# The University of Nottingham

SCHOOL OF COMPUTER SCIENCE

A LEVEL 2 MODULE, AUTUMN SEMESTER 2013-2014

**ALGORITHMS AND DATA STRUCTURES**

Time allowed NINETY Minutes

---

# *PARTIAL AN*SWERS AND HINTS

Note that these answers are just a sketch of the key essentials of what you should write.
As written they would get an average to good mark, but not a perfect one!.

1. This question is compulsory.

(a) Give appropriate descriptions in terms of each of \`O' (big-Oh), \`Θ' (big-Theta), and \`Ω' (big-Omega) of the function

$f(n) = 2n^3 + 7n^2 \log(n^4)$

You should justify your answers, but do not need to give complete proofs.

(5 marks)

**$\log(n^4) = 4 \log(n)$ and so at large n the 2nd term is dominated by the first term.**
**The constant 2 does not matter, and so, for the big-Oh family, f(n) is essentially $n^3$.**
**Hence, it is $O(n^3)$, $\Omega(n^3)$ and hence also $\Theta(n^3)$.**

(b) Consider the following code for insertion sort to sort an array of integers into non-decreasing order (note that the array indices start at 0 as usual)

```
1.  void sort( int[] A ) {
2.          for ( int k=1 ;  k < A.length ; k++ )
3.          {
4.                  int temp = A[ k ];
5.                  int i = k;
6.                  while( i > 0 && A[i-1] >= temp ) {
7.                          A[i] = A[i-1];
8.                          i--;
9.          }
10.                 A[i] = temp;
11.                 assertTrue( S );
12.         }
13.     }
```

Give an appropriate and useful choice for assertion S on line 11, and justify your answer. You may write your answer using either a logical expression or as pseudo-code.

(5 marks)

**Insertion sort works by keeping the front part of the array in sorted order.**

**So an appropriate assertion would say, for example**

**for all i < j in [0, k ]    arr[i] <= arr[j]**

(c) Consider the array

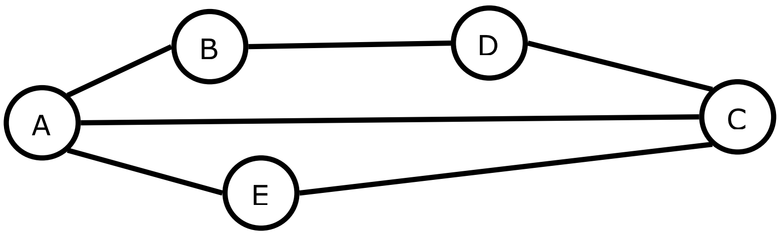
[ 2 5 3 1 4 ]

Show how `quicksort' can sort it into (increasing) order.
Use an initial pivot value of 3 during the first partition step.

(5 marks)

**During partition using 3 have to swap 5 and 1 to give [ 2 1 | 3 | 5 4 ]. The quicksort of the subarrays [ 2 1] and [5 4] can then be done using any pivots.**

(d) Consider the following graph



List all possible orders in which nodes can be visited during a depth-first traversal starting from node A.

For each possible traversal you only need to give the sequence of nodes, writing it in the format [ A _ _ _ _ ]. You do not to show the details of how each sequence is produced.

(4 marks)

**[ A B D C E ]**
**[ A E C D B ]**
**[ A C E D B ]**
**[ A C D B E ]**

**Some missed the traversals starting with the edge A C. It is important not to get fooled by the way that the graph is layed out on the paper.**

(e) Define and explain the concept of a “Minimum Spanning Tree” (MST) in an undirected weighted graph. Carefully, distinguish the MST problem from the shortest path problem by giving a graph in which the MST is not a path. Also, briefly describe an algorithm to find a MST.

