The University of Nottingham

SCHOOL OF COMPUTER SCIENCE

A LEVEL 4 MODULE, AUTUMN SEMESTER 2017-2018

ADVANCED ALGORITHMS AND DATA STRUCTURES

Time allowed: TWO HOURS

Candidates may complete the front cover of their answer book and sign their desk card but must NOT write anything else until the start of the examination period is announced

Answer ALL SIX questions

Marks available for sections of questions are shown in brackets in the right-hand margin.

No calculators are permitted in this examination.

Dictionaries are not allowed with one exception. Those whose first language is not English may use a standard translation dictionary to translate between that language and English provided that neither language is the subject of this examination. Subject specific translation dictionaries are not permitted.

No electronic devices capable of storing and retrieving text, including electronic dictionaries, may be used.

DO NOT turn examination paper over until instructed to do so

INFORMATION FOR INVIGILATORS: The exam paper should be collected and placed inside the answer book.

Question 1. This question is about the big-Oh family, recurrence relations, and randomised algorithms (20 marks total)

(a) Consider the following functions

$$\begin{aligned} f_1 &= n \\ f_2 &= n * \log(n) \\ f_3 &= \log(n)^2 \\ f_4 &= n^2 \end{aligned}$$

For each of the following complexity classes, state which of the above functions are a member of the class:

i. $o(n)$ (little-oh)
ii. $O(n)$ (big-Oh)
iii. $\Theta(n)$
iv. $\Omega(n)$

(4 marks)

(b) Consider the following recurrence relation:

$$\begin{aligned} T(n) &= 2\,T(n/3) \\ T(1) &= 1 \end{aligned}$$

Give the exact solution, and prove that your answer is correct using induction.

(6 marks)

(c) Consider the following recurrence relation:

$$\begin{aligned} T(n) &= 2\,T(n/2) + n^2 \\ T(1) &= 1 \end{aligned}$$

Use the Master theorem to give the asymptotic complexity of $T(n)$ (using an appropriate member of the big-Oh family.)

(5 marks)

(d) Define the "Coupon Collector Problem", and give a brief summary of the key results of its properties. (There is no need to give full derivations.)

Also, briefly discuss the implications of the Coupon Collector Problem for the efficiency of randomised search algorithms.

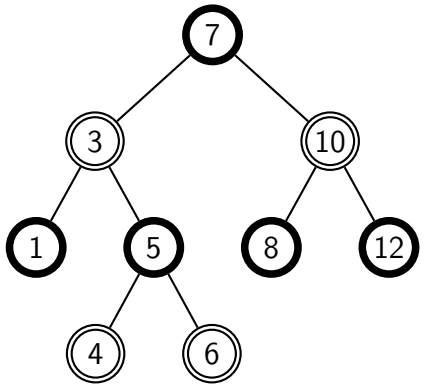
(5 marks)

Question 2: This question is about red-black trees (15 marks total, 5 marks for each part)

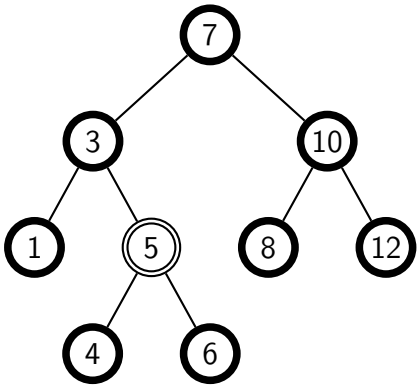
(a) Give the definition of *Red-Black Trees*. Clearly explain what is the data structure and which properties it must satisfy.

