Ömer GEVİK
161044004

CSE 321
Introduction to Algorithm Design
HOMEWORK 2

(1)

a) $T(n) = 27T(n/3) + n^2 \Rightarrow$ Using Master Theorem $T(n) = aT(n/b) + n^d$ $\Rightarrow a = 27,\ b = 3,\ d = 2$ $\quad a \overset{?}{\lesseqgtr} b^d \Rightarrow a > b^d \Rightarrow 27 > 3^2$ $\quad \log_3 3^3 = 3$
$\Rightarrow T(n) = \Theta(n^{\log_b a}) = \Theta(n^3)$

b) $T(n) = 9T(n/4) + n \Rightarrow$ 〃 $\Rightarrow a = 9,\ b = 4,\ d = 1$ $\quad a \overset{?}{\lesseqgtr} b^d \Rightarrow a > b^d$, $9 > 4^1 \Rightarrow T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_4 9})$

c) $T(n) = 2T(n/4) + \sqrt{n} \Rightarrow$ 〃 $\Rightarrow a = 2,\ b = 4,\ d = 1/2$ $\quad a \overset{?}{\lesseqgtr} b^d \Rightarrow a = b^d$, $2 = 4^{\frac{1}{2}} \Rightarrow T(n) = \Theta(n^d \cdot \log n) = \Theta(n^{\frac{1}{2}} \log n)$

d) $T(n) = 2T(n/2) + 17 \Rightarrow$ 〃 $\Rightarrow a = 2,\ b = 2,\ d = 0$ $\quad a \overset{?}{\lesseqgtr} b^d \Rightarrow a > b^d$, $2 > 2^0 \Rightarrow T(n) = \Theta(n^{\log_b a}) = \Theta(n)$ $\quad \Rightarrow \log_2 2 = 1$

e) $T(n) = 2T(\sqrt{n}) + 1 \Rightarrow T(n) = 2T(n^{\frac{1}{2}}) + 1 \Rightarrow$ Let $x = \log n \Rightarrow T(2^x) = 2T(2^{\frac{x}{2}}) + 1$, $n = 2^x$
Let $\Rightarrow Y(x) = T(2^x) \Rightarrow Y(x) = 2Y(x/2) + 2 \Rightarrow$ Using Master Theorem $\Rightarrow a = 2,\ b = 2,\ d = 0$ $\quad a \overset{?}{\lesseqgtr} b^d \Rightarrow a > b^d$, $2 > 2^0 \Rightarrow Y(x) = \Theta(n)$ $\quad \log_2 2 = 1$
$\Rightarrow T(2^x) = \Theta(n) \Rightarrow T(n) = \Theta(\log n)$

f) $T(n) = 4T(n/2) + n \Rightarrow$ Using Master Theorem $T(n) = aT(n/b) + n^d$ $\Rightarrow a = 4,\ b = 2,\ d = 1$ $\quad a \overset{?}{\lesseqgtr} b^d \Rightarrow a > b^d$, $4 > 2^1 \Rightarrow T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$ $\quad \Rightarrow \log_2 4 = 2$

g) $T(n) = T(n/3) + T(2n/3) + O(n) \Rightarrow$

$$
\begin{array}{ccccl}
 & & cn & & = cn \\
 & c(n/3) & & c(2n/3) & = cn \\
c(n/9) & c(\frac{2n}{9}) & c(\frac{2n}{9}) & c(\frac{4n}{9}) & = cn
\end{array}
$$

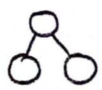
On each level, we obviously obtain cn operations, independent of the level. The longest path $\Rightarrow n \to 2/3n \to (2/3)^2 n \to \cdots \to 1$ the height of the tree. $(2/3)^h \cdot n = 1 \Rightarrow h = \log_{\frac{3}{2}} n$. We could expect the total cost is $cn \cdot \log_{3/2} n$

