CS 4310 - Algorithms – Spring 2022

HomeWork2

1. (25pts; Basic D&C) Assuming that the pivot is the 3rd element of a list and a list of at most two elements can be sorted directly, simulate the execution of Quicksort on the following input:

   24, 29, 18, 8, 27, 9, 21, 89, 04, 45, 0, 99, 47, 43, 23, 143, 75, 6, 19, 79

   Will the runtimes differ based on the data-structure used to store the data? Consider the cases of an array or a single linked list as a data structure to store the data.

→ Solution :- Given to consider every 3rd element Pivot Element

i j

=>	24	29	18	8	27	9	21	89	04	45	0	99	47	43	23	143	75	6	19	79
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

→ By doing partitioning technique we get

$\boxed{18}$

8, 9, $\boxed{4}$, 0, 6 | 24, 29, $\boxed{27}$, 21, 89, 45, 99, 47, 43, 23, 143, 75, 19, 79

0 | 6, 8, 9 | 24, 21, $\boxed{23}$, 19 | 29, 89, $\boxed{45}$, 99, 47, 43, 143, 75, 79

21, 19 | 24 | 29, 43 | 89, 99, $\boxed{47}$, 143, 75, 79

19, 21 | | 89, 99, $\boxed{143}$, 75, 79

89, 99, $\boxed{75}$, 79 |

| 89, 99, $\boxed{79}$

89, 99

→ The traversal is

0, 4, 6, 8, 9, 18, 19, 21, 23, 24, 27, 29, 43, 45, 47, 75, 79, 89, 99, 143

→ Quick Sort is recursive algorithm

→ Runtime is $O(n \log n)$

→ Worst case is $O(n^2)$

→ The difference can be extra load on the memory allocation part and so on. The other side there can be not many difference

→ Every iteration has $n/2$ which results in

$\frac{n}{2^k} = 1$

$n = 2^k$

$k = \log_2 n$

$O(n \log n)$ — Best case

→ Algorithm

```
QuickSort (l, h)
{
  if (l < h)
  {
     j = partition (l, h);
     Quicksort (l, j);
     Quicksort (j+1, h);
  }
}
```

→ Depending on the data structure used to hold the data, the runtime will defer. We can conduct random access with array but not with linked lists. Quick sort necessitates a lot of random access but in a linked list, we must traverse every node from the head to the ith node to reach the index. As a result, quicksort has a higher overhead.

2. (25pts; Basic D&C) Show all the steps of Strassen's matrix multiplication algorithm to multiply the following two 4 X 4 matrices

$$X = \begin{bmatrix} 2 & 1 & 2 & 1 \\ 3 & 0 & 3 & 1 \\ 0 & 3 & 2 & 4 \\ 4 & 3 & 4 & 2 \end{bmatrix}, Y = \begin{bmatrix} 2 & 4 & 1 & 1 \\ 1 & 1 & 4 & 3 \\ 4 & 3 & 2 & 4 \\ 3 & 2 & 4 & 2 \end{bmatrix}.$$ You might find it easier to write a program for Strassen's matrix multiplication and then printing intermediate results.

→ By using 4 x 4

$$X = \left[\begin{array}{cc|cc} \multicolumn{2}{c|}{A_{11}} & \multicolumn{2}{c}{A_{12}} \\ 2 & 1 & 2 & 1 \\ 3 & 0 & 3 & 1 \\ \hline 0 & 3 & 2 & 4 \\ 4 & 3 & 4 & 2 \\ \multicolumn{2}{c|}{A_{21}} & \multicolumn{2}{c}{A_{22}} \end{array}\right] \qquad Y = \left[\begin{array}{cc|cc} \multicolumn{2}{c|}{B_{11}} & \multicolumn{2}{c}{B_{12}} \\ 2 & 4 & 1 & 1 \\ 1 & 1 & 4 & 3 \\ \hline 4 & 3 & 2 & 4 \\ 3 & 2 & 4 & 2 \\ \multicolumn{2}{c|}{B_{21}} & \multicolumn{2}{c}{B_{22}} \end{array}\right]$$

