Midexam #2 (Total sheets-3)

Srinivasa Vamsi Bhargav Venuri
A.S.U.Id - 1225196596

(6.) Part-A

Since we have to check e in both the equal sized subarrays A_1 & A_2, we have to make a total of two recursive calls with half the original size array and also receive some constant time for checking and returning Boolean condition. So recursive relation will be;

$$T(n) = \begin{cases} 2T(n/2) + 1 & \text{otherwise} \\ 1 \text{ or } c & \text{if } n = 1 \end{cases}$$

and when size of array is 1 it will take constant time to check and return

$$\Rightarrow T(1) = 1$$

for the given problem the recurrence relation is given as

$$T(n) = \begin{cases} 2T(n/2) + 1 & \text{otherwise} \\ 1 & \text{for } n=1 \end{cases}$$

(P.T.O)

Part-2 :- This can be solved in three ways.

Best case for $n=1 \Rightarrow T(n)$ where $n=1$;

Let $T(n) = O(n)$ true for n. Now for

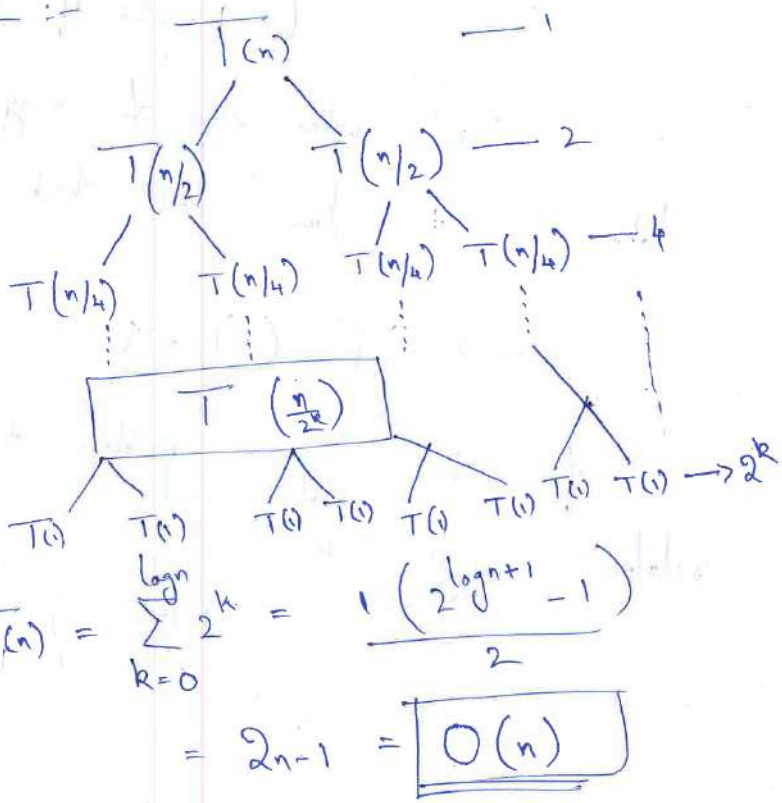
$$T(2n) = 2T\left(\frac{2n}{2}\right) + 1$$

$$= 2(n) + 1$$

$$= \boxed{O(n)}$$

Hence proved $O(n) = T(n)$

Way / Method-2 :-



$$T(n) = \sum_{k=0}^{\log n} 2^k = \frac{1\left(2^{\log n+1} - 1\right)}{2}$$

$$= 2n - 1 = \boxed{O(n)}$$

Way / Method-3 General form

$$T(n) = 2T(n/2) + 1$$

$$= 4T(n/4) + 2 + 1$$

$$\vdots$$

$$= 2^k T\left(n/2^k\right) + \left(2^{k-1} + \cdots + 1\right)$$

Our base case occurs when $\frac{n}{2^k} = 1 \Longrightarrow k = \log_2 n$

$$\Longrightarrow T(n) = \frac{1\left(2^{\log n + 1} - 1\right)}{2 - 1} = 2^{(n)} - 1 = \underline{O(n)}$$

(7.) Winning Set :- Let OPT (i, x) that evaluates true if there exists a subset of elements $1 \ldots i$ where sum is exactly x, otherwise false.

Case-1 OPT does not select the i^{th} element

→ Here we check for subset of $\{1, 2 \ldots i-1\}$ where sum is x.

Case-2 OPT selects item i.

→ New sum will be $x - s_i$ [Here $x \geq s_i$ otherwise Case-2 is not possible]

→ OPT selects best of $\{1, 2, \ldots i-1\}$ using this new sum limit.

$$OPT(i,x) = \begin{cases} 1 & i=0 \ \&\& \ x=0 \\ 0 & i=0 \ \&\& \ x \neq 0 \\ OPT(i-1,x) & \text{if } s_i > x \\ OPT(i-1,x) \vee OPT(i-1,x-s_i) & \text{otherwise.} \end{cases}$$

Part-B

Let total numbers in s be n.

Pseudo Code :-

```
Input : n, s1 .... sn, x
        for i = 0 to n
        for j = 0 to X {
            if (i == 0) { if j == 0 ) M[i][j] = 1;
                          else M[i][j] = 0;
        }
```

```
else M [i] [j] = empty ;
    }
    M - compute (n, X);

M - Compute (i, x) {
        if (M [i] [x] is empty ) {
    if (x < Si) M [j] [x] = M - Compute (i-1, x);
    else M [i] [x] = (M-Compute (i-1, x) V M-Compute
                                          (i-1, x-Si))
        }
    return M [i] [x];
        }
```

Part-C In the above problem we are using memorization. So we will have to fill that 'M' table which takes O (n x) time complexity in the worst case.

→ Time complexity O (n x) which is pseudo polynomial because X sometimes can be written as exponential power of n.