MGM’s College of Engineering and Technology Kamothe, Navi Mumbai

Department of Computer Engineering

Assignment-2

Course Code: CSC402 **Course Name: Analysis of Algorithms**
Class: SE/IV **AY: 2021-2022**
Date of Issue: 24/02/2022

Q.No	**Question**
Q1. Fill in the blanks	____________________
a)	Steps of Divide and Conquer approach Divide, Recur and Conquer
b)	The complexity of searching an element from a set of n elements using Binary search algorithm is O ($\log_2 n$)
c)	The running time of quick sort depends on the selection of how balanced the partitions are.
Q2. Choose Correct Options	
a)	What is the worst-case time complexity of a quick sort algorithm? a) O(N) b) O (N log N) [✓ c) O(N2)] d) O (log N)
b)	Which is the safest method to choose a pivot element? a) choosing a random element as pivot b) choosing the first element as pivot c) choosing the last element as pivot [✓ d) median-of-threepartitioningmethod]
c)	What is the worst-case time complexity of merge sort? [✓ a) O(nlogn)] b) O(n2) c) O (n2 log n) d) O (n log n2)

d)	Which of the following method is used for sorting in merge sort? a) merging b) partitioning c) selection d) exchanging

e) Which of the following stable sorting algorithm takes the least time when applied to an almost sorted array?

a) Quick sort

b) Insertion sort

c) Selection sort

d) Merge sort

Q3. State whether the following statements are true or false (Give Reasons)

a) Quick sort follows Divide-and-Conquer strategy. True

b) Merge sort is preferred for arrays over linked lists. True

c) Merge sort can be implemented using O (1) auxiliary space. False

Q4. Name the following or define or design the following

a) What is Divide-and-conquer?

Ans: Divide and conquer is a method of algorithm design that has created such efficient algorithms as Merge Sort.

• In terms or algorithms, this method has three distinct steps: –

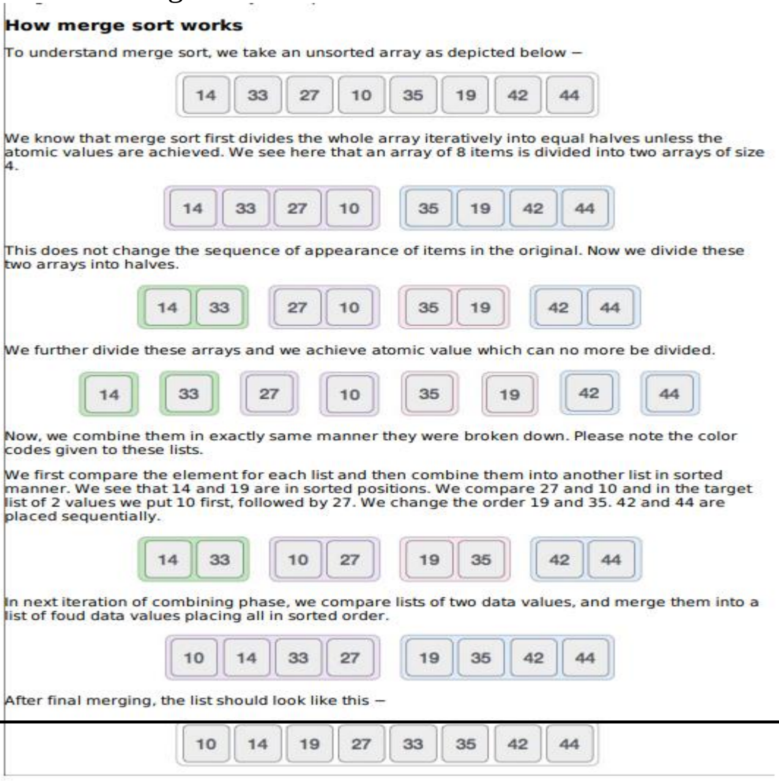
• Divide: If the input size is too large to deal with in a straightforward manner, divide the data into two or more disjoint subsets.