→ According to the Formula -

$P = (A_{11} + A_{22})(B_{11} + B_{22})$

$Q = (A_{21} + A_{22}) B_{11}$

$R = A_{11}(B_{12} - B_{22})$

$S = A_{22}(B_{21} - B_{11})$

$T = (A_{11} + A_{12}) B_{22}$

$U = (A_{21} - A_{11})(B_{11} + B_{12})$

$V = (A_{12} - A_{22})(B_{22} + B_{21})$

$C_{11} = P + S - T + V$

$C_{12} = R + T$

$C_{21} = Q + S$

$C_{22} = P + R - Q + U$

$A_{11} = \begin{bmatrix} 2 & 1 \\ 3 & 0 \end{bmatrix}$ $A_{12} = \begin{bmatrix} 2 & 1 \\ 3 & 1 \end{bmatrix}$ $A_{21} = \begin{bmatrix} 0 & 3 \\ 4 & 3 \end{bmatrix}$ $A_{22} = \begin{bmatrix} 2 & 4 \\ 4 & 2 \end{bmatrix}$

$B_{11} = \begin{bmatrix} 2 & 4 \\ 1 & 1 \end{bmatrix}$ $B_{12} = \begin{bmatrix} 1 & 1 \\ 4 & 3 \end{bmatrix}$ $B_{21} = \begin{bmatrix} 4 & 3 \\ 3 & 2 \end{bmatrix}$ $B_{22} = \begin{bmatrix} 2 & 4 \\ 4 & 2 \end{bmatrix}$

$$P = \begin{bmatrix} 4 & 5 \\ 7 & 2 \end{bmatrix} \begin{bmatrix} 4 & 8 \\ 5 & 3 \end{bmatrix} = \begin{bmatrix} 16+25 & 32+15 \\ 28+10 & 56+6 \end{bmatrix} = \begin{bmatrix} 41 & 47 \\ 38 & 62 \end{bmatrix}$$

$$Q = \begin{bmatrix} 2 & 4 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 7 \\ 8 & 5 \end{bmatrix} = \begin{bmatrix} 11 & 15 \\ 21 & 37 \end{bmatrix}$$

$$R = \begin{bmatrix} 2 & 1 \\ 3 & 0 \end{bmatrix} \begin{bmatrix} -1 & -3 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -2 & -5 \\ -3 & -9 \end{bmatrix}$$

$$S = \begin{bmatrix} 2 & 4 \\ 4 & 2 \end{bmatrix} \begin{bmatrix} 2 & -1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 12 & 2 \\ 12 & -2 \end{bmatrix}$$

$$T = \begin{bmatrix} 2 & 4 \\ 4 & 2 \end{bmatrix} \begin{bmatrix} 4 & 2 \\ 6 & 1 \end{bmatrix} = \begin{bmatrix} 16 & 20 \\ 16 & 26 \end{bmatrix}$$

$$U = \begin{bmatrix} -2 & 2 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 3 & 5 \\ 5 & 4 \end{bmatrix} = \begin{bmatrix} 4 & -2 \\ 18 & 17 \end{bmatrix}$$

$$V = \begin{bmatrix} 0 & -3 \\ -1 & -1 \end{bmatrix} \begin{bmatrix} 6 & 7 \\ 7 & 4 \end{bmatrix} = \begin{bmatrix} -21 & -12 \\ -13 & -11 \end{bmatrix}$$

→ Substituting values of P, Q, R, S, T, U, V in C_{11}, C_{12}, C_{21}, C_{22}.

