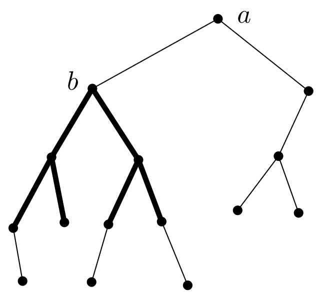
DATA STRUCTURES AND ALGORITHM ANALYSIS
COMP 3804

Assignment 2

Date Due: Nov 6, 2020
Time Due: 6pm
Your assignment should preferably be typed and should be submitted online on CuLearn.

1. (9 pts: 3 pts description of algorithm, 3 pts analysis of recursion, 3 pts correctness) A rooted binary tree is a complete tree if every leaf has exactly the same depth d (i.e. the length of the path from the root to a leaf is d for every leaf) and there are 2^d leaves. A subtree is a connected subgraph of the tree. Notice that the subtree is also rooted. Describe and analyze a recursive algorithm to compute the largest complete subtree of a given binary tree. Your algorithm should return the root and the depth of this subtree. See the figure below:



Given the binary tree rooted at vertex a, the largest complete subtree is highlighted in bold. The algorithm should return b as the root of this subtree and 2 as the depth of the leaves in this subtree.

Figure 1: Example output.

2. (12 pts) Given a sorted array A of n numbers, recall that in $\lg n$ steps, Binary Search can determine whether or not a number x is in A. It does so by comparing x with $A[n/2]$ and recursing on the side that contains x when $x \neq A[n/2]$. Since half the elements of A are removed from consideration at every step, the recurrence for the runtime is $T(n) = T(n/2) + O(1)$ which resolves to $O(\log n)$.

   Suppose that you are given a function BinSearch(A, x) that returns a value i if x is $A[i]$ and -1 otherwise. Now, notice that although it is efficient, sometimes Binary Search takes too much time to return an element. For example, consider a linear search algorithm that searches for x by checking if x is $A[1], A[2], \ldots, A[n]$. This is a terrible algorithm in the worst case, because searching for the maximum element takes $\Omega(n)$ time. However, it is very quick to find the minimum element, much quicker than Binary Search.

   Suppose that if we inserted x into A, the rank of x in the sorted array A would be k. We want to design an algorithm that can find x in $O(\log k)$ time as opposed to $O(\log n)$ time which is the case in Binary Search.

   (a) (6pts) Suppose we are given a second function RankA(A, x) that returns the rank x would have if we inserted it in A in $O(\log k)$ time, where k is the rank of x. Using BinSearch(A, x) and RankA(A, x) as subroutines, design an algorithm called FastSearch(A, x) that determines in $O(\log k)$ time whether x is in A where k is the rank returned by RankA(A, x). FastSearch(A, x) returns the index of x in A if it is in A, and -1 otherwise. You must outline your algorithm, explain why it is correct and explain why its runtime is $O(\log k)$.

   (b) (6pts) Show how you can design RankA(A, x). You must outline the algorithm, explain why it is correct and explain why its runtime is $O(\log k)$.

3. (10 pts) In class, we saw a deterministic $O(n)$ time algorithm to compute the kth smallest element in an unsorted array A of n numbers. The main idea to get the linear time algorithm was to find a good *pivot* element. This was done by partitioning the array A into $n/5$ sets of size 5. Then computing the median of each of these sets of size 5, resulting in a set of $n/5$ medians. The pivot element is the median of these $n/5$ medians and is computed recursively. What if we partition the array A into $n/3$ elements of size 3, instead.

   (a) (3 pts) How many elements are guaranteed to be smaller than the pivot and how many are guaranteed to be larger than the pivot? Explain your reasoning (you do not need to give a formal proof).

   (b) (3 pts) Give the recurrence for the new running time of the algorithm.

   (c) (4 pts) Solve the recurrence using any method of your choice and state the running time of the algorithm in terms of n using Big-Oh notation.

4. (9 pts: 3 pts description of modification, 3 pts analysis of runtime, 3 pts correctness) In class we saw the following algorithm to compute the longest increasing subsequence given an input sequence S of length n.

```
Algorithm LIS(S) {
    initialize array L[1..n] - all entries are set to zero
    maxL = 0
    For j = 1 to n {
        L[j] = 1
        For i = 1 to j-1 {
            If S[i] < S[j] then L[j] = max{L[j], L[i] + 1}
        }
        maxL = max{maxL, L[j]}
    }
    Return maxL
}
```

   Recall from class that the runtime of this algorithm is $O(n^2)$. Modify this algorithm to run in $O(n \log n)$ time.

5. (10 pts) A palindrome is any string that is exactly the same as its reversal, like I, or DEED, or AMANAPLANACANALPANAMA. For example, RACECAR is the longest subsequence that is a palindrome in THRISACEBRMCARZSMRUF, so given this string as input, your algorithm should output the number 7.

   (a) (7 pts) Describe in pseudo-code an $O(n^2)$ time algorithm to find the length of the longest subsequence of a given string that is also a palindrome. The key is to find a recursive description of the optimal solution in terms of the optimal solution of smaller subproblems. If you are unable to find an $O(n^2)$ time algorithm, then for part marks, you can give an $O(n^3)$ or $O(n^4)$ time algorithm.

   (b) (3 pts) Analyze the running time of your algorithm.