• Recur: Use divide and conquer to solve the subproblems associated with the data subsets.

• Conquer: Take the solutions to the subproblems and “merge” these solutions into a solution for the original

b) How does Merge Sort work?

Ans:



How merge sort works

To understand merge sort, we take an unsorted array as depicted below –

We know that merge sort first divides the whole array iteratively into equal halves unless the atomic values are achieved. We see here that an array of 8 items is divided into two arrays of size 4.

This does not change the sequence of appearance of items in the original. Now we divide these two arrays into halves.

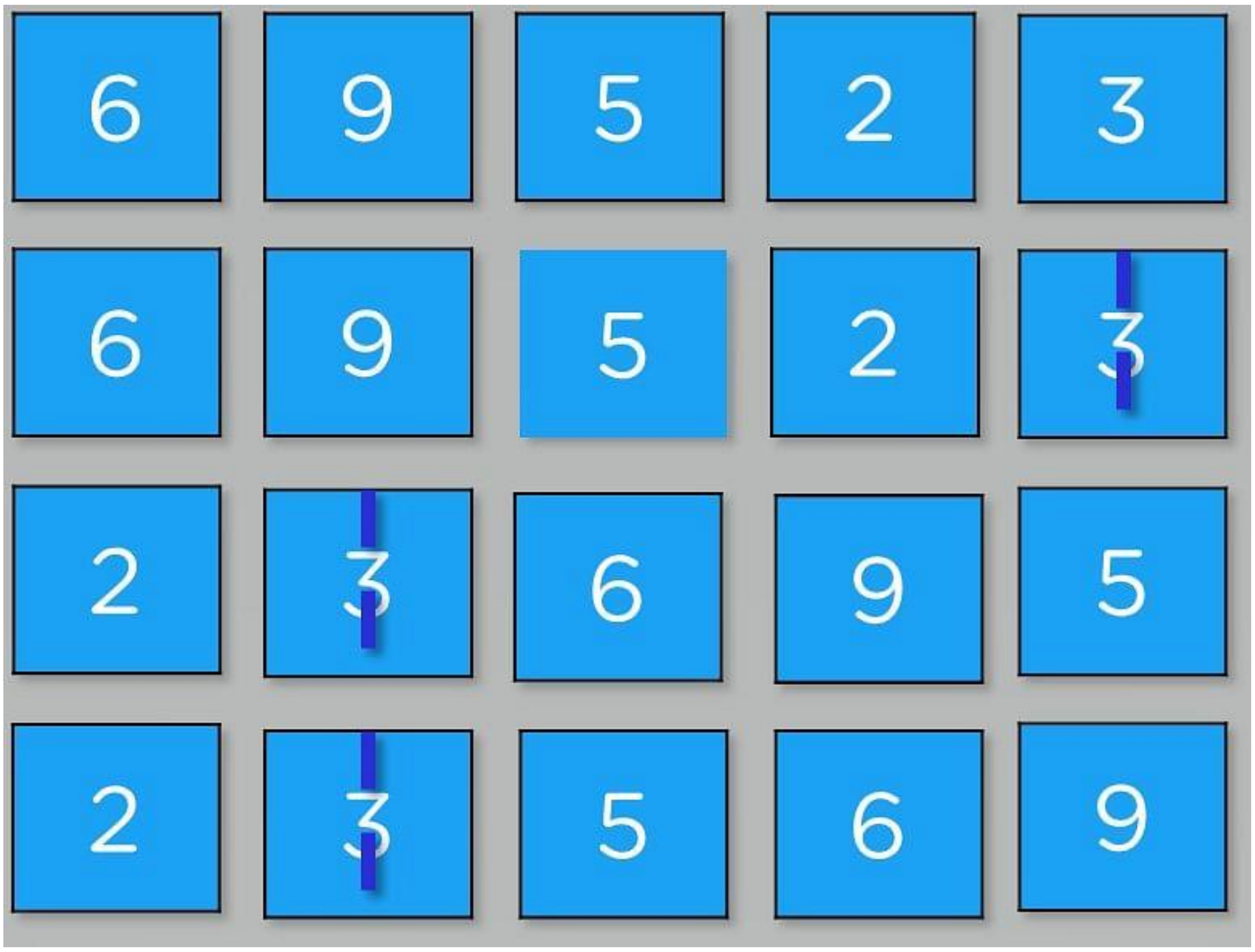
We further divide these arrays and we achieve atomic value which can no more be divided.