$$C_{11} - \begin{bmatrix} 41 & 47 \\ 38 & 62 \end{bmatrix} + \begin{bmatrix} 12 & 2 \\ 12 & -2 \end{bmatrix} - \begin{bmatrix} 16 & 20 \\ 16 & 26 \end{bmatrix} + \begin{bmatrix} -21 & -12 \\ -13 & -11 \end{bmatrix}$$

$$= \begin{bmatrix} 16 & 17 \\ 21 & 23 \end{bmatrix}$$

$$C_{12} - \begin{bmatrix} -2 & -5 \\ -3 & -9 \end{bmatrix} + \begin{bmatrix} 16 & 20 \\ 16 & 26 \end{bmatrix} = \begin{bmatrix} 14 & 15 \\ 13 & 17 \end{bmatrix}$$

$$C_{21} - \begin{bmatrix} 11 & 15 \\ 21 & 37 \end{bmatrix} + \begin{bmatrix} 12 & 2 \\ 12 & -2 \end{bmatrix} = \begin{bmatrix} 23 & 17 \\ 33 & 35 \end{bmatrix}$$

$$C_{22} - \begin{bmatrix} 41 & 47 \\ 38 & 62 \end{bmatrix} + \begin{bmatrix} -2 & -5 \\ -3 & -9 \end{bmatrix} - \begin{bmatrix} 11 & 15 \\ 21 & 37 \end{bmatrix} + \begin{bmatrix} 4 & -2 \\ 18 & 17 \end{bmatrix}$$

$$= \begin{bmatrix} 32 & 25 \\ 32 & 33 \end{bmatrix}$$

(F)

$$\begin{bmatrix} 16 & 17 & 14 & 15 \\ 21 & 23 & 13 & 17 \\ 23 & 17 & 32 & 25 \\ 33 & 35 & 32 & 33 \end{bmatrix} = Z$$

$$T(n) = \begin{cases} 1 & n \leq 2 \\ 7T(n/2) + n^2 & n > 2 \end{cases}$$

$\log_2^7 = 2.81 \qquad k = 2$

$O(n^{\log_2^7}) = O(n^{2.81})$

- Pseudocode :-

1. By dividing X and Y in 4 sub-matrices of size $n/2 \times n/2$ each.
2. Evaluate the 7 matrix multiplications recursively.
3. Compute the submatricies to Z
4. Combining them we get matrix Z

3. (25pts; hw/sw co-design)

Recall the naïve divide and conquer O(n^2)-time multiplication algorithm, call it NaiveD&Cmult, to multiply two n-digit numbers. Design a 8-bit multiplier using NaiveD&Cmult algorithm with basic building blocks of 1-bit multiplier and m-bit adders and shifters, m>0. Note that in class we did not consider the cases when n is not a power of two, you may have to modify the algorithm to take care of these cases. Also, we discussed algorithm using decimal digits, obviously it can be easily extended to any radix, so for this problem radix is 2.

→ Implementing Algorithm of karatsuba in the 8-bit multiplier by the following :-

Algorithm karatsuba (num0, num1)
if (num0 < 10), (num1 < 10)
then return num0 * num1
/* we calculate the size of the answer and check if it has a carry for the following radix 2 answer */.
S = max (radix 2 (num 0), radix 2 (num 1))
D = D/2
/* Dividing the sequence from the middle to find the half of each */
high0, low0 = divide (num0, D)
high1, low1 = divide (num 1, D)
/* karatsuba algorithm used next */
x1 = karatsuba (low0, low1)
x2 = karatsuba (low0 + high0), (low1 + high1)
x3 = karatsuba (high0, high1)
return (X3 * 10^(2 * D)) + ((X3 - X2 - X1) * 10^(D)) + X1

ther Explained Algorithm for every component :- (7)

$x \cdot y = 10^{n} ac + 10^{n/2}(ad + bc) + bd$ [Given Formula from Professor]

