Homework Assignment 2

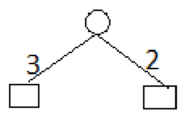
CS 430 Introduction to Algorithms

Name : Karthik Mahesh

CWID : A20383027

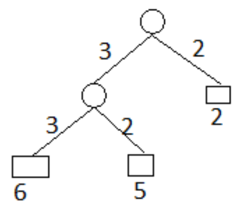
1.Solution:

When we consider an optimal tree, the leaves cannot be weighted much differently.
Let’s consider the tree with 2 nodes



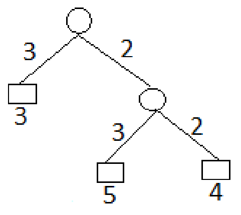
Here, cost of the tree is 3 and this is an optimal tree and no changes can be done

Let’s consider the tree with 3 nodes



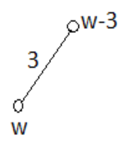
Here, cost of the tree is 6 and this is not an optimal tree.

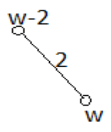
This can be changed to



And the cost of the tree will be 5 instead of 6.
So the cost difference of greater than 3 will not be an optimal tree.
So weights w, w+2, w+3 can be present in the optimal tree

To find the maximum number of leaf nodes of cost w
We can generate a node of weight w either from





w-3 or w-2

Let M_w be the maximum number of nodes of weight w.
Since a node of cost w is derived from nodes of cost w-2 or w-3
We get

$$\mathbf{M_w = M_{w-2} + M_{w-3}}$$

3.Solution:

As given in the hint, this problem can be solved by using merge sort technique. First we have to recursively partition the sequence of n elements into two sub sequences having the same or nearly equal size until only one element is left in each subsequence. After that the merging takes place. We can use the merging process in the merge sort and record the total number of significant inversions during the merge. The correctness comes from that the elements in the left subsequence have smaller indices than the elements on the right subsequence. We can add a condition to see if ai > 2aj during the merging process.

```
Sort-and-Count(L)
    If the list has one element then
        there are no inversions
    Else
        Divide the list into two halves:
          A contains the first n/2 elements
          B contains the remaining n/2 elements
        ($r_A$, A) = Sort-and-Count(A)
        ($r_B$, B) = Sort-and-Count(B)
        (r, L) = Merge-and-Count(A, B)
    Endif
Return r = $r_A + r_B + r$, and the sorted list L
```

```
Merge-and-Count(A,B)
    Maintain a Current pointer into each list, initialized to point to the front elements
    Maintain a variable Count for the number of inversions, initialized to 0
    While both lists are nonempty:
        Let $a_i$ and $b_j$ be the elements pointed to by the Current pointer
        Append the smaller of these two to the output list
        If $2b_j < a_i$ then
            Increment Count by the number of elements remaining in A
        Endif
    Advance the Current pointer in the list from which the smaller element was selected.
    EndWhile
Once one list is empty, append the remainder of the other list to the output
Return Count and the merged list
```

Merge-and-Count procedure takes O(n) time

The Sort-and-Count algorithm correctly sorts the input list and counts the number of inversions. It runs in O(n log n) time for a list with n elements.

2.Solution:

b.

T(n) = 2T(n - 1) + 3T(n - 2)

$a_n = 2a_{n-1} + 3a_{n-2}$

The annihilator is E^2-2E-3

The annihilator factors into (E-ϕ) (E-ϕ’)

The generic solution is $a_n = \alpha \phi^n + \beta \phi'^n$
The constants α, β satisfy the equation
$a_0 = 0 = \alpha+\beta$
$a_1 = 1 = \alpha \phi + \beta \phi'$
$a_2 = 2 = \alpha\phi^2 + \beta\phi'^2$

Solving the equations gives us
$\alpha = \frac{\sqrt{5}+2}{\sqrt{5}}, \beta = \frac{\sqrt{5}-2}{\sqrt{5}};$

$$a_n = \frac{\sqrt{5}+2}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n + \frac{\sqrt{5}-2}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

Therefore **$T(n) = O(2^n)$**

f.
$T(n) = T(\sqrt{n}) + 3n$
Let k be a variable such that $2^k = n$
$T(2^k) = T(2^{k/2}) + 3*2^k$
Let $s(k) = T(2^k)$
$S(k) = S(k/2)+3.2^k$
By master theorem
$a=1, b=2, f(n)=3.2^k$
$n\hat{}\log_b a = 1$
$1 < 3.2^k$
case 3 applies
Therefore $T(n) = \Theta(n)$ as $n = 2^k$ and ignoring 3 as some constant c.

References used : Algorithm Design by Jon Kleinberg, Eva Tardos
Analysis and Design of Algorithms by Padma Reddy