Now, we combine them in exactly same manner they were broken down. Please note the color codes given to these lists.

We first compare the element for each list and then combine them into another list in sorted manner. We see that 14 and 19 are in sorted positions. We compare 27 and 10 and in the target list of 2 values we put 10 first, followed by 27. We change the order 19 and 35. 42 and 44 are placed sequentially.

In next iteration of combining phase, we compare lists of two data values, and merge them into a list of foud data values placing all in sorted order.

After final merging, the list should look like this –

c) Ans:	How does Quick Sort work? To sort an array, you will follow the steps below: 1. You will make any index value in the array as a pivot. 2. Then you will partition the array according to the pivot. 3. Then you will recursively quicksort the left partition 4. After that, you will recursively quicksort the correct partition.  1. You will pick any pivot, let's say the highest index value. 2. You will take two variables to point left and right of the list, excluding pivot. 3. The left will point to the lower index, and the right will point to the higher index. 4. Now you will move all elements which are greater than pivot to the right. 5. Then you will move all elements smaller than the pivot to the left partition.

Q5. Answer the following questions in brief (20 to 30 words)

a) Ans:

Explain Minimum and Maximum algorithm.

To find the maximum and minimum numbers, the following straightforward algorithm can be used. Algorithm: **Max-Min-Element** (numbers []) max: = numbers [1] min: = numbers [1] for i = 2 to n do if numbers [i] > max then max: = numbers [i] if numbers [i] < min then min: = numbers [i] return (max, min)

b) Ans:

Explain Analysis of Binary search algorithm.

• Problem Statement: Binary search can be performed on a sorted array. In this approach, the index of an element x is determined if the element belongs to the list of elements. If the array is unsorted, linear search is used to determine the position.

• Solution: In this algorithm, we want to find whether element x belongs to a set of numbers stored in an array numbers []. Where l and r represent the left and right index of a sub-array in which searching operation should be performed.

• Algorithm: Binary-Search(numbers[], x, l, r)

```
if l = r then
return l
else
m: = ⌊(l + r) / 2⌋
if x ≤ numbers[m] then
return Binary-Search(numbers[], x, l, m)
else
return Binary-Search(numbers[], x, m+1, r)
```

c) Explain Binary search algorithm.

Ans:

Given an array of elements with values or records sorted such that , and target value , the following subroutine uses binary search to find the index of in .[7]

1. Set to and to .
2. If , the search terminates as unsuccessful.
3. Set (the position of the middle element) to the floor of , which is the greatest integer less than or equal to .
4. If , set to and go to step 2.
5. If , set to and go to step 2.
6. Now , the search is done; return .

This iterative procedure keeps track of the search boundaries with the two variables and . The procedure may be expressed in pseudocode as follows, where the variable names and types remain the same as above, `floor` is the floor function, and `unsuccessful` refers to a specific value that conveys the failure of the search.[7]

