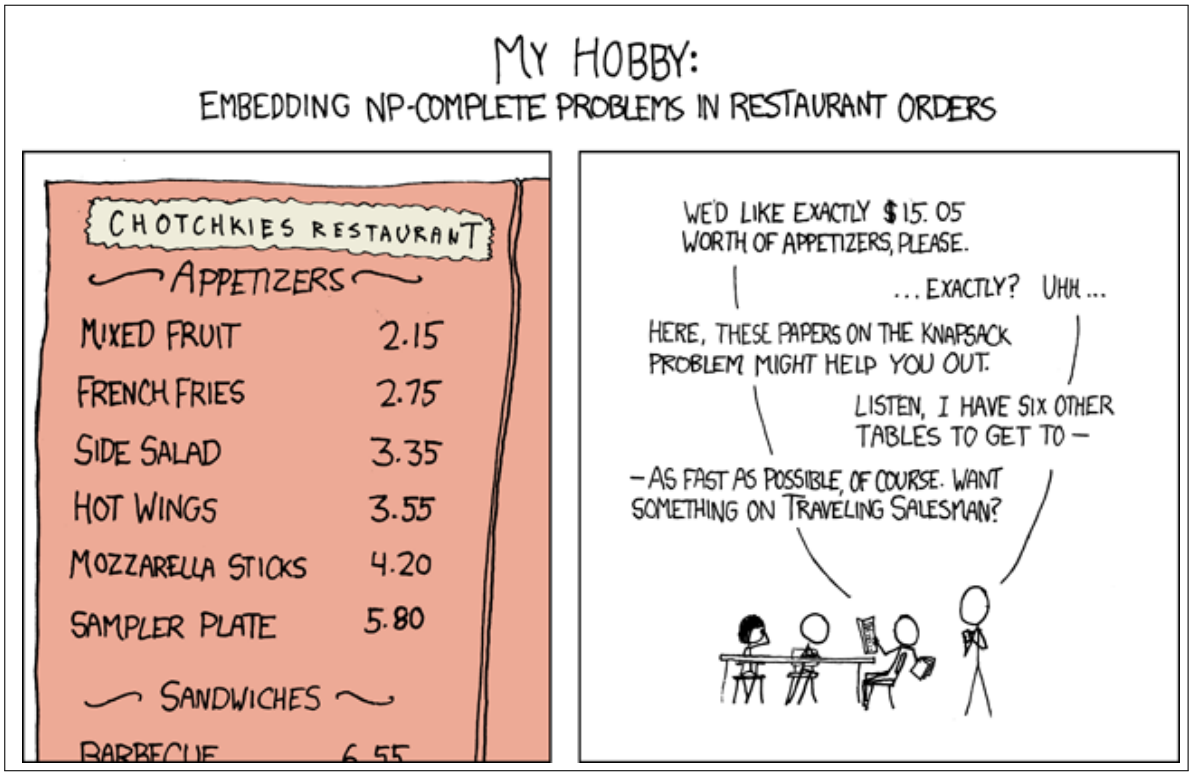


Source: Wikipedia NP-complete entry

NP-complete problems

- Boolean satisfiability (SAT) Cook-Levin theorem
- Conjunctive Normal Form 3SAT
- Hamiltonian path problem
- Travelling salesman problem
- NP-complete — see list of problems

XKCD on NP-Complete Problems



Source & Explanation: XKCD 287

5.16.1 NP-Completeness and Boolean Satisfiability

- The *Boolean satisfiability problem (SAT)* was the first decision problem shown to be *NP-Complete*
- This section gives a sketch of an explanation
- **Health Warning** different texts have different notations and there will be some inconsistency in these notes
- **Health warning** these notes use some formal notation *to make the ideas more precise* — computation requires precise notation and is about manipulating strings according to precise rules.