(6 marks)

**[This is almost entirely in the lectures[**
**Spanning tree is a tree that includes all the modes.**
**A minimum is one that has a minimum sum of the edge weights.**
**Any example such as a star graph will suffice.**
**For example a graph on nodes ABCD with edges AB AC AD. The graph is a tree and so the spanning tree must be the original graph, and it is not a path.**

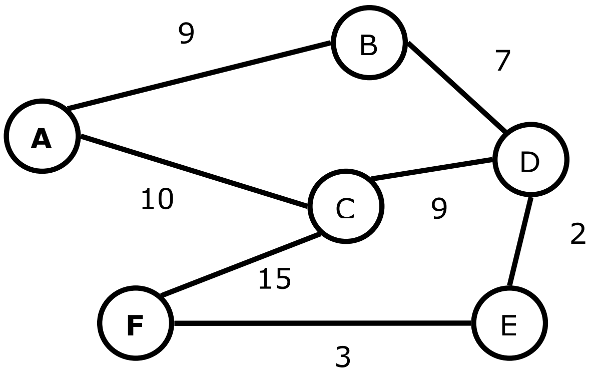
**The standard algorithm is to greedily construct a tree.**
**Start with any node. At each stage add the lowest cost edge that goes from the tree to a a node that is not yet included.**

2. This question concerns paths and trees in graphs.

   (a) Explain, and give pseudo-code for, Dijkstra's algorithm to find the shortest path in an undirected graph when the distances for edges are given and all are strictly positive (greater than zero).

   Include at least one appropriate assertion or loop invariant within your code. In particular, carefully explain why the algorithm is guaranteed to give a shortest path.

   Use Dijkstra's algorithm to find the shortest path from node A to node F in the graph below. Show your working.



(10 marks)

**Standard pseudo-code from lectures can be used. Essentially, it maintains a Priority Queue of 'open nodes' that are sorted according to the best-known shortest path to them. The top of the PQ is removed. If it is the goal then the algorithm is finished. Otherwise the node is expanded by visiting the neighbours (excluding those on a closed list) and inserting the new path into the PQ. The node is then closed.**

**It works because it never closes a node until the shortest path to that node has been discovered, and this can be used at the invariant.**

**In this case we should see PQ and closed being**
**[A(0)] []**
**[B(9), C(10)] [A{0)]**
**[C(10),D(16)] [A(0),B(9)]**
**[D(16), F(25)] [A(0),B(9), C(10)]**
**[E(18),F(25)] [A(0),B(9), C(10), D(16)]**
**[F(21)] [A(0),B(9), C(10), D(16), E(18)]**
**goal**

final path length 21 via A B D E F

(b) Discuss whether or not the standard Dijkstra algorithm can be improved or made more efficient when it is known in advance that the graph is a tree.

(4 marks)

**Yes, we no longer need to account for the possibility of there being multiple paths to a node. Hence we can terminate as soon as we discover any path to the goal.**

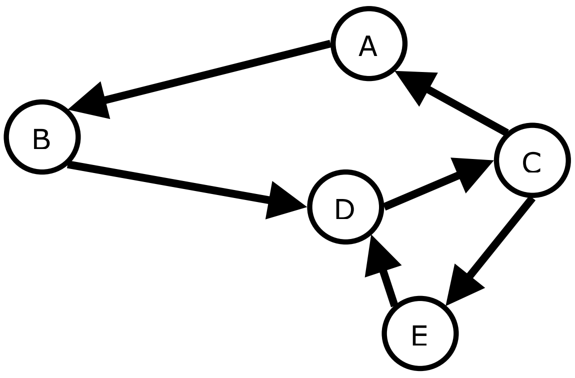
(c) Show how depth first search (DFS) can be used to detect cycles in a directed (unweighted) graph and illustrate your answer using a small example graph.

(7 marks)

**Can be the same as from the 2013-14 lecture notes.**
**Perform a standard DFS using a Stack and backtracking. However, also mark the nodes with colours: and then use these to detect when the expansion of a node has a neighbour that is already in the stack. Any simple graph with a cycle will suffice as an example.**

(d) As in part (c) it is desired to detect cycles in (unweighted) directed graphs; however, with the additional requirement to find those cycles with the smallest number of edges. For example, in the graph below it is required to find the `3-cycle' D-C-E and not just the `4-cycle' A-B-D-C.

Discuss why DFS does not automatically do this, and then make some suggestions for a better algorithm.



