CA - II ( Design and analysis of algorithm)

NAME - KANKAN PARAMANIK

ROLL NO. - 19CS8148

1) (a) Solution :

choosing $n$, such that $n \geq n_1$ which implies $n/2 + 17 \leq 3n/4$.
We'll find $c$ and $d$ such that $T(n) \leq cn\log n - d$.

$$T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$$
$$\leq 2(c(n/2+17)\log(n/2+17) - d) - n$$
$$\leq cn\log(n/2+17) + 17c\log(n/2+17) - 2d + n$$
$$\leq cn\log(3n/4) + 17c\log(3n/4) - 2d + n$$
$$= cn\log n - d + cn\log(3/4) + 17c\log(3n/4) - d + n$$

Taking $c = -2/\log(3/4)$ and $d = 34$. Then we have

$T(n) \leq cn\log n - d + 17c\log(n) - n$.

Since ~~$\log n = O(n)$~~ $\log n = O(n)$, there exists $n_2$ such that $n \geq n_2$ implies $n \geq 17c\log n$.

Let $n_0 = \max\{n_1, n_2\}$ we have that $n \geq n_0$ implies

$T(n) \leq cn\log n - d$.

Therefore, $T(n) = O(n\log n)$.

(Proved)

(b) Solution :

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$$

By ~~sb~~ substitution method.

We guess that $T(n) \leq cn - d$ where $d \geq 0$ is a constant

Now

$$T(n) \leq (c\lfloor n/2 \rfloor - d) + (c\lceil n/2 \rceil - d) + 1$$

$$= cn - 2d + 1$$

$$\leq cn - d \quad [\text{for } d \geq 1]$$

So $T(n) = \Theta(n)$.

If we would have chosen $T(n) \leq cn$ instead of $T(n) \leq cn - d$

then

$$T(n) \leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1$$

$T(n) \leq cn + 1 \not\Rightarrow T(n) \leq cn$ ~~so~~ ~~contradi~~

so $T(n) \leq cn - d$ is assumed.

(Proved)

---

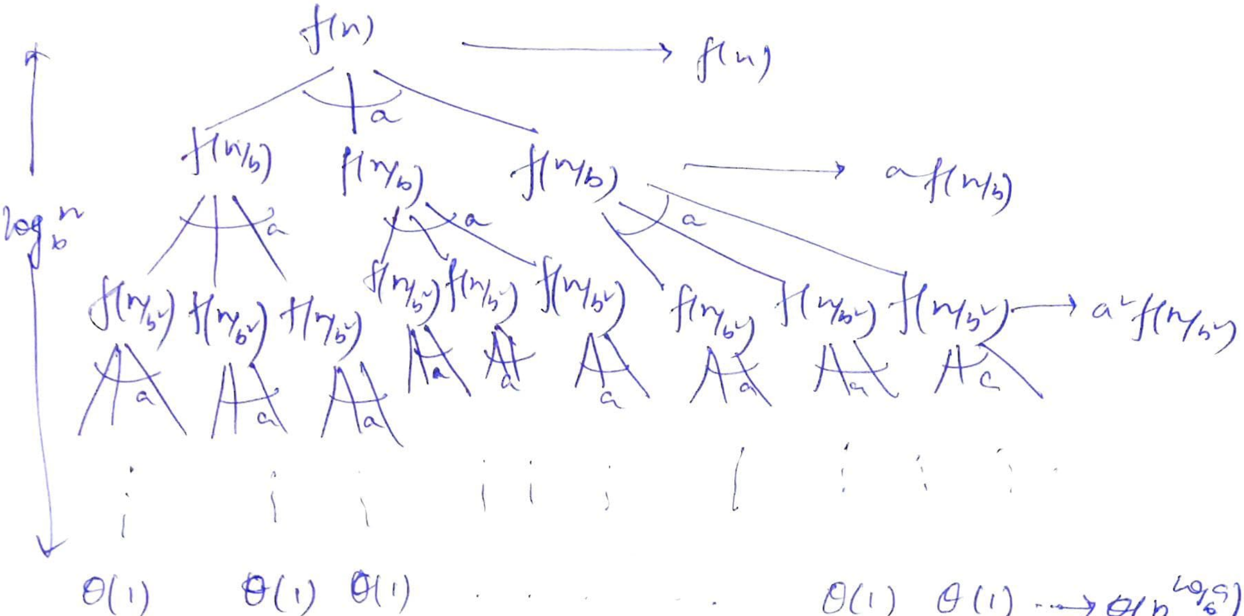
(c) Solution:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ aT(n/b) + f(n) & \text{if } n = b^i, \end{cases}$$

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b a - 1} a^j f(n/b^j) \qquad (1)$$

We use recursion tree, The root of the tree has cost $f(n)$, and it has $a$ children each with cost $f(n/b)$ and each children have $a$ ~~ch~~ children with cost $f(n/b^2)$, and thus there are $n^2$ nodes that are distance 2 ~~from~~ from the root. In general there are $a^j$ nodes that are at a distance $j$ from root and each has cos $f(n/b^j)$

cost of each leaf is $T(1) = \Theta(1)$ and each leaf is at depth $\log_b n$ since $n/b^{\log_b a} = 1$. There are $a^{\log_b n} = n^{\log_b a}$ leaves in the tree.




$f(n) \longrightarrow f(n)$

$\longrightarrow a f(n/b)$

$\longrightarrow a^2 f(n/b^2)$

$\longrightarrow \Theta(n^{\log_b a})$

Total $\Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$

We can obtain eq. (1) by:

summing the cost of each level of the tree.