Alphabets, Strings and Languages

- Notation:
- Σ is a set of symbols — the alphabet
- Σ^k is the set of all string of length k, which each symbol from Σ
- Example: if $\Sigma = \{0, 1\}$
  - $\Sigma^1 = \{0, 1\}$
  - $\Sigma^2 = \{00, 01, 10, 11\}$
- $\Sigma^0 = \{\epsilon\}$ where ϵ is the empty string

- Σ^* is the set of all possible strings over Σ
- $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$
- A *Language*, L, over Σ is a subset of Σ^*
- $L \subseteq \Sigma^*$

Language Accepted by a Turing Machine

- Language accepted by Turing Machine, M denoted by L(M)
- L(M) is the set of strings $w \in \Sigma^*$ accepted by M
- For *Final States* F = {Y, N}, a string $w \in \Sigma^*$ is accepted by M $\Leftrightarrow$ (if and only if) M starting in q_0 with w on the tape halts in state Y
- Calculating a function (function problem) can be turned into a decision problem by asking whether f(x) = y

The NP-Complete Class

- If we do not know if P $\neq$ NP, what can we say ?
- A language L is *NP-Complete* if:
  - $L \in$ NP and
  - for all other $L' \in$ NP there is a *polynomial time transformation* (Karp reducible, reduction) from L' to L
- Problem P_1 *polynomially reduces* (Karp reduces, transforms) to P_2, written $P_1 \propto P_2$ or $P_1 \leq_p P_2$, iff $\exists f : dp_{P_1} \rightarrow dp_{P_2}$ such that
  - $\forall I \in dp_{P_1} [I \in Y_{P_1} \Leftrightarrow f(I) \in Y_{P_2}]$
  - f can be computed in polynomial time
- More formally, $L_1 \subseteq \Sigma_1^*$ polynomially transforms to $L_2 \subseteq \Sigma_2^*$, written $L_1 \propto L_2$ or $L_1 \leq_p L_2$, iff $\exists f : \Sigma_1^* \rightarrow \Sigma_2^*$ such that
  - $\forall x \in \Sigma_1^* [x \in L_1 \Leftrightarrow f(x) \in L_2]$
  - There is a polynomial time TM that computes f
- *Transitivity* If $L_1 \propto L_2$ and $L_2 \propto L_3$ then $L_1 \propto L_3$
- If L is NP-Hard and $L \in$ P then P = NP
- If L is NP-Complete, then $L \in$ P if and only if P = NP
- If L_0 is NP-Complete and $L \in$ NP and $L_0 \propto L$ then L is NP-Complete
- Hence if we find one NP-Complete problem, it may become easier to find more
- In 1971/1973 Cook-Levin showed that the Boolean satisfiability problem (SAT) is NP-Complete

The Boolean Satisfiability Problem

- A propositional logic formula or Boolean expression is built from variables, operators: AND (conjunction, $\wedge$), OR (disjunction, $\vee$), NOT (negation, $\neg$)
- A formula is said to be *satisfiable* if it can be made True by some assignment to its variables.
- *The Boolean Satisfiability Problem* is, given a formula, check if it is satisfiable.
  - *Instance:* a finite set U of Boolean variables and a finite set C of clauses over U
  - *Question:* Is there a satisfying truth assignment for C ?
- A *clause* is is a disjunction of variables or negations of variables
- *Conjunctive normal form (CNF)* is a conjunction of clauses
- Any Boolean expression can be transformed to CNF
- Given a set of Boolean variable $U = \{u_1, u_2, \ldots, u_n\}$
- A literal from U is either any u_i or the negation of some u_i (written $\overline{u_i}$)
- A clause is denoted as a subset of literals from U — $\{u_2, \overline{u_4}, u_5\}$
- A clause is satisfied by an assignment to the variables if at least one of the literals evaluates to True (just like disjunction of the literals)
- Let C be a set of clauses over U — C is satisfiable iff there is some assignment of truth values to the variables so that every clause is satisfied (just like CNF)
- $C = \{\{u_1, u_2, u_3\}, \{\overline{u_2}, \overline{u_3}\}, \{u_2, \overline{u_3}\}\}$ is satisfiable
- $C = \{\{u_1, u_2\}, \{u_1, \overline{u_2}\}, \{\overline{u_1}\}\}$ is not satisfiable
- Proof that SAT is NP-Complete looks at the structure of NDTMs and shows you can transform any NDTM to SAT in polynomial time (in fact logarithmic space suffices)
- SAT is in NP since you can check a solution in polynomial time
- To show that $\forall L \in NP : L \propto SAT$ invent a polynomial time algorithm for each polynomial time NDTM, M, which takes as input a string x and produces a Boolean formula E_x which is satisfiable iff M accepts x
- See Cook-Levin theorem

