HOME WORK - 3

Name: Sai Rohit Kalyan Gandham.

StudentID: 1008070724

EmailID: Sxg0724@mavs.uta.edu

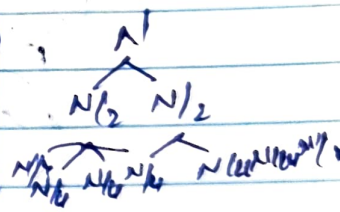
Sol) 4-2:

Given an array is passed by pointer. Time = O(1)
* an array by copying is Time $\geq \Omega(N)$ N: Size.
* an array passed by copying and Subrange that might be accurd procedure Time = $\Theta(q-p+1)$
Subarray A[p..q]

a) Given as per then $T(n) = T(n/2) + c \cdot \Theta(\lg n)$

2. $O(n \lg n)$

$T(n) = T(n/2) + cN$
$= 2cN + T(n/4)$
$= 3cN + T(n/8)$
$= \sum_{i=0}^{\lg n - 1} (2^i cN/2^i)$
$= cN \lg N$
$= O(n \lg n)$



3. $T(n) = T(n/2) + cn$ $\quad aT(n/b) + \Theta(N)$

master method

$n^d = n^1 \quad d=1, a=1, b=2$

$\log_b a > d \Rightarrow \log_2 1 = 0$

$0 < d \Rightarrow f(n) \Rightarrow O(f(n))$

$= O(n)$.

b) Master's method.

Sol) $T(n) = 2T(n/2) + cn$

Given $2T(n/2) + cn$ compare with master method formal. $aT(n/b) + cn$.

$a = 2, b = 2$ and $f(n) = c(n)$.

$\Rightarrow n^d = n^1 \Rightarrow \boxed{d = 1}$

$\Rightarrow \log_b a, \log_2 d = 1$

$d = 1 = \log_b a = \log_2 = 2$ so $n \log n$

$n \log n$ Hence proved

② Given $T(n) = 2T(n/2) + cn + 2N$

By Not master theorm so change using Substion method.

$T(n) = 2T(n/2) + cn + 2N \rightarrow ①$

$T(n/2) = 2T(n/4) + \frac{c(n)}{2} + 2N \rightarrow ②$

$T(n/4) = 2T(n/8) + c(n/4) + 2N \rightarrow ③$

Now Eq ② into Equation ①.

$T(n) = 2(2T(n/4) + cn + 2N) + cn + 2N$

$= 4T(n/4) + 2c(n/2) + 4N + cn + 2N.$

$= 4T(n/4) + 2c(n/2) + 6N + cn$

$= 4T(n/4) + 4c(n/2) + 6N.$

$\hookrightarrow$ Eq ③

$\Rightarrow 4T(n/4) + 2c(n/2) + cn + 4N$

$2\{2T(n/8) + c(n/4) + 2N) + 2c(n/2) + cn + 4N)$

$8T(n/8) + 4c(n/4) + 8N + 2c(n/2) + cn + 4N$

$8T(n/8) + 4c(n/4) + 2cn + 8N$

$= \sum_{i=0}^{\lg n - 1} 2^3(n/2^3) + 2^2 c(n/2^2) + 2^1 cn + 2^3 N)$

$= \sum_{i=0}^{\lg n - 1} 2^i N + cn$

$= cn \lg n + nN - N = O(nN)$

Sol) ③ $T(n) = 2T(n/2) + cn + 2(n/2)$

$= 2T(n/2) + n(c+1)$ → master theorm.

Compare with master thoorm.

$aT(n/b) + f(n)$

$\Rightarrow a = 2, b = 2, n^d = n^1 \Rightarrow d = 1$

$\log_b a = d \Rightarrow \log_2 2 = d = 1$ all Equal

Ans ② $(n^d \log n) \Rightarrow n^1 \log n$

$: O(n \log n)$

7.2

Sol) Given problem Statement of is Support all Element values are Same?
To find: Randomized Quicksort Running time?

Sol) The partition value (or) of the Randomized Quicksort is return's (or) because all Element Same. look Sudo Code: in python.

```
def partitionIndex(nums, start, stop):
    pivot = start
    i = start + 1
    for j in range(start+1, stop+1):
        if num[j] ≤ num[pivot]
            (num[i], num[j] = num[j], num[i])
            i = i+1
    (num[pivot], num[i-1] = num[i-1], num[pivot])
    pivot = i-1
    return pivot
def randomizedpartition(arr, start, stop)
    randompivot = random.randrange(start, stop)
    (arr[start], arr[randompivot]) =
        (arr[randompivot], arr[start])
    return partitionIndex(arr, start, stop).
def QuickSort(arr, start, stop):
    if (start < stop):
        pivotIndex = partitionrandom(arr, start, stop)
```

Quicksort (arr, Start, pivotIndex - 1)
Quicksort (arr, Start pivotIndex - 1, Stop)

→ So here each Split will be (n-1) - to - 1
So

$\Theta(n^2)$.

(b) like as per the above Code of Randomized Quick
Sol) Sort,

→ The new(partition)' is also Similar to the partition, Except One when you arrange the Elements according to pivot q.
it moves all elements [t > q : A[t] = r]
from the right of the pivot.

→ So what happens is the procedure also Send Second pass at this array, So
the $\Theta(n) + \Theta(n) \Rightarrow \Theta(n) = \Theta(r-p)$

(c) → Since all the Elements are Same we have Compare Best Case instead of Worst Case.

→ The QuickSort works on divide and Conquer.
So the Equation looks like this

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

→ applying master theorem.

$$aT(n/b) + n^d$$

→ $a = 2$, $b = 2$ and $d = 1$

→ $\log_b a = \log_2 2 = 1$ and also $d = 1$ Case(2)

* ie $n^d = f(n)$ $\quad f(n) = n\log(n)$

$n^d = n^1 \log(n)$

$\Rightarrow O(n\log(n))$.

(d) This is code please find below.

Ans 4)

(c) The modified code for Tail-recursion-QuicuSort so that worst case stack dept will be $O(\lg n)$ maintain the $O(n \lg n)$ Expected running time of the algorithms.

Sol)

Original Tail Recursive QuicuSort

* The pseudo code is

Tail Recursive QuicuSort (A, p, r)

while $p < r$

q : partition (A, p, r)

Tail. recursive - Quution $(A, p, q-1)$

$p = q+1$

Modified Tail Recursive QuickSort

```
Optimised-Tail-Recursive-QuickSort (A,p,r)
    while p<r
    if  q < floor ((p+r)/2)
        New-Tail-Recursive-QuickSort(A,p,q-1)
            p=q+1
    Else
        New-Tail-Recursive-QuickSort (A,q+1,r)
            r=q-1
```