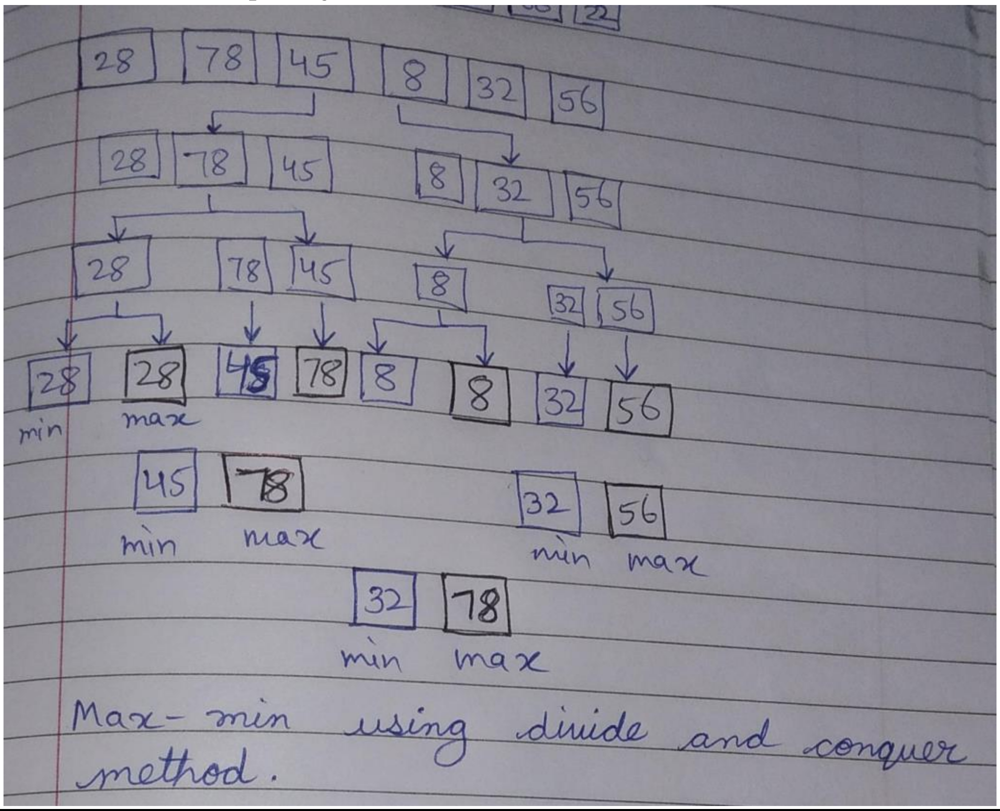
```
function binary search(A, n, T) is
    L := 0
    R := n − 1
    while L ≤ R do
        m := floor((L + R) / 2)
        if A[m] < T then
            L := m + 1
        else if A[m] > T then
            R := m − 1
        else:
            return m
    return unsuccessful
```

Alternatively, the algorithm may take the ceiling of . This may change the result if the target value appears more than once in the array.

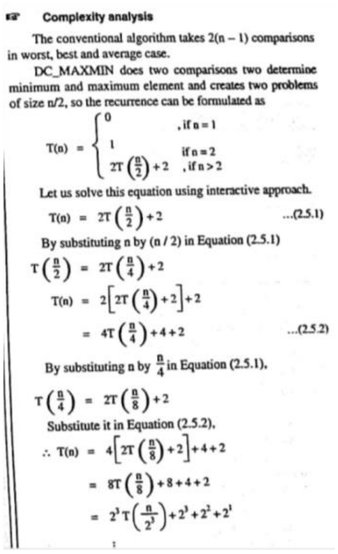
Q6. Answer the following questions in brief (50 to 70 words)

a) Apply Minimum and Maximum algorithm the following numbers. Show each step clearly. 28, 78, 45, 8, 32, 56. Derive the complexity

Ans:



Max-min using divide and conquer method.



☞ **Complexity analysis**

The conventional algorithm takes $2(n-1)$ comparisons in worst, best and average case.

DC_MAXMIN does two comparisons two determine minimum and maximum element and creates two problems of size n/2, so the recurrence can be formulated as

$$T(n) = \begin{cases} 0 & \text{, if } n = 1 \\ 1 & \text{if } n = 2 \\ 2T\left(\frac{n}{2}\right) + 2 & \text{, if } n > 2 \end{cases}$$

Let us solve this equation using interactive approach.

$$T(n) = 2T\left(\frac{n}{2}\right) + 2 \qquad \text{...(2.5.1)}$$

By substituting n by (n / 2) in Equation (2.5.1)