Sources

- Garey and Johnson (1979, page 34) has the notation $L_1 \propto L_2$ for polynomial transformation
- Arora and Barak (2009, page 42) has the notation $L_1 \leq_p L_2$ for *polynomial-time Karp reducible*
- The sketch of Cook's theorem is from Garey and Johnson (1979, page 38)
- For the satisfiable C we could have assignments $(u_1, u_2, u_3) \in \{(T, T, F), (T, F, F), (F, T, F)\}$

Coping with NP-Completeness

- What does it mean if a problem is NP-Complete ?
  - There is a P time verification algorithm.
  - There is a P time algorithm to solve it iff P = NP (?)
  - No one has yet found a P time algorithm to solve any NP-Complete problem
  - So what do we do ?
- Improved exhaustive search — Dynamic Programming; Branch and Bound
- Heuristic methods — *acceptable* solutions in *acceptable* time — compromise on optimality
- Average time analysis — look for an algorithm with good average time — compromise on generality (see Big-O Algorithm Complexity Cheatsheet)
- Probabilistic or Randomized algorithms — compromise on correctness

Sources

- *Practical Solutions for Hard Problems* Rich (2007, chp 30)
- *Coping with NP-Complete Problems* Garey and Johnson (1979, chp 6)

6 M269 Exam 2014J Q Part2

- Answer every question in this part.
- The marks for each question are given at the end of the question.
- Answers to this part should be written in the separate answer book.

Go to Exam 2014J Soln Part2

6.1 M269 2014J Exam Q 16

- The Book Brigade is a start-up online bookseller specialising in electronic books.
- The company asks customers to rate the books they have read on a scale of 1 (dross) to 10 (magnificent), and maintains data in two sequences, B and R.

1. B is an unsorted sequence of ISBNs (International Standard Book Numbers — a unique numerical code for every book published), together with the title of the book the ISBN denotes. Thus each item in B is itself a 2-tuple with items: (1) the ISBN, and (2) the title.
2. R is an unsorted sequence of ISBNs with, for each item, a list of customer ratings for that book. Each item in R is also a 2-tuple with items: (1) the ISBN, and (2) a sequence of ratings.

- Some books in B may not have been rated, and these will not appear in R. Moreover, the order of books in R is not necessarily the same as the order in B.

(a) The company requires a computer system that generates a list of book titles that have been rated, with the average rating of each. Unrated books should not appear in this list. **(5 marks)**

(i) Using the following template, formally state this as a computational problem, in the style adopted by M269.

Name: BookRatings

Inputs:

Outputs:

(ii) Suggest one possible postcondition for this computational problem.

(b) Sketch out an initial insight for a computational solution of the BookRatings problem. **(6 marks)**

(c) Many potential solutions to the BookRatings problem may be inefficient, arising from the fact that neither B nor R are sorted.

- The Book Brigade now require a more efficient solution which will require that both these lists, and the list of results returned by the system, will be sorted.
- Since all these lists are likely to be very long, an efficient sorting algorithm is required, and it has been decided to use *Quicksort*.

