CS 381 Introduction to the Analysis of Algorithms
Fall 2021 Simina Branzei and Alexandros Psomas HW 1

Due Fri Sep 10 at 11:59PM

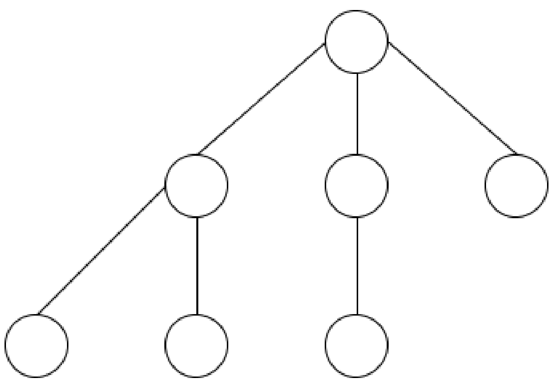
1. (20 points) Define the Foonacci sequence as follows: $f(0) = -1, f(1) = -5, f(2) = 1$, and $f(n) = 3f(n-1) + 4f(n-2) - 12f(n-3)$ for $n > 2$.

   Prove: Using induction (weak or strong), show that $f(n) = 3^n - 3(2^n) + (-2)^n$ for all $n \geq 0$.

2. (20 points) Let T be a tree with $n > 1$ vertices where every vertex has degree at most 3 (the degree of a vertex is the number of edges incident to it). Let $S_d(T)$ be the set of vertices in T that have degree exactly d.

   Prove: Using induction (weak or strong), show that for any such T, $|S_1(T)| - |S_3(T)| = 2$ holds.

   For example, in the example below there are 4 vertices of degree 1, 1 vertex of degree 2, and 2 vertices of degree 3. $|S_1(T)| - |S_3(T)| = 4 - 2 = 2$.



3. (20 points) For the following pairs of functions, relate one function to the other with a big O bound, or with a big Theta bound if applicable. For each pair of functions, state **and prove** whether the first function is big O, Theta, or big Omega of the second function. (If Theta is possible, you must prove Theta).

   - n^3 and $n^3 + 6n^2$
   - $n!$ and n^n
   - $8^{\log_2(n)}$ and n^3
   - $n + \ln(n)$ and $\ln(n^n)$
   - $\log_2(n)$ and $\log_{10}(100n)$
   - $\ln(n^2) - \ln(2n)$ and $\log_2(16^n)/\sqrt{n}$

4. (20 points) For the following code segment, provide a tight big O bound on the number of times "foo()" is called **with a proof** for why this bound is tight:

```
while n > 1 do
    for i = 1 to n
        k = n;
        while k > 1 do
            foo();
            k = k / 3;
    n = n / 9
```

5. (20 points) Let $T(n)$ satisfy the following recurrence

$$T(n) = T(n/2) + 1 \tag{1}$$

Prove: Assuming $T(1) = 0$ and $n = 2^m$ where m is a natural number bigger than zero, show using the telescoping method (or using a recursion tree) that $T(n) = m$. Do not use the Master Theorem.

6. (20 points) Consider the following pseudo code which presents a variant of the merge sort algorithm:

```
variantsort(Array A){
    n = size_of(A);

    if (n == 1){
        return;
    }

    j = 1;
    A1,A2,A3 = [];
    for i = 1 to n/3:{
        A1[j] = A[i];
        j = j+1;
    }

    j = 1;
    for i = n/3+1 to 2n/3:{
        A2[j] = A[i];
        j = j+1;
    }

    j = 1;
    for i = 2n/3 + 1 to n:{
        A3[j] = A[i];
        j = j+1;
    }

    variantsort(A1);
    variantsort(A2);
    variantsort(A3);

    A4 = merge(A1,A2);
```

```
        A5 = merge(A4,A3);

        return A5;
    }
```

Assume that the function C = merge(A,B) takes two sorted arrays A (of size n_1) and B (of size n_2) and combines and returns them as one big sorted array C (of size $n_1 + n_2$) in $n_1 + n_2$ steps.

Solve:

(a) State the recurrence relation for the running time of the above pseudocode for an input of size n (assume n is a power of 3). (5 points)

(b) Solve the recurrence relation you obtained in the previous step. Do not use the master theorem. (15 points)

You can present the recurrence relation and the final solution in big O notation wherever appropriate. Note that you have to use the best possible bound for the big O notation i.e., you have to use O(n^3), not O(n^4) if the algorithm runs in $n^3 + 2n^2$ steps.