The cost for a level $j$ of internal nodes is $a^j f(n/b^j)$, and so the total of all internal node levels is.

$$\sum_{j=0}^{\log_b n - 1} a^j f(n/b^j) = g(n) \text{ (let)}$$

this can be bounded asymptotically for exact powers of b.

1. if $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$ then.
   $g(n) = O(n^{\log_b a})$

2. if $f(n) = \Theta(n^{\log_b a})$ then $g(n) = \Theta(n^{\log_b a} \log n)$.

3. if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and for all $n \geq b$, then $g(n) = \Theta(f(n))$

~~Proof~~

Case ①

$$g(n) = \Theta\left( \sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \varepsilon} \right)$$

Now,

$$\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \varepsilon} = n^{\log_b a - \varepsilon} \sum_{j=0}^{\log_b n - 1} \left( \frac{a b^{\varepsilon}}{b^{\log_b a}} \right)^j$$

$$= n^{\log_b a - \varepsilon} \sum_{j=0}^{\log_b n - 1} \left( b^{\varepsilon} \right)^j$$

$$= n^{\log_b a - \varepsilon} \left( \frac{b^{\varepsilon \log_b n} - 1}{b^{\varepsilon} - 1} \right)$$

$$= n^{\log_b a - \varepsilon} \left( \frac{n^{\varepsilon} - 1}{b^{\varepsilon} - 1} \right)$$

So $g(n) = O(n^{\log_b a})$ $\because$ $b$ and $\varepsilon$ are constants.

So, now $T(n) = \Theta(n^{\log_b a}) + O(n^{\log_b a}) = \Theta(n^{\log_b a})$

Case ②

$$g(n) = \Theta\left( \sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a} \right)$$

Now,

$$\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a} = n^{\log_b a} \sum_{j=0}^{\log_b n - 1} \left( \frac{a}{b^{\log_b a}} \right)^j$$

$$= n^{\log_b a} \sum_{j=0}^{\log_b n - 1} 1$$

$$= n^{\log_b a} \log_b n$$

$\therefore$ $g(n) = \Theta(n^{\log_b a} \log_b n)$

So now, $T(n) = \Theta(n^{\log_b a}) + \Theta(n^{\log_b a} \log n)$

$= \Theta(n^{\log_b a} \log n)$.

(Proved)

2) a) ~~Solution~~:

Let us say a thief is robbing a grocery store. He can carry a maximum weight of $W$ in ~~the~~ his knapsack. There are $n$ different grocery items and weight of $i$th item is $w_i$ and profit is $p_i$. The thief can apply fractional knapsack.

Since grocery items can be broken down he ~~can~~ can select fractions.

- $n$ items in the store
- $w_i > 0$ • $p_i > 0$
- capacity $= W$

fraction of item is $x_i$

$$0 \le x_i \le 1$$

$i$th fraction has weight $x_i w_i$ and profit $x_i p_i$

objective is to maximize $\left(\sum_{i=1}^{n} (x_i p_i)\right)$

constraint is $\sum_{i=1}^{n} (x_i w_i) \le W$

Optimal solution is to fill the knapsack completely

$$\sum_{i=1}^{n} x_i w_i = W$$

we need to sort the items according to profit per unit weight $p_i/w_i$ so that $\frac{p_{i+1}}{w_{i+1}} \le \frac{p_i}{w_i}$

5) Solution:

Pseudo code (for given example)

```
greedy-fractional knapsack (w[1,...n], p[1...n], W)
{
    for i=1 to n
        do x[i] =0
    weight =0
    for (i=1 to n)
        if (weight + w[i] ≤ W)
            x[i] =1
            weight =weight + w[i]
        else
            x[i] = (W - weight) / w[i]
            weight = W
            break.
    return x.
}
```

Analysis:

Assuming that the parameter array $w[1..n]$ is ~~ater~~ already sorted in decending order for $P_i / w_i$. This sorting takes an upperbound of $O(n \log n)$.

And the loop takes $O(n)$ time for adding the fractions in knapsack.

$$O(n\log n + n) = O(n \log n) \text{ (Ans)}.$$

The solution is obtimal as we have maximum profit and whole knapsack is filled.

$$\therefore \sum_{i=1}^{n} x_i w_i = W \text{ and } \max\left(\sum_{i=1}^{n} x_i P_i\right)$$

c) Solution:

Suppose we add a decrement operation to the k-bit counter with k-bits, the Increment operation is $O(k)$ in the worst case (flip all ones to zeros).

with Decrement, the worst case is also $O(k)$ (flip all zeros to ones)

Ex: 100 . . 00 $\Rightarrow$ 011 . . . 1

So, given a sequence of n operations consisting of ~~alternat~~ alternating Increment and Decrement operations, the time would be $\Theta(nk)$.

---

d) Solution:

By defination $o(g(n))$ is a set of functions $f(n)$ such that $0 \le f(n) < c_1 g(n)$ for any positive constants $c_1 > 0$ and for ~~f~~ all $n \geq n_0$.

$\omega(g(n))$ is a set of all function $f(n)$ such that $0 \le c_2 g(n) < f(n)$ for any $c_2 > 0$ and for all $n \geq n_0$.

So ~~o~~ $o(g(n)) \cap \omega(g(n))$ is a set of all function $f(n)$ such as

$$0 \le c_2 g(n) < f(n) < c_1 g(n)$$

The inequality can't be true asymptotically as n becomes very large $f(n)$ can't be simultaneously greater than $c_2 g(n)$ and less than $c_1 g(n)$ for $c_1, c_2 > 0$.

Hence no such $f(n)$ exists.

$\therefore o(g(n)) \cap \omega(g(n)) = \phi$.

proved/