(i) Write a short paragraph explaining the process by which Quicksort sorts a list in-place; **(5 marks)**

(c) (ii) The list

44	55	12	42	94	18	6	67

- is being sorted using an in-place Quicksort, with the first item of a partition being chosen as the pivot.
- Draw diagrams illustrating the pivot and the left and right pointers: **(5 marks)**

(1) at the very start (of the first pass),

(2) immediately before the first swap and

(3) immediately before the second swap.

(d) Storing data items (e.g. book title, ratings) that are associated with a key (e.g. ISBN) in a list, as Book Brigade have opted to do, can be problematic, in particular because retrieval may be slow when there are very large numbers of keys. A more suitable means of storage is to associate keys with their data in a *hash table*.

(d) (contd) Write two short paragraphs such that

- the first paragraph explains the basic principles of hash tables and hashing (about 5 sentences), and
- the second paragraph outlines one problem that might arise from hashing, and mentions a strategy for addressing the aforementioned problem (about 3 sentences).

- You do not need to describe the details of specific hashing functions or strategies. **(4 marks)**

Go to Soln 16

6.2 M269 2014J Exam Q 17

- An educational TV show is planning to do a presentation on the Halting Problem. To brief them, you've been asked to write a report for the producers. Assume that the producers do not have a background in computer science.
- Your report **must** have the following structure:

1. A suitable title
2. A paragraph setting the scene and explaining in layperson's terms what is meant by *the Halting Problem* [about two sentences]
3. One paragraph in which you describe the relationship between the Halting Problem and a Turing Machine [about two sentences]
4. One paragraph in which you describe (and give an example in layperson's terms of) proof by contradiction [about three sentences]
5. A conclusion, giving reasons, about the significance of the Halting Problem not being computable [one sentence]

- Note that a significant number of marks will be awarded for coherence and clarity, so avoid abrupt changes of topic and make sure your sentences fit together to tell an overall story.
- Allow up to four additional sentences to ensure this, although note that the numbers of sentences specified, in points 1 to 5 above and in this paragraph, are for guidance only. **(15 marks)**

Go to Soln 17

7 M269 Exam 2014J Soln Part2

- Part 2 solutions

Go to Exam 2014J Q Part2

7.1 M269 2014J Exam Soln 16

(a) (i)

Name: BookRatings

Inputs: An unsorted sequence of tuples $B = (b_1, b_2, \ldots, b_n)$ where $b_n = (i_n, t_n)$ and ISBN, i_n, and title,t_n, are strings.

An unsorted sequence of tuples $R = (r_1, r_2, \ldots, r_k)$ where $r_k = (i_k, l_k)$ and i_k is an ISBN and l_k is a list of ratings (0–10)

Outputs: a list of tuples O = $(o_1, o_2, \ldots, o_p)$ where $o_p = (t_p, a_p)$ and t_p is a title and a_p is a real number between 0 and 10.

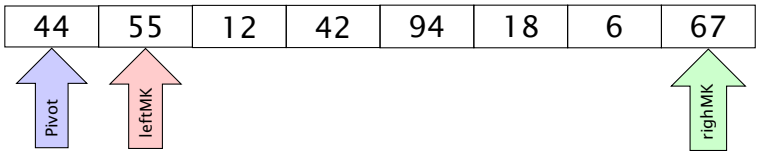
- **(ii)** The length of R must equal the length of O, k = p

(b) Generate the elements of O by iterating over R

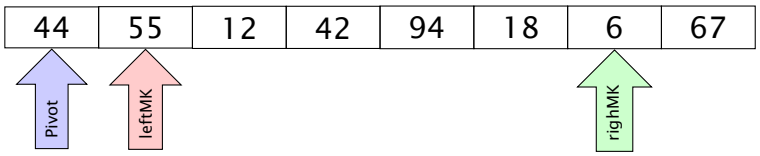
- For each element of R, find the title from the ISBN by iterating over B
- and calculate the average rating from the list of ratings

(c) **(i)** *Quicksort* chooses an item in the list to be the *pivot* item.

