Data Structures and Algorithms – 2018

Assignment 3 – 25% of Continuous Assessment Mark

Deadline : 5pm Monday 26th February, via Canvas

Question 1 (12 marks)

The numbers [25, 12, 6, 17, 29, 18] need to be inserted one at a time, in that order, into an initially empty Binary Heap Tree. Draw the state of the Heap Tree after each number has been inserted.

Now draw the Heap Tree after each of the first two highest priority items have been removed from the resulting Heap Tree.

Finally, draw the Heap Tree after each of the two removed items have been added back to the Heap Tree in the order they were removed.

To what extent does the order of the initial list of items affect the heap tree that is produced and the order in which the highest priority items are removed?

Question 2 (10 marks)

Write an efficient recursion-based procedure `isHeap(t)` that returns `true` if the binary tree `t` is a heap tree, and `false` if it is not. You can call any of the standard primitive binary tree procedures `isEmpty(t)`, `root(t)`, `left(t)` and `right(t)`, and also a procedure `complete(t)` that returns `true` if `t` is complete, and `false` if it is not.

What can be said about the overall complexity of your algorithm?

Question 3 (14 marks)

Insert the integers [5, 4, 9, 1, 3, 8] in that order into an initially empty binomial heap. Show the heap after each item is inserted.

Now delete, one at a time, the two highest priority items. Show the binomial heap that is left after each item has been deleted.

Question 4 (12 marks)

The following segment of C/Java code will sort an array `a` of finite size `n`:

```
for( i = 1 ; i != n ; i++ ) {
    j = i;
    t = a[j];
    while( j > 0 && t < a[j-1] ) a[j] = a[--j];
    a[j] = t;
}
```

Which sorting algorithm discussed in the lectures is this an implementation of?

Work through the sorting of the array [6, 4, 8, 5, 2, 7] using this algorithm, writing down the values of `i, j, t` and the array `a` at the end of each iteration of the `for` loop.

State an appropriate *loop invariant* for this algorithm, and use that to argue why the algorithm is guaranteed to terminate with the array `a` sorted.

Question 5 (22 marks)

One often needs to determine whether a given collection of items contains any duplicates. Write two simple procedures, `duplicates1(a,n)` and `duplicates2(a,n)`, not involving any form of hash map, that efficiently return `true` if the array `a` of `n` integers contains duplicates, and `false` if it does not. The first procedure cannot use any form of sorting, while the second must start with a procedure `Xsort(a,n)` which returns a sorted version of array `a` with average and worst case time complexity $O(n \log n)$. You should assume that all array `a` indices run from `0` to `n-1`.

Give informal arguments why your algorithms are correct.

Explain the worst-case and average-case time complexities of each of your two algorithms, and hence comment on which approach is best for each case. [Hint: The average-case time complexities will depend on what you are averaging over. Consider the extreme cases of duplicates being very common and duplicates being very rare.]

Question 6 (14 marks)

When Binary Search Trees are used for storing items, it makes sense to require that all the search keys inserted into them are unique. In the Lectures Notes (section 7.4), the standard insertion procedure `insert(v,bst)` is presented that inserts a value, or search key, `v` into such a binary search tree `bst`. Summarize, in the form of a list of possible cases that need to be dealt with, how that procedure works.

If the binary search trees are to be used as the basis of a sorting algorithm (Treesort), it will generally be necessary to allow duplicate items. Describe the simplest possible ways the standard `insert(v,bst)` procedure could be modified to accommodate duplicate entries.

Comment on how the possible modifications would affect the stability of a Treesort algorithm based on it.

One often needs to sort a set of items with any duplicate items removed at the same time. State in words how the standard `insert(v,bst)` procedure for Binary Search Trees could easily be modified to create a Treesort algorithm that returns the items sorted in that way.

Question 7 (16 marks)

Suppose, the standard `insert(v,bst)` procedure for Binary Search Trees was modified to return an `EmptyTree` if `v` already exists in `bst`, rather than terminating with an error message. Write an efficient pseudocode procedure `duplicates3(a,n)` using it that returns `true` if integer array `a` of size `n` contains duplicates, and `false` if it does not.

Explain the worst-case and average-case time complexities of your algorithm, and compare them with those you found for your algorithms in Question 5. State which of the three algorithms is best in which circumstances.