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + 2$$

$$T(n) = 2\left[2T\left(\frac{n}{4}\right) + 2\right] + 2$$

$$= 4T\left(\frac{n}{4}\right) + 4 + 2 \qquad \text{...(2.5.2)}$$

By substituting n by $\frac{n}{4}$ in Equation (2.5.1),

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + 2$$

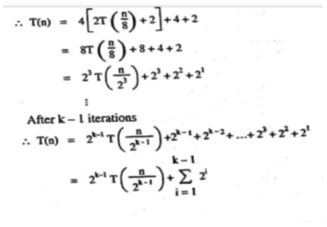
Substitute it in Equation (2.5.2),

$$\therefore T(n) = 4\left[2T\left(\frac{n}{8}\right) + 2\right] + 4 + 2$$

$$= 8T\left(\frac{n}{8}\right) + 8 + 4 + 2$$

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

$$\vdots$$



$$\therefore T(n) = 4\left[2T\left(\frac{n}{8}\right) + 2\right] + 4 + 2$$

$$= 8T\left(\frac{n}{8}\right) + 8 + 4 + 2$$

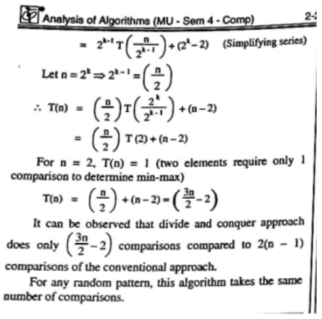
$$= 2^3 T\left(\frac{n}{2^3}\right) + 2^3 + 2^2 + 2^1$$

$$\vdots$$

After k – 1 iterations

$$\therefore T(n) = 2^{k-1} T\left(\frac{n}{2^{k-1}}\right) + 2^{k-1} + 2^{k-2} + \ldots + 2^3 + 2^2 + 2^1$$

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



$$= 2^{k-1} T\left(\frac{n}{2^{k-1}}\right) + (2^k - 2) \qquad \text{(Simplifying series)}$$