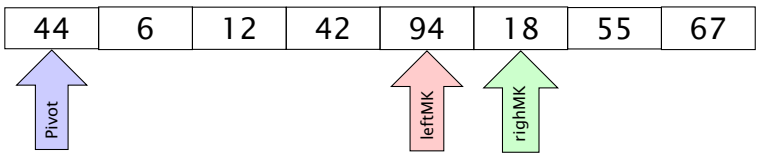
- The algorithm partitions the list into two sublists
- One list comprises items in the list less than the pivot
- The other list comprises the elements in the list greater than or equal to the pivot
- Recursively sort the sub lists (with Quicksort)
- Join the sorted sub lists together with the pivot
- An array based implementation uses two pointers, *leftMK* and *rightMK*, to do the partitioning in place
- At the start of the first pass



- Before the first swap



- Before the second swap



(d) Hash function and hash tables

- Hash function maps each input key to a hash value (or slot)
- Perfect hash function maps each key to a different hash value
- Limited storage leads to hash functions having collisions — a hash function mapping two keys to the same slot
- Hash function collisions result in the need to either store multiple items in a single slot (closed table) or open addressing/open tables that use some mechanism to find a free slot

Go to Q 16

7.2 M269 2014J Exam Soln 17

- **Title** The Significance of the Halting Problem
- **Definition** Can we write an algorithm (a program) that will check whether any other program (algorithm) will terminate (halt) or loop forever
- We need a formal definition of *algorithm* — hence the need for the definition of Turing machine
- Turing machine (and the equivalent Lambda Calculus and others) formalises our idea of functions that are computable — functions that can be calculated.
- Proof by argument similar to Cantor's Diagonal argument and Proof by contradiction
- Direct consequences in mathematics and computer science — the Entscheidungsproblem from David Hilbert 1928 — can we have an algorithm that takes a statement in first order logic and checks if it is valid

Go to Q 17

8 White Slide

9 Web Sites & References

9.1 Web Sites

- **Logic**
  - WFF, WFF'N Proof online http://www.oercommons.org/authoring/1364-basic-wff-n-proof-a-teaching-guide/view
- **Computability**
  - Computability
  - Computable function
  - Decidability (logic)
  - Turing Machines
  - Universal Turing Machine
  - Turing machine simulator
  - Lambda Calculus
  - Von Neumann Architecture
  - Turing Machine XKCD http://www.explainxkcd.com/wiki/index.php/205:_Candy_Button_Paper
  - Turing Machine XKCD http://www.explainxkcd.com/wiki/index.php/505:_A_Bunch_of_Rocks

  - Phil Wadler Bright Club on Computability http://wadler.blogspot.co.uk/2015/05/bri
    club-computability.html

- **Complexity**
  - Complexity class
  - NP complexity
  - NP complete
  - Reduction (complexity)
  - P versus NP problem
  - Graph of NP-Complete Problems

Acknowledgements Toby Thurston for sample answers

Note on References — the list of references is mainly to remind me where I obtained some of the material and is not required reading.

References

Adelson-Velskii, G M and E M Landis (1962). An algorithm for the organization of information. In *Doklady Akademia Nauk SSSR*, volume 146, pages 263–266. Translated from *Soviet Mathematics — Doklady*; 3(5), 1259–1263.

Arora, Sanjeev and Boaz Barak (2009). *Computational Complexity: A Modern Approach*. Cambridge University Press. ISBN 0521424267. URL http://www.cs.princeton.edu/theory/complexity/.

Chiswell, Ian and Wilfrid Hodges (2007). *Mathematical Logic*. Oxford University Press. ISBN 0199215626.