(4 marks)

**DFS has no pressure or mechanism to find small loops and so starting from D might well discover the 4-cycle first.**

**Definitely not expected to be able to give a full answer, but would hope to be able to hint that would need to do something that is much more breadth first in nature.**
**For example, to do BFS from every point, and use the same colouring trick.**
**Or (equivalently) to modify Dijkstra so that it would detect when it has discovered a cycle.**

**This is challenging and was not directly taught so was marked generously!**

3. This question concerns Heaps and Binary Search Trees.

(a)

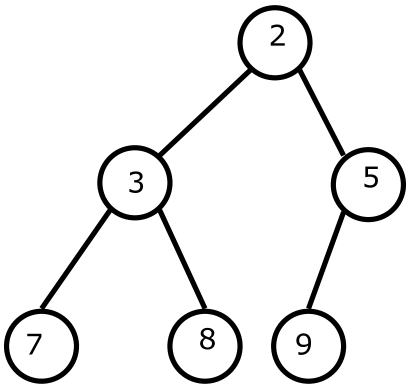
i) State and briefly explain the standard methods of the interface of Map Abstract Data Type (ADT).
ii) State what it means for a binary tree to have the binary search tree property and why this makes it suitable to implement a Map.
iii) State and briefly explain the standard methods of the interface of Priority Queue Abstract Data Type (ADT).
iv) State what it means for a binary tree to have the heap property and why this makes it suitable to implement a Priority Queue.
v) Carefully explain and justify the ways in which the heap property differs from the binary search tree property.

(8 marks)

**Included as, in the past, many have confused heaps and BSTs.**
**Standard answer, from the lectures.**
**BST: all entries of left sub tree < node < all entries of right subtree**
**HEAP: all children > their parent, and the tree is complete up till the last level, which is shifted left as much as possible.**

**The BST is to allow one to find any entry, but for the Heap we only need to access the root and the last entry.**

(b) The following binary tree is the initial state of a heap



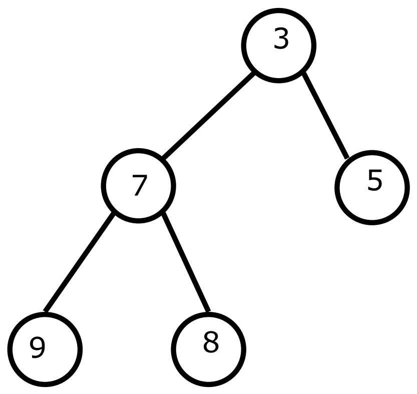
Do a removeMin() operation to remove the minimum key, 2, then insert an entry with key 4.

What is the final state of the tree? Show your working.

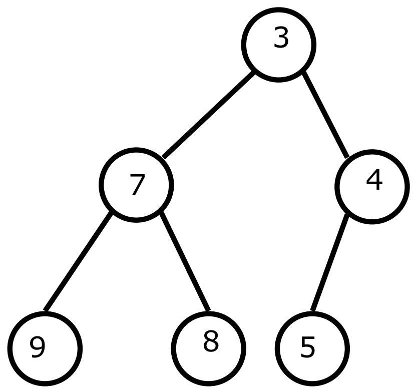
Also convert to the final tree to an array based representation in the standard fashion used for heaps, that is, with the root of the heap placed at index 1, etc. Write your final result in the form of an array with a '#' denoting an unused or blank entry.

(7 marks)

**ANSWER: remove 2: swap 9 to the root, and then down-heap by swapping with the <u>smallest</u> child.**

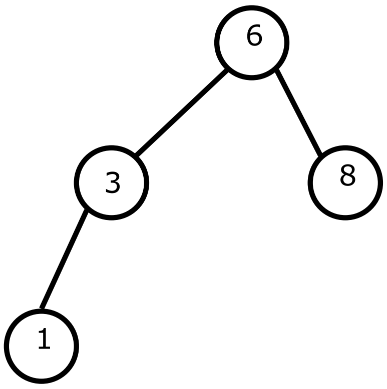


**then for insert, add to the end and up-heap swaps it with 5**



**Finally as an array this is [# 3 7 4 9 8 5 # ]**

(c) The following diagram shows the initial state of a binary search tree.