(b) Which ones of the following trees are correct Red-Black Trees? (There may be more than one correct tree.)
For those that are not, expain which property they violate.
(thick circle = black node, double circle = red node, leaves not shown)

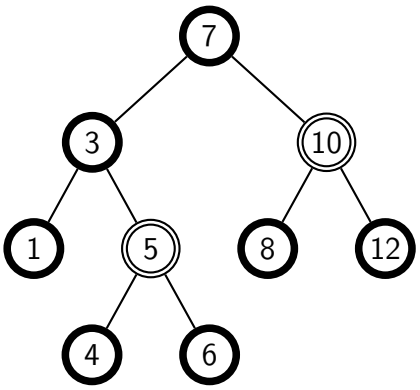
(i)



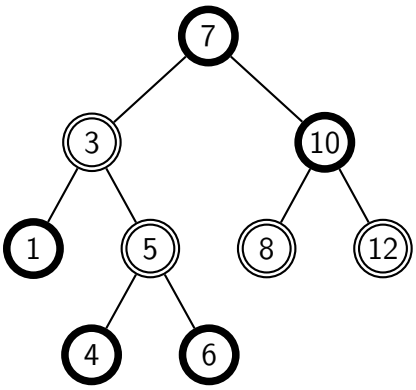
(ii)



(iii)



(iv)



(c) What is the largest possible number of nodes in a red-black tree with black-height k? What is the smallest possible number?

Give a proof of your answers.

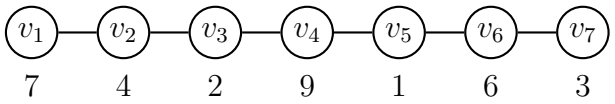
Question 3. This question is about Dynamic Programming (DP) (15 marks total)

(a) What do *optimal substructure* and *overlapping substructure* mean in the context of DP? How do they make a problem is suitable for DP? Illustrate your answer with a simple standard problem of your choice (e.g. "shortest path", or "change-giving") (5 marks)

(b) Consider the **maximum weight independent set** problem for graphs:
Input: An undirected graph G = (V,E), with a positive integer weight $w(v)$ for each vertex $v \in V$
Output: A subset S of vertices, such that no two nodes in S have an edge between them, with a maximal total sum of weights:

$$\text{maximize } \sum_{v \in S} w(v), \quad \text{subject to: } (u,v) \in E \Rightarrow \neg(u \in S) \vee \neg(v \in S).$$

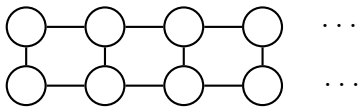
Show how to solve this problem using DP, **for the restricted case of graphs that are simple "*chains*":** A "chain" is a connected graph in which each node has at most two neigbours. For example:



In this example, the optimal solution for S uses the nodes $\{v_1, v_4, v_6\}$, and has total weight $7 + 9 + 6 = 22$.

Hint: Work down the chain from left to right and keep track of the "best value(s) so far". (6 marks)

(c) Consider the same problem as (b) but in the restricted case of the graph looking like a "ladder", that is, with the structure of two chains connected together, as illustrated below:



Do you expect the be able to efficiently solve this special 'ladder' case using dynamic programming? Briefly justify your answer. (4 marks)

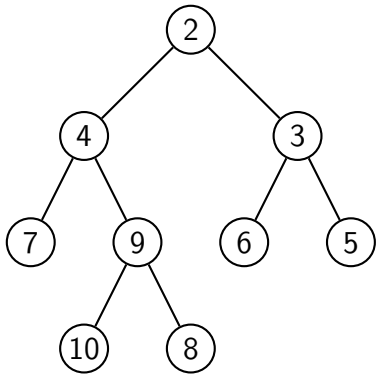
Question 4: This question is about heaps (15 marks total, 5 marks for each part)

(a) Give an informal definition of *Priority Queues* (also called *Heaps*) as an abstract data type: What methods must an implementation of heaps support?

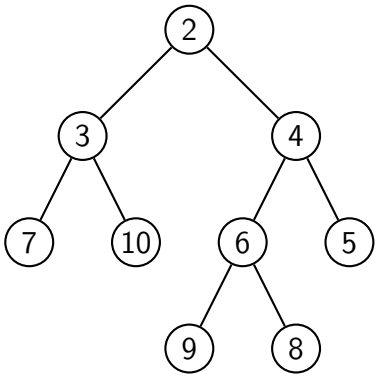
(b) Define the implementation of priority queues as *Leftist Heaps*: What property must a heap satisfy? How are the methods implemented?