Let $n = 2^k \Rightarrow 2^{k-1} = \left(\frac{n}{2}\right)$

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

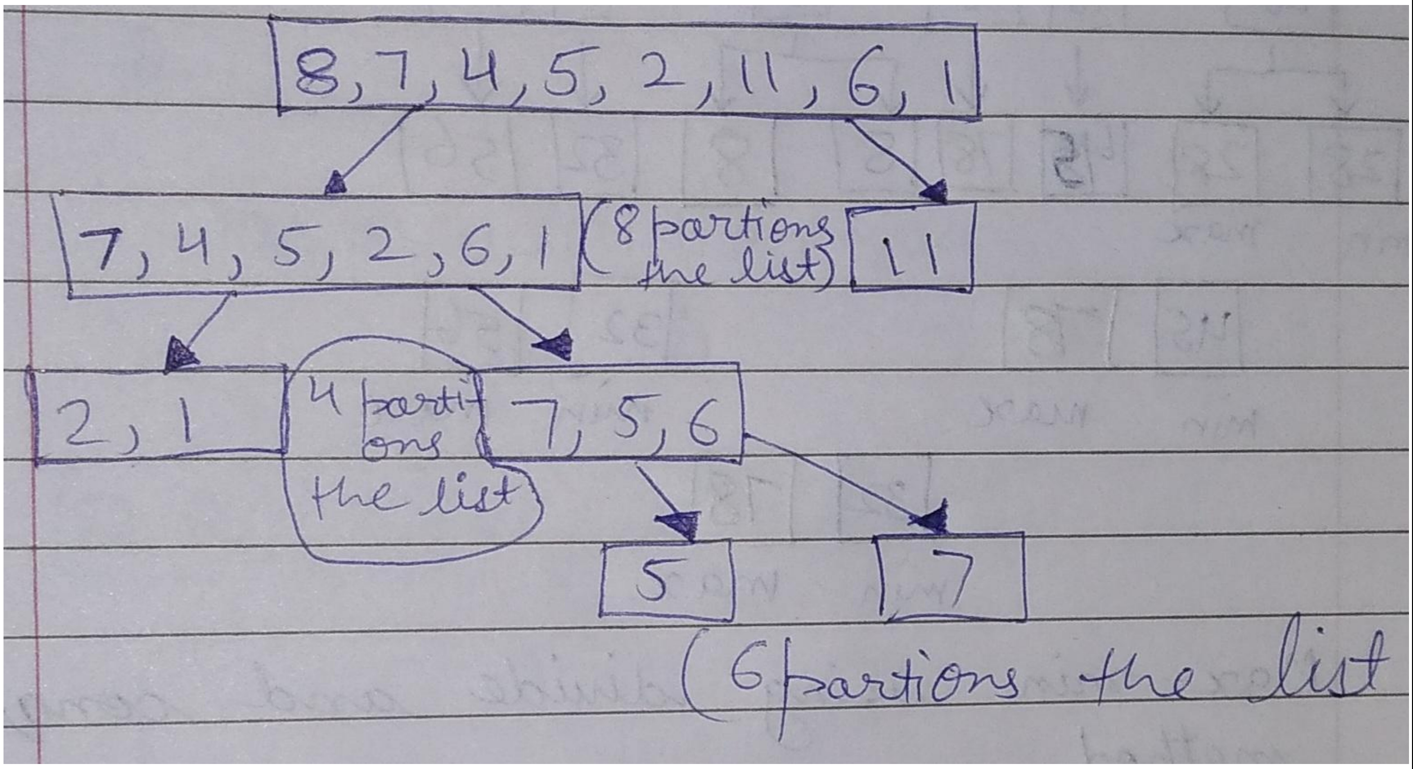
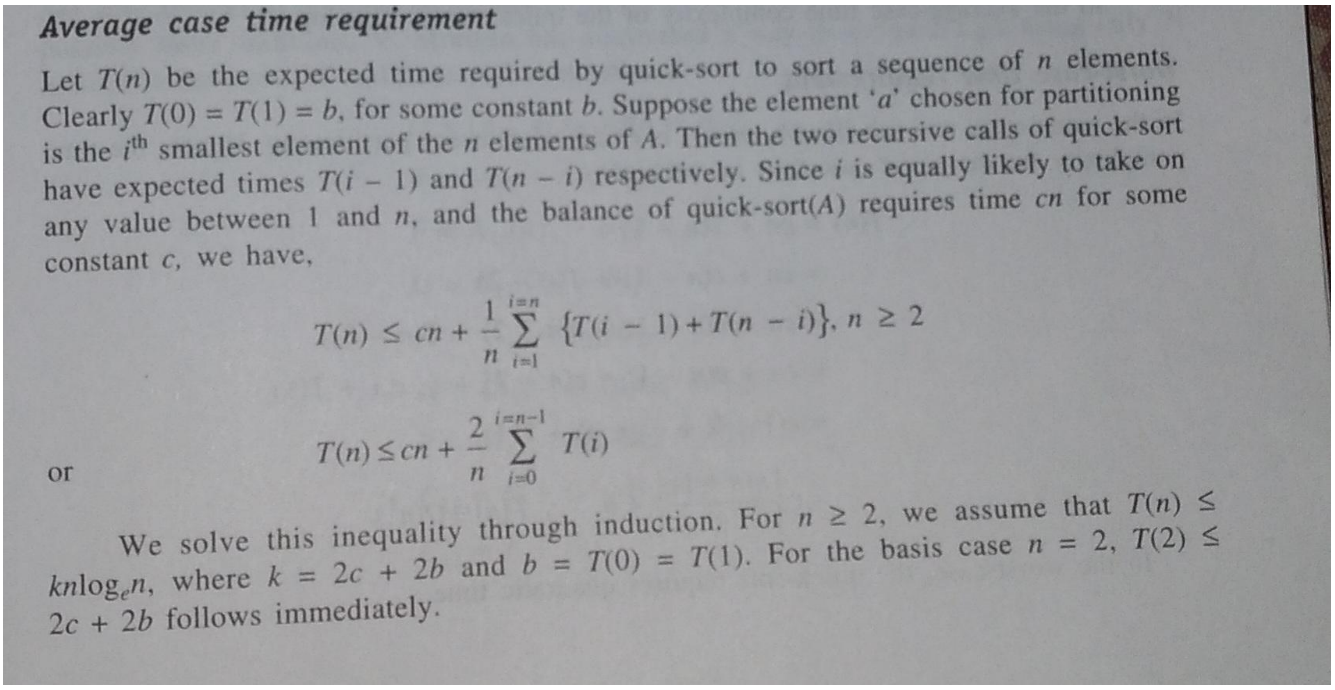
$$= \left(\frac{n}{2}\right) T(2) + (n-2)$$