$= (A_L \cdot 10^{n/2} + A_R)(B_L \cdot 10^{n/2} + B_R)$

$= A_L \times B_L \times 10^{n} + (A_L B_R + A_R B_L) 10^{n/2} + A_R B_R$

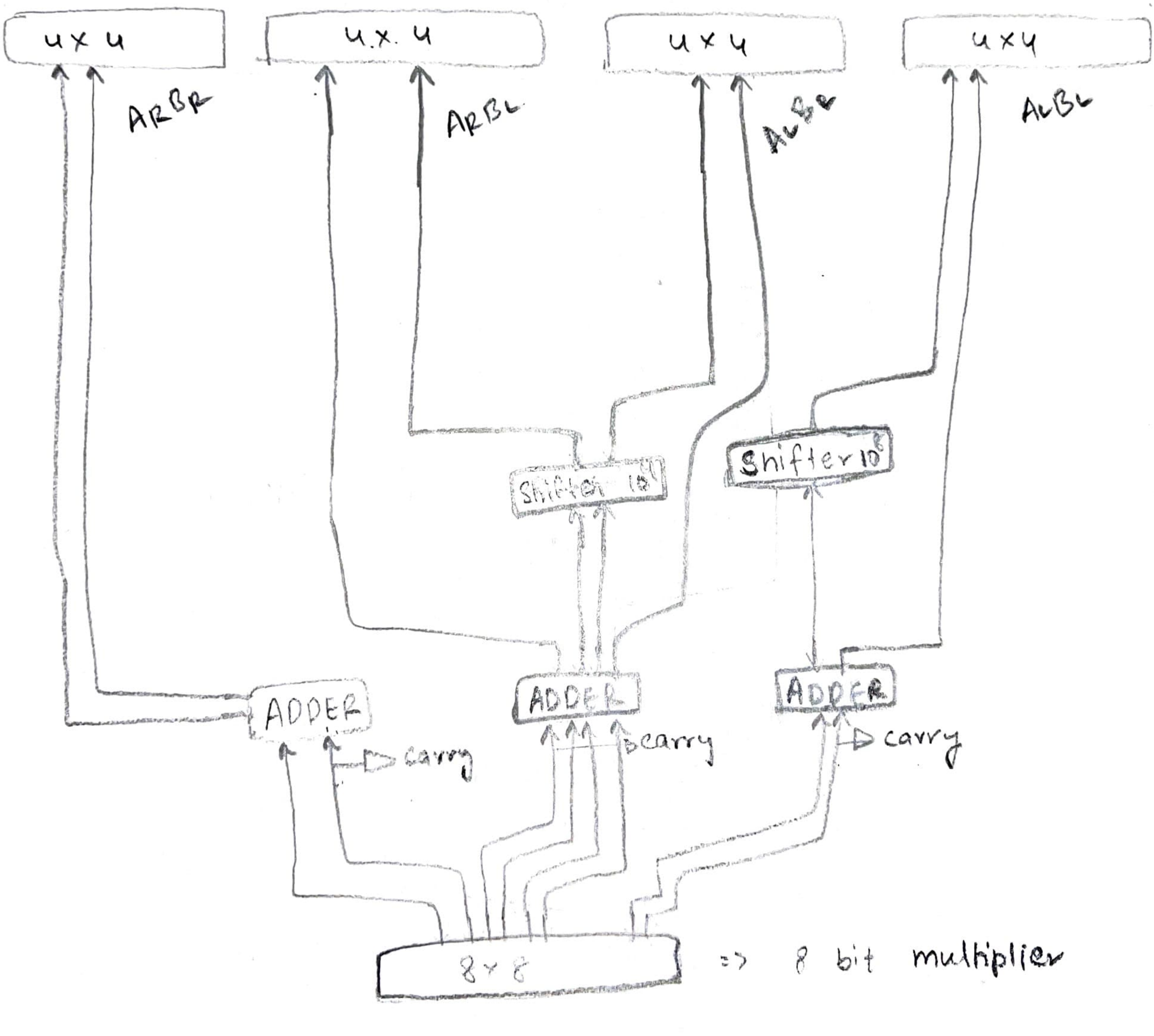
→ for 8-bit multiplier

(1) $A_L B_L \times 10^{8} + (A_L B_R + A_R B_L) 10^{8/2} + A_R B_R$ [n/2 D&N for each shift/Adder]

