ELL781: Minor Test II

October 4, 2018

Maximum Marks: 20

1. Consider the below array.

   `[5, 4, 7, 3, 10, 2, 6, 8, 1]`

   (a) Suppose the quicksort partition function is called on this entire array, with pivot value `5`. Show the state of the array through the sequence of iterations in the partition function. For each iteration, the final locations of the two cursors (`i` and `j`) should be clearly indicated, and then the state of the array after the swap should be shown for the next iteration. **[2]**

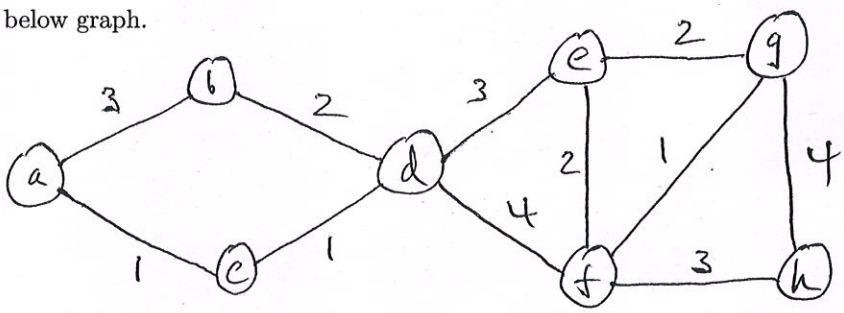
   (b) What is the total number of comparisons that happen during the sequence of iterations of the partition function? And the total number of swaps? **[1.5]**

   (c) What is the final value (index) returned by the partition function? Assume the above array is indexed starting from `0`. **[0.5]**

   (d) Suppose we use mergesort instead of quicksort on the above array, making the first two recursive calls on the subarrays from index `0` to index `4`, and from index `5` to index `8`. Obtain the total number of comparisons that happen *only in the final merge* of these two subarrays, after each has been recursively sorted. **[1.5]**

   (e) Compare your answers to parts (b) and (d). Based on these, which algorithm appears to be more efficient in terms of the number of comparisons/swaps involved? Is this consistent with what was mentioned in class, in terms of which of the two has better average-case complexity? If not, why do you think that is? Is there some other kind of cost that is not being accounted for here, which might explain the discrepancy? **[3]**

2. Consider the below graph.



   (a) Using the MST property, show that the edge (c, d) must be part of some MST. **[1.5]**

   (b) Show the execution of Kruskal's algorithm on the above graph. Initially, and at the end of each iteration, draw the following: (i) the current spanning forest (set of connected components); (ii) the current state of the priority queue (as a min-heap or partially ordered tree). After the final iteration, (i) should correspond to the obtained MST. **[3]**

3. The *Fibonacci numbers* are defined as follows:

$$F_0 = 0;$$
$$F_1 = 1;$$
$$F_n = F_{n-1} + F_{n-2},\ \forall n \geq 2.$$

(a) Give a recursive algorithm for computing the n^{th} Fibonacci number which directly makes use of the above recurrence relation. Write clear pseudocode, and analyse the time complexity of your algorithm. **[1.5]**

(b) Illustrate the execution of the algorithm for $n = 5$, by drawing a recursion tree of all the recursive calls made, and how the values returned from the base cases are propagated up the tree. **[1.5]**

(c) Based on the answer to (b), do you think this is the most efficient way to compute Fibonacci numbers? If not, why not? Point out specifically where you think the inefficiency is. **[1]**

(d) Which algorithm design strategy discussed in class can be used to solve this problem more efficiently? Using this strategy, give an $O(n)$ algorithm to compute the n^{th} Fibonacci number. Write clear pseudocode, and explain why its time complexity is $O(n)$. Illustrate the execution of the algorithm for $n = 5$, clearly showing the sequence of subproblem solutions that are computed, stored, and re-used. **[3]**