LC Data Structures and Algorithms

Solutions

Main January Examination 2023

Question 1

(a) If a Binary Tree is either empty or is a node with a left child and a right child that are both Binary Trees, what additional conditions must a Binary Tree satisfy for it to be

(i) *Complete*, and

(ii) a *Binary Heap Tree*?

[4 marks]

(b) The items [6, 9, 4, 8, 12, 5] need to be inserted one at a time into a *Binary* **Heap** *Tree*, starting from an empty tree. Show the state of the tree after each item has been inserted. **[6 marks]**

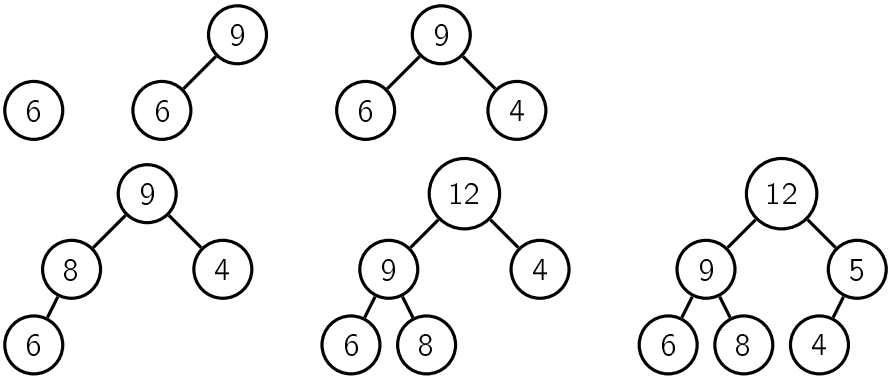
(c) One approach for sorting an array of items, a, is to first insert them into a *Binary Search Tree*, t, and then output them back into the array in order. Write, in pseudocode, a recursive procedure for filling the array from the *Binary Search Tree*, and specify what the initial call of the procedure is. You may call any of the standard primitive operators for binary trees **[10 marks]**

Model answer:

(a) A Binary Tree is Complete if it has each layer filled from left to right before moving on to the next layer.

A Binary Tree is a Binary Heap Tree if it is Complete and each non-leaf node value is greater than or equal to all the node values in its two sub-trees.

(b)



(c)

```
1  fillArray(tree t, array a, int j)
2  {
3      if ( not isEmpty(t) )
4      {
5          j = fillArray(left(t),a,j)
6          a[j++] = root(t)
7          j = fillArray(right(t),a,j)
8      }
9      return j
10 }
```

The initial call is fillArray(t,a,0).

Question 2

(a) Describe how one can establish the best possible average-case time complexity of any comparison-based sorting algorithm. Does *Binary Search Tree*-based sorting achieve that? **[6 marks]**

(b) Show how *Two-Phase Radix Sort* can be used to sort the following set of dates given in day/month format: [17/7, 12/7, 8/4, 8/7, 9/3, 12/4, 17/3, 12/6]. **[6 marks]**

(c) Explain how *Radix Sort* could be used to sort a set of integers without duplicates. Describe the time complexity of your approach and determine if it is more efficient than the best possible comparison-based sorting algorithms. **[8 marks]**

Model answer:

(a) Given n items to sort, one can represent the comparisons as a binary decision tree with leaves representing the $n!$ potential orderings of the n items. The average case time complexity of the most efficient sorting algorithm will then correspond to height h of the smallest binary tree that can accommodate the $n!$ leaves. That means $2^h \approx n!$ or $h \approx \log n! \approx O(n \log n)$, so the best average case time complexity will be $O(n \log n)$, which is achieved by *Binary Search Tree*-based sorting.

(b) The first phase produces a queue of items for each value of the least significant key (the day), and those queues are concatenated in day order.

```
8:  8/4, 8/7
9:  9/3
12:  12/7, 12/4, 12/6
17;  17/7, 17/3
=>  8/4, 8/7,  9/3, 12/7, 12/4, 12/6, 17/7, 17/3
```

Then those items are put into queues for each value of the most significant key (the month), preserving their existing order.

```
3:  9/3, 17/3
4:  8/4, 12/4
6:  12/6
7:  8/7, 12/7, 17/7
```

When those queues are concatenated in month order, the array is sorted as required:

```
9/3, 17/3,  8/4, 12/4, 12/6,  8/7, 12/7, 17/7
```

(c) A set of integers can be sorted by forming queues and concatenating for each digit in order of increasing significance. If there are n integers to sort, each phase has $O(n)$ complexity, since all n items have to be processed, and there will be as many phases as there are digits in the integers, which will be $O(\log n)$ if there are no/few duplicates. Thus the overall complexity is $O(n \log n)$ which is no better than the best comparison-based sorts.

Question 3

(a) Consider a graph represented by a symmetric $N \times N$ weight matrix. What does the **size** and **symmetry** of that matrix indicate?

In the weight matrix, the symbol ∞ is used to represent the lack of an edge connecting the vertices indicated by the row and column where the symbol appears, and the **connectivity level** of a graph is defined as the actual number of edges divided by the maximum possible number of edges.

Given a symmetric $N \times N$ weight matrix representing a graph, provide a formula for the connectivity level of the graph as a function of N and the number M of ∞ symbols it contains? **[6 marks]**

(b) Describe an efficient greedy edge-based algorithm for determining a minimal spanning tree of a weighted graph. In what sense is your algorithm greedy? **[6 marks]**

(c) What aspect of the algorithm you describe in part (b) above contributes most to its time complexity, and what is this algorithm's overall time complexity? Comment on the speed of this algorithm on highly connected graphs compared to Jarník-Prim's vertex-based algorithm for the same problem. **[8 marks]**

Model answer:

(a) The size indicates the number of nodes/vertices in the graph and the symmetry indicates the graph is undirected. The maximum possible number of edges is $E = \frac{N(N-1)}{2}$. The number M of ∞ symbols is twice the number of missing edges. Thus the connectivity level is:

$$\frac{E - M/2}{E} = \frac{N(N-1) - M}{N(N-1)} = 1 - \frac{M}{N(N-1)}$$

(b) Kruskal's algorithm: At each stage, for the current set of edges T, consider all edges not yet in T and add the one with minimal weight that does not produce a cycle. Starting with an empty set of edges, and repeating till all vertices are included, leads to a minimal spanning tree. It is greedy in the sense that it makes decisions based on what is best at each stage, rather than what might be best overall.

(c) The time complexity of the edge based algorithm is dominated by that of sorting the edges, so it is $O(e \log e)$ overall, where e is the number of edges in the graph. Highly connected graphs have $e \approx v^2$, where v is the number of vertices in the graph, and hence the time complexity is then $O(v^2 \log v)$. The time complexity of Jarník-Prim's algorithm for highly connected graphs is only $O(v^2)$, so that will be faster for highly connected graphs.