Foundations of Computer Science (Semester 2) – 2015

Assessed Exercise Sheet 5 – 10% of Continuous Assessment Mark

Deadline : 11pm Sunday 22nd February, via Canvas

Question 1 (20 marks)

The numbers [25, 12, 8, 17, 26, 18] need to be inserted in that order into an initially empty 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.

Question 2 (20 marks)

There are three obvious ways of merging two similarly sized heap trees `s` and `t` into a single heap tree:

(i) Move the items one at a time from the smaller heap tree into the larger heap tree using the `insert` algorithm studied in the lecture notes (section 7.4).

(ii) Repeatedly move the last items from one heap tree to the other, with bubbling up, until the new binary tree `makeTree(0,t,s)` is complete. Then move the last item of the new tree to replace the dummy root "0", and bubble down that new root.

(iii) Concatenate the array forms of `s` and `t` and use the `heapify` algorithm from the lecture notes (section 7.6) to convert it into a new heap tree.

Comment on the average-case time complexities of each approach, and thus suggest which approach would be the best in practice to use in general. Are there any special cases for which a different choice might be appropriate?

Question 3 (18 marks)

It is stated in the lecture notes (section 7.7) that the structure of a Binomial Heap with n nodes "is unique, and will consist of at most $\log_2 n+1$ trees". Give an intuitive explanation of why that must be true.

Insert the integers [5, 3, 7, 2, 4, 8] in that order into a Binomial Heap. Show the heap after each item is inserted.

Question 4 (20 marks)

Often one needs to determine whether an array contains any duplicate items. Write a procedure `duplicates(a)`, which doesn't involve sorting, that returns `true` if integer

array `a` contains duplicates, and `false` otherwise. You may assume that you have access to a procedure `size(a)` that returns the size of an array `a`.

What is the worst case time complexity of your algorithm?

What about the average case time complexity of your algorithm?

Question 5 (22 marks)

Suppose you have access to a procedure `Xsort(a)` that returns a sorted version of array `a`, that you know has average and worst case time complexity of at least $O(n \log n)$. Write an efficient new procedure `duplicates2(a)` to perform the test for duplicates, that begins by calling `Xsort(a)` to sort the given array. You may again assume that you have access to a procedure `size(a)` that returns the size of an array `a`.

What is the overall worst case time complexity of your algorithm?

What about the overall average case time complexity of your algorithm?

Thus, comment on when it is worth using `Xsort` in a duplicates testing algorithm.