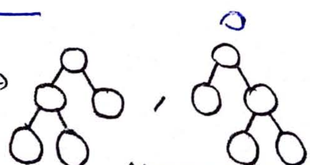
$T(n) = O(n \log n)$

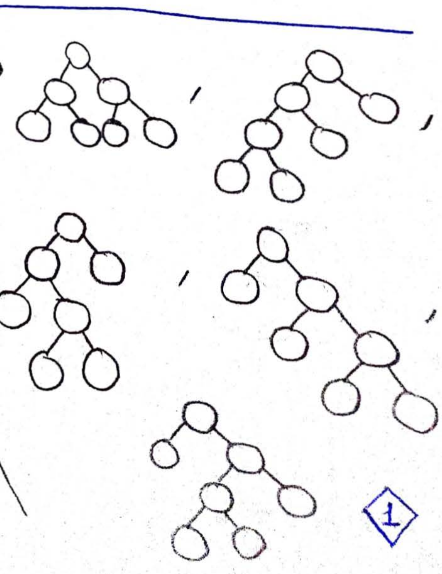
h) $T(n) = T(n-1) + n^c$, where $c > 0$ and c is a constant $\Rightarrow$ Let assume $c = 1$ $\quad T(n) = T(n-1) + n^c = T(n-2) + (n-1) + n$
$= T(n-3) + (n-2) + (n-1) + n = \cdots = T(1) + (2 + 3 + \cdots + n) = T(0) + (1 + 2 + \cdots + n) = T(0) + n * (n+1)/2$
$= n^2 \Rightarrow T(n) = n^{1+1} = n^{c+1} \Rightarrow T(n) = n^c + (n-1)^c + (n-2)^c + \cdots + 3^c + 2^c + 1 = n^{c+1} \Rightarrow T(n) = O(n^{c+1})$

i) $T(n) = T(n-1) + c^n$, where $c > 0$ and c is a constant $\Rightarrow T(n) = T(n-1) + c^n = T(n-2) + c^{n-1} + c^n$
$= T(n-3) + c^{n-2} + c^{n-1} + c^n = \cdots = T(1) + (c^2 + c^3 + \cdots + c^n) = T(0) + c + c^2 + \cdots + c^n = \frac{c^{n+1} - 1}{c - 1} \Rightarrow T(n) = O(c^n)$

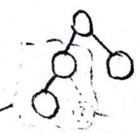
---

(2) a) $B_3 \Rightarrow$  1

, $B_5 \Rightarrow$  2

, $B_7 \Rightarrow$  5

We left out even numbers of vertices. Because even numbers don't generate "full binary" trees. For example B_4:



Not full

Full binary trees say that each node must have 2 children or no children.

Ömer GEVİK
161044004

2

b) $B_n = \sum_{i=1}^{n-2} B_i \cdot B_{(n-i-1)}$ $\Rightarrow$ For a tree with n nodes, the number of nodes in left subtree and right sub tree will be like $1,3,5,\dots,n-2$ and $n-2, n-4, \dots, 1$ respectively. That formula explains the recurrance relation.

c) Showing the $B_n \geq c.2^n$ for all $n \geq n_0$ where $c, n_0 > 0$ will be explained for average case.
Let's assume $B_k \geq c.2^k$ for all odd values $1,3,\dots,k$.
So $B_{k+2} \geq c.2^{k+2}$?

$\Rightarrow B_{k+2} = \sum_{i=1}^{k} B_i \cdot B_{(k-i+1)} \Rightarrow B_{k+2} \geq c.2^i.c.2^{k-i+1}.(k+1)/2$

$\Rightarrow B_{k+2} \geq c.2^{k+1}.\frac{c(k+1)}{2}$ $\Rightarrow$ If we select c and n_0 and $\frac{c(k+1)}{2} \leq 2$ and we solve it.

$\Rightarrow B_{k+2} \geq c.2^{k+2}$ Then we see it the average case: $\Omega(2^n)$

Let's assum $B_k \leq c.2^k$ for all odd values $1,3,\dots,k$.
So $B_{k+2} \leq c.2^{k+2}$?

$\Rightarrow B_{k+2} = \sum_{i=1}^{k} B_i B_{(k-i+1)} \Rightarrow B_{k+2} \leq c.2^i.c.2^{k-i+1}.(k+1)/2$

$\Rightarrow B_{k+2} \leq c.2^{k+1}.c.\frac{(k+1)}{2}$ $\Rightarrow$ If we select c and n_0 and $c\frac{(k+1)}{2} \geq 2$ and we solve it.