(c) Which of the following trees are correct Leftist Heaps? Give the rank of each. For those that are not Leftist Heaps, explain which property they violate.

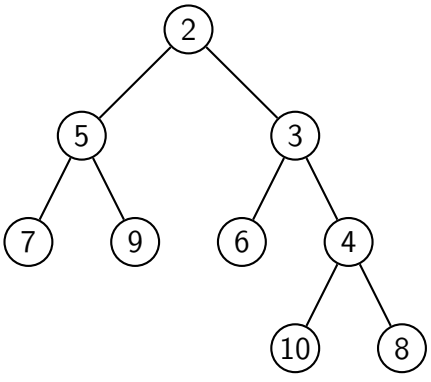
(i)



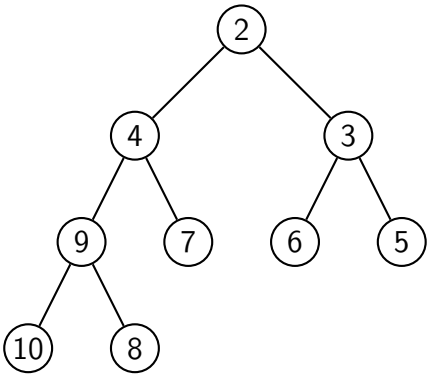
(ii)



(iii)



(iv)



Question 5. This question is about network flow (15 marks total)

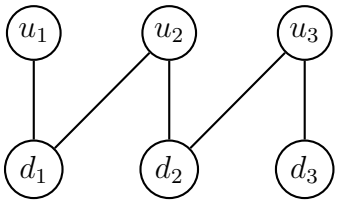
(a) What is the definition of a flow function in a flow network? What is the definition of the value of a flow?

Also, briefly describe the algorithm used to solve the maximum flow problem. In particular, define "residual network" and "augmenting path".

(7 marks)

(b) Define the maximum bipartite matching problem. Explain how network flow methods can be used to solve it.

In particular, consider the maximum bipartite matching problem on the following graph:



Describe how to convert this to a problem in network flow.

Then, suppose that the initial sub-optimal solution uses the edges (u_2, d_1) and (u_3, d_2).

Show the corresponding network flow diagram.

Then show the resulting augmenting path in the residual network; and how it increases the flow and hence optimally solves the matching problem.

(8 marks)

Question 6: This question concerns amortised analysis and queues (20 marks total)

(a) What do we mean by the *amortize complexity* of an algorithm? Describe the *potential* method of amortized analysis of time complexity works.

(5 marks)

(b) *Double-ended queues* are data structures containing a sequence of elements with operations to add and delete elements at both ends. Suppose that the double-ended queues are defined as pairs of lists, both lists should be non-empty if the queue contains at least two elements.

```
type Deque a = ([a],[a])
```

The actual queue corresponding to a pair `(f,r)` is the concatenation of `f` with the reverse of `r`. For example the double-ended queue $\langle 7, 2, 5, 8, 0, 4, 3\rangle$ can be represented by the pair of lists $([7, 2, 5], [8, 0, 4, 3])$, or, equivalently, by $([7, 2, 5, 8, 0], [4, 3])$.

Given this data structure, define (in pseudo-code) efficient operations inserting and deleting elements at both ends of the queue:

```
cons :: a -> Deque a -> Deque a
tail :: Deque a -> Deque a
snoc :: a -> Deque a -> Deque a
init :: Deque a -> Deque a
```

You should give code or pseudo-code.

Be careful to maintain the invariant that if the queue contains at least two elements, then both lists must be non-empty. For example:

```
tail ([7],[2,5,8,0,4,3]) = ([2,5,8],[0,4,3])
```

(8 marks)

(c) Define a potential function on queues. (Hint: use the difference in length of the two list components).

Use it to prove that all operations have amortized time cost of $O(1)$.

(7 marks)