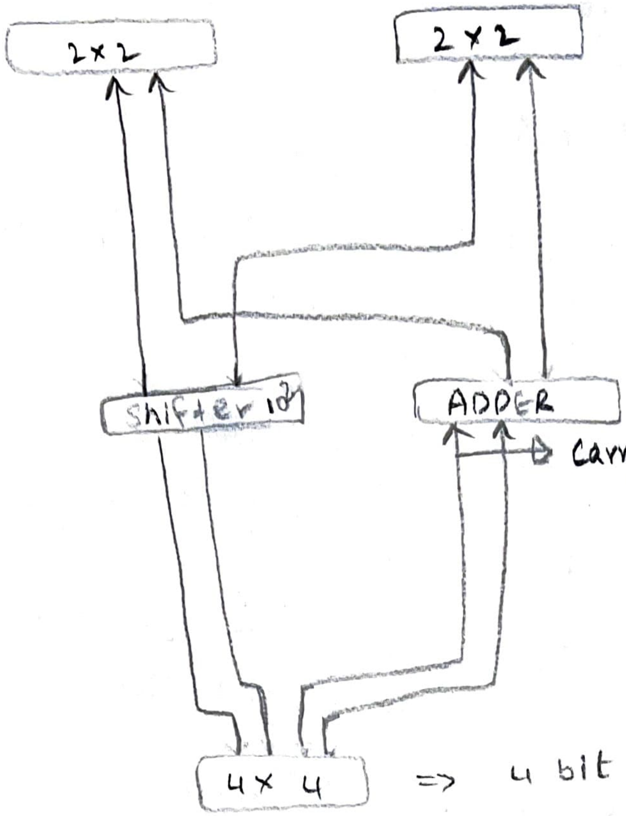
(2) $A_L B_L \times 10^{4} + (A_L B_R + A_R B_L) 10^{4/2} + A_R B_R$ [n/4 " "]

(3) $A_L B_L \times 10^{2} + (A_L B_R + A_R B_L) 10^{2/2} + A_R B_R$ [n/8 " "]

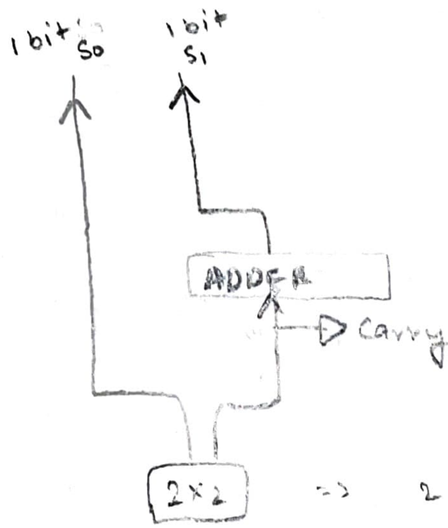
→ Schematic Architecture of 8-bit Multiplier



=> 8 bit multiplier



⟹ 4 bit multiplier



⟹ 2 bit multiplier

∴ We see that the algorithm show's how it first divides it from 8-bit to 4 bit then to 2 bit & at last 1-bit multiplier

→ So the time complexity reduces the multiplication of 2 n digits to $n^{\log_2 3} = O(n^{1.58})$.