$\Rightarrow B_{k+2} \leq c.2^{k+2}$ Then we see it the average case: $O(2^n)$

In the result; $\Omega(2^n)$ and $O(2^n) \Rightarrow \Theta(2^n)$

0

3

a) $T(n) = 7T(n/2) + \Theta(n)$

$\Rightarrow$ Using Master Theorem $\Rightarrow T(n) = aT(n/b) + n^d \Rightarrow a \overset{?}{\lesseqgtr} b^d \Rightarrow$ $a=7$, $b=2$, $d=1$; $a > b^d$, $7 > 2^1 \Rightarrow T(n) = \Theta(n^{\log_b a})$

$\Rightarrow T(n) = \Theta(n^{\log_2 7})$

b) $T(n) = 2T(n-1) + \Theta(n) = 2T(n-1) + n = 2^2T(n-2) + (n-2) + n = 2^3T(n-3) + (n-3) + (n-2) + n = \dots =$
$\Rightarrow 2^{n-1}T(1) + cn = 2^nT(0) + cn \Rightarrow T(n) = O(2^n)$

c) $T(n) = 4T(n/2) + T(n^2) \Rightarrow$ Using Master Theorem $\Rightarrow$ $a=4$, $b=2$, $d=2$; $a \overset{?}{\lesseqgtr} b^d \Rightarrow 4 = 2^2 \Rightarrow T(n) = \Theta(n^d \lg n)$

$\Rightarrow T(n) = \Theta(n^2 \lg n)$

$\Rightarrow$ I would choose algorithm A. Becouse $T(A) < T(C) < T(B)$

0

4

Pseudo code for algorithm:

```
f(G):
    while( G.length > 2 ): do
        Vertices = random(G.keys())
        weights = random(G[vertices])
        merge(G, vertices, weights)
    end while
    return G[0]
end
```

```
merge(G, vertices, weights):
    for w in G[weights] do
        if w != vertices then
            G[vertices].append(w)
        end if
        G[w].remove(weights)
        if w != vertices then
            G[w].append(vertices)
        end if
    end for
    delete G[weights]
end
```

2

Ömer GEVİK
161044004

4 Time Complexity Analysis

For Best Case Scenerio: If the graph nodes are less than or equal to 2 then merge method and the while loop will not be executed. Then the cost is going to be constant time.
$T(n) = O(1)$

For Worst Case Scenerio: Flow of the algorithm shows us f(G) method inside while loop executes n times. In that while loop merge() method call becomes. There is a for loop in that method. We can assume that for loop can be executed n times. Consequently, we have nxn times in cost. It equals to quadratic time.
$T(n) = O(n^2)$

For Average Case Scenerio: Flow of the f(G) method we know the while loop must be executed as n times. The merge() method has a for loop but if we get the average we can assume the for loop is going to execute logn times like a tree. Finally, we have nxlogn times in cost.
$T(n) = O(n\log n)$

5
```
function f(n):
    res=0
    if n ≤ 1:
        res=1
    else:
        for i in range(n):
            res+=f(i)*f(n-i-1)
    print(res)
    return res
```

⇒ Recurrance relation ⇒ $\sum_{i=0}^{n} f(i).f(n-i-1)$ times print executed.

We see that $f(i)$ and $f(n-i-1)$ definitely works reversly the same result.

We can write it like $2\sum_{i=0}^{n-1} f(i)$. ⇒ $2\sum_{i=0}^{n-1} T(i)$ for time complexity.

$T(0) = 1$
$T(1) = 1$
$\vdots$
$T(n-1) = T(n-2)*T(0) + T(n-3)*T(1) + \dots + T(1)*T(n-3) + T(n-2)*T(0) = 2(T(n-2) + T(n-3) + \dots + T(0))$
$T(n-2) = T(n-3)*T(0) + T(n-4)*T(1) + \dots + T(1)*T(n-4) + T(n-3)*T(0) = 2(T(n-3) + T(n-4) + \dots + T(0))$

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

for $n = 3 \Rightarrow T(2) = 3T(1) = 3^1$ $\quad n-1$
'' $n = 4 \Rightarrow T(3) = 3T(2) = 3(3T(1)) = 3^2T(1) = 3^2$ $\quad n-1$
$\vdots$
'' $n = n \Rightarrow T(n) = 3T(n-1) = 3^{n-1}$

Consequently, function f(n) is running in 3^{n-1} times and it prints res value 3^{n-1} times.