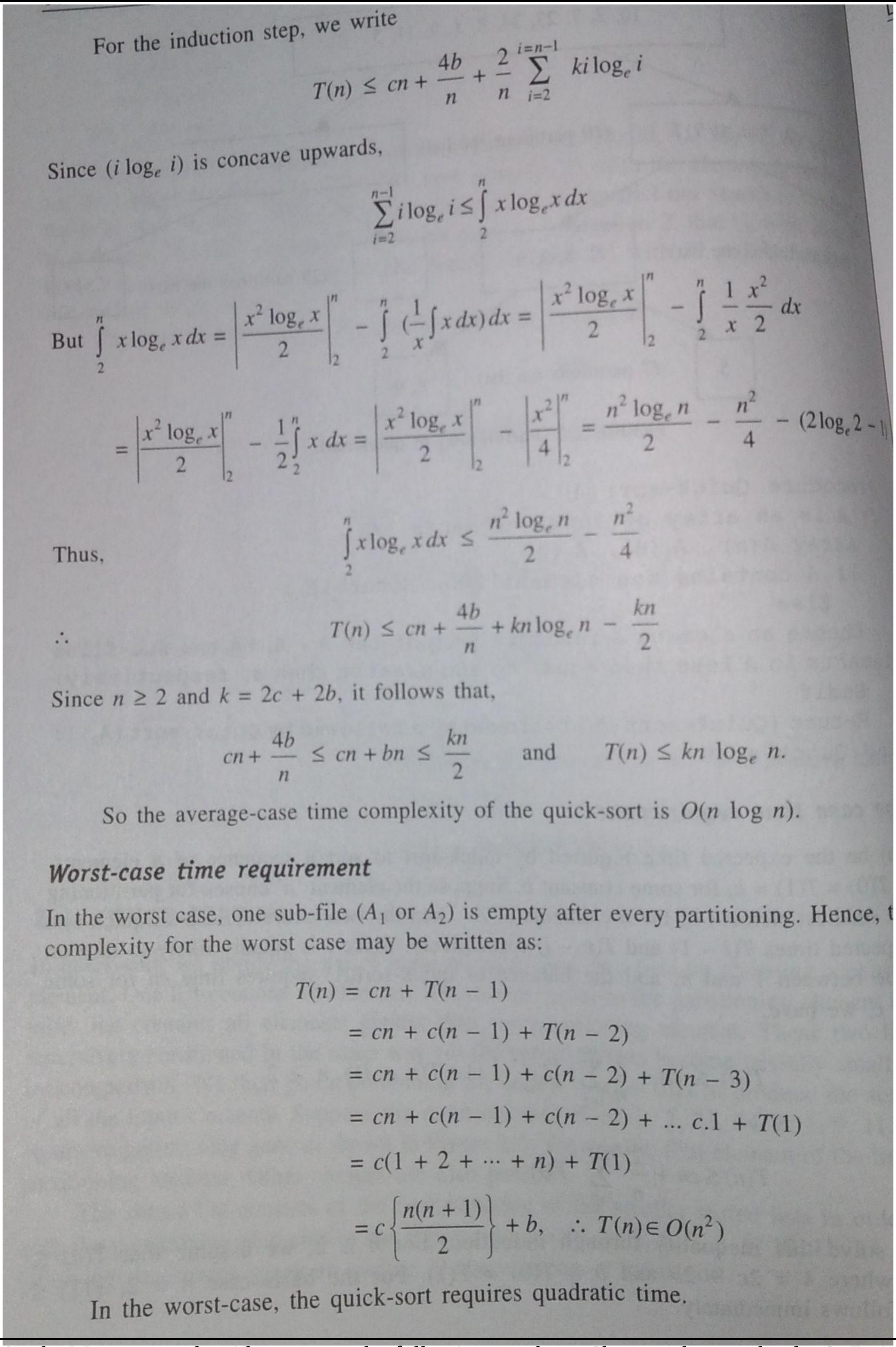
For $n = 2$, $T(n) = 1$ (two elements require only 1 comparison to determine min-max)