8

(13)

9

4. (25pts; Recurrence relations and basic D&C) Consider the following extension to Binary Search a sorted list A for an item x: Let's call it UnevenBinSearch. Let *ap* be the partitioning element of A so that A is divided into two lists, left sublist is close to length [n/3] and the right sublist is of length n-[n/3]~[2n/3] (i.e., a difference of 1 or 2 is allowed between the sizes of these sublists but no more). Based on *ap*, decide the sublist where x may exist. Repeat this process until the sublist is small enough that the answer can be directly obtained.
    1. 4.1. Write pseudo-code for the above UnevenBinSearch method, specifying all the parameters correctly.
    2. 4.2. Derive the recurrence relation for the running time of the UnevenBinSearch algorithm.
    3. 4.3. Solve the recurrence relation, find a close-bound for the running time of UnevenBinSearch and then express this close-bound using asymptotic notation. Justify your answer.
    4. 4.4. Derive an expression and its asymptotic bound for the space complexity of UnevenBinSearch.
    5. 4.5. Repeat steps 4.1-4.4 if the partitioning element *ap* happens to divide the list so that the left sublist is of size [2n/3] and the right sublist is of length n-[2n/3].

1) Pseudo code

Algorithm UnevenBinSearch

let start = 0 & finish = n-1

Evalute as (start + finish) / 3 , reduce down value obtained.

Array Bin = (start + finish) / 3

If array [Bin] equals objective, then stop and return

If array [Bin] did not achieve, set finish = Bin - 1

then start from evaluate again

If value is not found return -1

∴ The list here is divided as (n/3) left sublist and $(\frac{2n}{3})$ right sublist

Algorithm


```
int UnevenBinSearch(int arr[], int start, int finish, int objective)
{
    if (start <= finish) {
        int Bin = start + ((finish - start)/3);
        if (arr[Bin] == Objective)
            return Bin.
        else if (arr[Bin] < objective)
            return UnevenBinSearch(arr, Bin+1, finish, objective);
        else
            return UnevenBinSearch(arr, start, Bin-1, objective);
    }
    return -1;
}
```

4.2] Recurrance Relation for the Uneven Bin Search

$T(n) = T\left(\frac{2n}{3}\right) + 1$ (for any constant 1)

Step - 1

$$T(n) = \left[\left(\frac{2^2 n}{3^2}\right) + 1\right] + 1$$

Step - 2

$$T(n) = \left[\frac{2^3 n}{3^3} + 1\right] + 1 + 1$$

Step - 3

$$T(k) = \left[\frac{2^k n}{3^k} + k\right]$$

This is the stopping Condition.

(d)

Assume that $\left(\frac{2}{3}\right)^k n = 1$ [by iteration Method]

$\rightarrow \left(2/3\right)^k = 1/n \rightarrow$ (1)

$\rightarrow \log_{2/3}\left(2/3\right)^k = \log_{2/3}\left(1/n\right)$

$\rightarrow k \log_{2/3}\left(2/3\right) = \log_{2/3}\left(1/n\right)$

$\rightarrow k\,(1) = \log_{2/3}\left(1/n\right)$

$\rightarrow k = \log 1/n \rightarrow$ (2)

$T(n) = \left[T\left(\left(2/3\right)^k n\right)\right] + k$ from (1) & (2)

$= \left[T(1)\right] + \log\left(1/n\right)$

$= \log 1/n + T(1)$

$= \log 1/n + 1$

$= \Theta(\log n)$

$\therefore$ So, $T(n) = T(2n/3) + 1 = \Theta(\log n)$

4.3] Taking Big Omega equation

$$f(n) \geq c \cdot g(n)$$

$$\log n \geq 1 \times 2^{10}$$

Taking Big oh Equation

$$f(n) \leq c \cdot n$$

$$\log n \leq \sqrt{n}$$

→ If we take the height of the tree, it will be $2^{\log(n)}$ where the full work done by the left side and adding that gives us.

$$T(n) \geq T(n_1) = 2^{\log(n)} + \log_3(n)$$

$$= n^{1.6} + \log_3(n)$$

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

4.4] The space complexity of the algorithm is $O(n)$ as it is recursively using the search operation in the given list.