Church, Alonzo et al. (1937). Review: AM Turing, On Computable Numbers, with an Application to the Entscheidungsproblem. *Journal of Symbolic Logic*, 2(1):42–43.

Cook, Stephen A. (1971). The Complexity of Theorem-proving Procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158. ACM, New York, NY, USA. doi:10.1145/800157.805047. URL http://doi.acm.org/10.1145/800157.805047.

Copeland, B. Jack; Carl J. Posy; and Oron Shagrir (2013). *Computability: Turing, Gödel, Church, and Beyond*. The MIT Press. ISBN 0262018993.

Cormen, Thomas H.; Charles E. Leiserson; Ronald L. Rivest; and Clifford Stein (2009). *Introduction to Algorithms*. MIT Press, third edition. ISBN 0262533057. URL http://mitpress.mit.edu/books/introduction-algorithms.

Davis, Martin (1995). Influences of mathematical logic on computer science. In *The Universal Turing Machine A Half-Century Survey*, pages 289–299. Springer.

Davis, Martin (2012). *The Universal Computer: The Road from Leibniz to Turing*. A K Peters/CRC Press. ISBN 1466505192.

Dowsing, R.D.; V.J Rayward-Smith; and C.D Walter (1986). *First Course in Formal Logic and Its Applications in Computer Science*. Blackwells Scientific. ISBN 0632013087.

Franzén, Torkel (2005). *Gödel's Theorem: An Incomplete Guide to Its Use and Abuse*. A K Peters, Ltd. ISBN 1568812388.

Fulop, Sean A. (2006). *On the Logic and Learning of Language*. Trafford Publishing. ISBN 1412023815.

Garey, Michael R. and David S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-completeness*. W.H.Freeman Co Ltd. ISBN 0716710455.

Halbach, Volker (2010). *The Logic Manual*. OUP Oxford. ISBN 0199587841. URL http://logicmanual.philosophy.ox.ac.uk/index.html.

Halpern, Joseph Y; Robert Harper; Neil Immerman; Phokion G Kolaitis; Moshe Y Vardi; and Victor Vianu (2001). On the unusual effectiveness of logic in computer science. *Bulletin of Symbolic Logic*, pages 213–236.

Hindley, J. Roger and Jonathan P. Seldin (1986). *Introduction to Combinators and λ-Calculus*. Cambridge University Press. ISBN 0521318394. URL http://www-maths.swan.ac.uk/staff/jrh/.

Hindley, J. Roger and Jonathan P. Seldin (2008). *Lambda-Calculus and Combinators: An Introduction*. Cambridge University Press. ISBN 0521898854. URL http://www-maths.swan.ac.uk/staff/jrh/.

Hodges, Wilfred (1977). *Logic*. Penguin. ISBN 0140219854.

Hodges, Wilfred (2001). *Logic*. Penguin, second edition. ISBN 0141003146.

Hopcroft, John E.; Rajeev Motwani; and Jeffrey D. Ullman (2001). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, second edition. ISBN 0-201-44124-1.

Hopcroft, John E.; Rajeev Motwani; and Jeffrey D. Ullman (2007). *Introduction to Automata Theory, Languages, and Computation*. Pearson, third edition. ISBN 0321514483. URL http://infolab.stanford.edu/~ullman/ialc.html.

Hopcroft, John E. and Jeffrey D. Ullman (2001). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, first edition. ISBN 020102988X.

Lemmon, Edward John (1965). *Beginning Logic*. Van Nostrand Reinhold. ISBN 0442306768.

Levin, Leonid A (1973). Universal sorting problems. *Problemy Peredachi Informatsii*, 9(3):265–266.

Manna, Zoher (1974). *Mathematical Theory of Computation*. McGraw-Hill. ISBN 0-07-039910-7.

Miller, Bradley W. and David L. Ranum (2011). *Problem Solving with Algorithms and Data Structures Using Python*. Franklin, Beedle Associates Inc, second edition. ISBN 1590282574. URL http://interactivepython.org/courselib/static/pythonds/index.html.