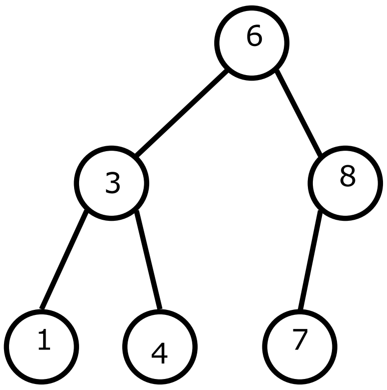
Show and explain your working for the resulting state of the binary search tree after performing each step of the following sequence of operations:

i) insert( 4 ), and then

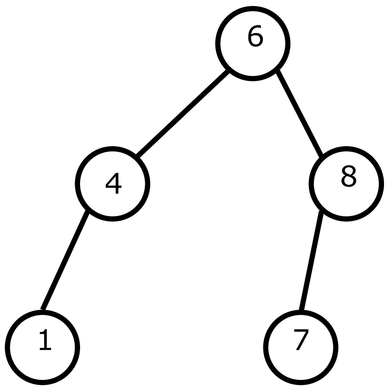
ii) insert( 7 ), and then

iii) delete( 3 ) (6 marks)

**ANSWER: Insert 4 and then 7 gives**



**then deleting 3, looks at the 'first' element of the right subtree and moves it into the place of 3, hence giving**


6
4
8
1
7

(d) Suppose that someone suggests defining a ‘ternary heap’ in which each node can have up to 3 children, and argues that it would be faster because it will have a reduced height and so would be faster than a binary heap.

Discuss whether or not such a heap is possible to define and implement.
Also, discuss reasons for whether or not it might it would be faster in practice.

(6 marks)

**Yes it is possible to define such a heap. The upheap will stay the same, and the downheap needs to still swap with the smallest of the children.**
**However, the reduced height is just a relatively small factor log(3)/log(2) and the gain is likely to be more than offset by the higher cost of finding the smallest child on the downheap.**
**Note this is quite a hard question and will be marked generously.**

**NOTE: Of course it was also C/W 3 of 2014-15** ☺

4. This question is concerned with the 'big-Oh' family and amortised complexity

   (a) Give, and also explain and justify, the definitions of big-Oh, big-Omega, big-Theta and little-oh by completing each of the following:

      i) 'big-Oh': f(n) is O( g(n) ) ….

      ii) 'big-Omega': f(n) is Ω( g(n) ) ….

      iii) 'big-Theta': f(n) is Θ( g(n) ) ….

      iv) 'little-oh': f(n) is o( g(n) ) ….

   Your justification and explanation should address the most important parts of each definition showing why they are defined in the way that they are, and in particular, the choices of the quantifiers "there exist" and "for all" and their relative ordering.

   (10 marks)

**All the definitions are standard from the lecture notes. They should observe the little oh has a "for all c" instead of "exist c".**

**Justification should include some explanation of the portions, e.g.**

**for all n >= n0**

**is because we want to characterise the behaviour of the function at large n, and want to allow the behaviour at small values of n to be messier and irregular.**

   (b) From the definitions prove or disprove that $f(n) = 2\,n^3 + 7\,n$ is

      i) $O(n^3)$ (big-Oh)

      ii) $o(n^3)$ (little-oh)

   (7 marks)

**i) true. just need any value of c > 2. E.g. c=3, and correctly compute the minimum $n_0$ that is needed.**
**ii) false. Observe that a value of c <= 2 will fail. E.g. c = 1 will give a contradiction**

   (c) Briefly, explain the notion and usage of amortised complexity when considering a sequence of operations on a data structure. In particular, explain the difference from the worst case complexity.

   (3 marks)

**Worst case refers to any single operation. But amortised refers to the average over an achievable sequence of operations. The point of a sequence is that they are linked,**

**and so the effects (costs) of an operation will be correlated with previous ones, and not totally independent. For example if we just had to double the size the array in a Vector, then we will not have to double it again on the next operation. However with an "average" we would average over a set of operations that might well be independent.**

**Note this difference is important, but is relatively subtle and difficult, and so would be marked relatively generously.**

(d) Consider a Vector implemented using an underlying array. When the array is full then it needs to be resized.

i) Consider the scheme in which the array size is doubled when needed. Give the worst case complexity and compute the amortised complexity.

ii) Briefly discuss what would change to the worst case and amortised complexities if the array size were tripled as opposed to doubled.

(5 mark

**The case of doubling is from the lecture notes, and answer should consider the total cost of n additions starting from an empty vector. Sum the geometric series and so get a total cost that is still O(n) giving an amortised cost of O(1).**

**Slightly harder is that in the case of tripling, the main term of the geometric series is still the same up to constants, and so the amortised cost will still be O(1).**