$$f(n) \leq c \cdot g(n)$$ Big O

$$n \leq n \log n$$

$$f(n) \geq c \cdot g(n)$$ Big omega

$$n > \sqrt{n}$$

$$c_2\, g(n) \leq f(n) \leq c_1\, g(n)$$ Big theta

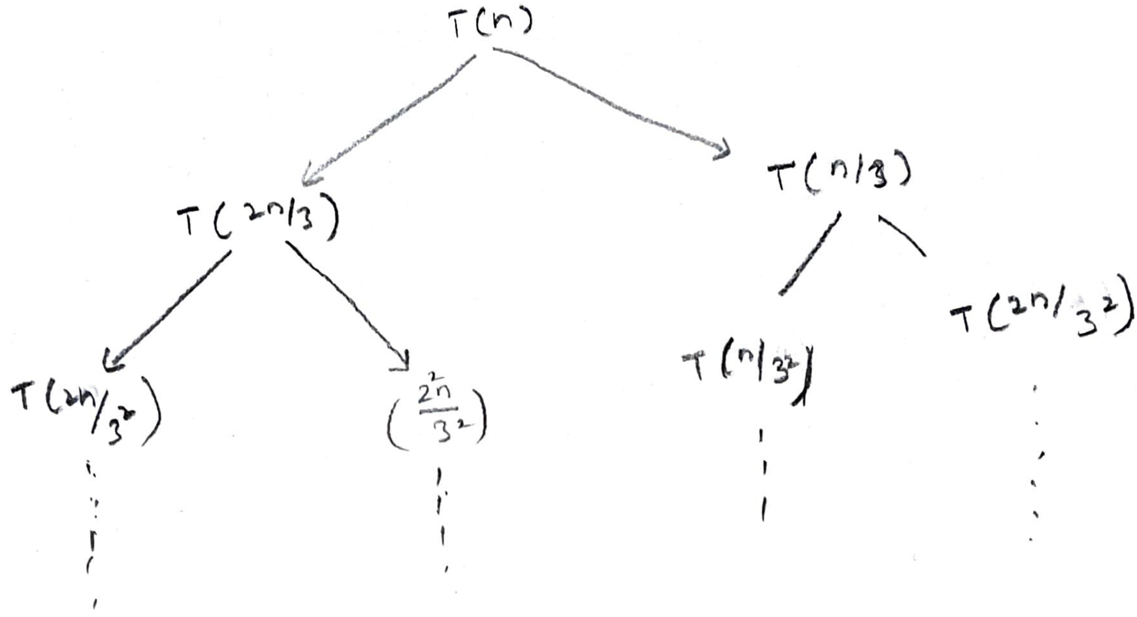
$$\sqrt{n} \leq n \leq n \log n$$

[4.5] The partition element ap happens to divide the list T (2n/3) times and the right sublist n - [2n/3] times

$\Rightarrow n - \frac{2n}{3}$

$\Rightarrow \frac{3n - 2n}{3} = \frac{n}{3}$

→ Then the recurrence tree looks like



→ According the interchange of the sublists from left to right doesn't make any chance.
so, this give's us log(n) as the time complexity from 4.1 - 4.4.

→ The space complexity will be O(n)

→ The close bond will be O(nlogn)

→ Recurrence relation of the list will be same as 4.2 which gives O(logn)

Discussed solutions with

1. Yashoda
2. Sahar

I give permission to the instructor to share my solution.

PLAGIARISM DECLARATION

1. I know that plagiarism means taking and using the ideas, writings, programs, code, works, or inventions of another as if they were one's own. I know that plagiarism not only includes verbatim copying, but also the extensive use of another person's ideas without proper acknowledgement (which includes the proper use of quotation marks). I know that plagiarism covers this sort of use of material found in textual sources and from the Internet.

2. I acknowledge and understand that plagiarism is wrong.

3. This assignment is my own work, or my group's own unique group assignment. I acknowledge that copying someone else's assignment, or part of it, is wrong, and that submitting identical work to others constitutes a form of plagiarism.

4. I have not allowed, nor will I in the future allow, anyone to copy my work with the intention of passing it off as their own work.

Name: AFFAN MOYEED Student #: 08560172?

Signed .... Date: 02/18/22