Pelletier, Francis Jeffrey and Allen P Hazen (2012). A history of natural deduction. In Gabbay, Dov M; Francis Jeffrey Pelletier; and John Woods, editors, *Logic: A History of Its Central Concepts*, volume 11 of *Handbook of the History of Logic*, pages 341–414. North Holland. ISBN 0444529373. URL http://www.ualberta.ca/~francisp/papers/PellHazenSubmittedv2.pdf.

Pelletier, Francis Jeffry (2000). A history of natural deduction and elementary logic textbooks. *Logical consequence: Rival approaches*, 1:105–138. URL http://www.sfu.ca/~jeffpell/papers/pelletierNDtexts.pdf.

Rayward-Smith, V J (1983). *A First Course in Formal Language Theory*. Blackwells Scientific. ISBN 0632011769.

Rayward-Smith, V J (1985). *A First Course in Computability*. Blackwells Scientific. ISBN 0632013079.

Rich, Elaine A. (2007). *Automata, Computability and Complexity: Theory and Applications*. Prentice Hall. ISBN 0132288060. URL http://www.cs.utexas.edu/~ear/cs341/automatabook/.

Smith, Peter (2003). *An Introduction to Formal Logic*. Cambridge University Press. ISBN 0521008042. URL http://www.logicmatters.net/ifl/.

Smith, Peter (2007). *An Introduction to Gödel's Theorems*. Cambridge University Press, first edition. ISBN 0521674530.

Smith, Peter (2013). *An Introduction to Gödel's Theorems*. Cambridge University Press, second edition. ISBN 1107606756. URL http://godelbook.net.

Smullyan, Raymond M. (1995). *First-Order Logic*. Dover Publications Inc. ISBN 0486683702.

Soare, Robert Irving (1996). Computability and Recursion. *Bulletin of Symbolic Logic*, 2:284–321. URL http://www.people.cs.uchicago.edu/~soare/History/.

Soare, Robert Irving (2013). Interactive computing and relativized computability. In *Computability: Turing, Gödel, Church, and Beyond*, chapter 9, pages 203–260. The MIT Press. URL http://www.people.cs.uchicago.edu/~soare/Turing/shagrir.pdf.

Teller, Paul (1989a). *A Modern Formal Logic Primer: Predicate and Metatheory: 2*. Prentice-Hall. ISBN 0139031960. URL http://tellerprimer.ucdavis.edu.

Teller, Paul (1989b). *A Modern Formal Logic Primer: Sentence Logic: 1*. Prentice-Hall. ISBN 0139031707. URL http://tellerprimer.ucdavis.edu.

Thompson, Simon (1991). *Type Theory and Functional Programming*. Addison Wesley. ISBN 0201416670. URL http://www.cs.kent.ac.uk/people/staff/sjt/TTFP/.

Tomassi, Paul (1999). *Logic*. Routledge. ISBN 0415166969. URL http://emilkirkegaard.dk/en/wp-content/uploads/Paul-Tomassi-Logic.pdf.

Turing, Alan Mathison (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265.

Turing, Alan Mathison (1937). On computable numbers, with an application to the Entscheidungsproblem. A Correction. *Proceedings of the London Methematical Society*, 43:544–546.

van Dalen, Dirk (1994). *Logic and Structure*. Springer-Verlag, third edition. ISBN 0387578390.

van Dalen, Dirk (2012). *Logic and Structure*. Springer-Verlag, fifth edition. ISBN 1447145577.

Author Donna & Phil Written 20 May 2017 (2 sessions) *Printed 19th May 2017*
Subject dir: ⟨*baseURL*⟩/OU/M269/M269Exams/M269SpecimenExam/M269ExamRevision
Topic path: /M269ExamRevision2016J/M269ExamRevision2016JA/M269ExamRevision2016JA.pdf