$$T(n) = \left(\frac{n}{2}\right) + (n-2) = \left(\frac{3n}{2} - 2\right)$$

It can be observed that divide and conquer approach does only $\left(\frac{3n}{2} - 2\right)$ comparisons compared to $2(n-1)$ comparisons of the conventional approach.

For any random pattern, this algorithm takes the same number of comparisons.

b)	Apply quick sort algorithm to sort the following numbers. Show each step clearly. 8, 7, 4, 5, 2, 11, 6, 1. Derive the complexity of Quick sort.
Ans:	 8, 7, 4, 5, 2, 11, 6, 1 7, 4, 5, 2, 6, 1 (8 partions the list) 11 2, 1 4 partitions the list 7, 5, 6 5 7 (6 partions the list **Average case time requirement** Let $T(n)$ be the expected time required by quick-sort to sort a sequence of n elements. Clearly $T(0) = T(1) = b$, for some constant b. Suppose the element 'a' chosen for partitioning is the i^{th} smallest element of the n elements of A. Then the two recursive calls of quick-sort have expected times $T(i-1)$ and $T(n-i)$ respectively. Since i is equally likely to take on any value between 1 and n, and the balance of quick-sort(A) requires time cn for some constant c, we have, $T(n) \le cn + \frac{1}{n}\sum_{i=1}^{i=n} \{T(i-1) + T(n-i)\},\ n \ge 2$ or $T(n) \le cn + \frac{2}{n}\sum_{i=0}^{i=n-1} T(i)$ We solve this inequality through induction. For $n \ge 2$, we assume that $T(n) \le kn\log_e n$, where $k = 2c + 2b$ and $b = T(0) = T(1)$. For the basis case $n = 2$, $T(2) \le 2c + 2b$ follows immediately.



For the induction step, we write

$$T(n) \le cn + \frac{4b}{n} + \frac{2}{n}\sum_{i=2}^{i=n-1} ki\log_e i$$

Since $(i \log_e i)$ is concave upwards,

$$\sum_{i=2}^{n-1} i\log_e i \le \int_2^n x\log_e x\,dx$$

But $\displaystyle\int_2^n x\log_e x\,dx = \left|\frac{x^2\log_e x}{2}\right|_2^n - \int_2^n \left(\frac{1}{x}\int x\,dx\right)dx = \left|\frac{x^2\log_e x}{2}\right|_2^n - \int_2^n \frac{1}{x}\,\frac{x^2}{2}\,dx$

$$= \left|\frac{x^2\log_e x}{2}\right|_2^n - \frac{1}{2}\int_2^n x\,dx = \left|\frac{x^2\log_e x}{2}\right|_2^n - \left|\frac{x^2}{4}\right|_2^n = \frac{n^2\log_e n}{2} - \frac{n^2}{4} - (2\log_e 2 - 1$$

Thus,
$$\int_2^n x\log_e x\,dx \le \frac{n^2\log_e n}{2} - \frac{n^2}{4}$$

$\therefore$
$$T(n) \le cn + \frac{4b}{n} + kn\log_e n - \frac{kn}{2}$$

Since $n \ge 2$ and $k = 2c + 2b$, it follows that,

$$cn + \frac{4b}{n} \le cn + bn \le \frac{kn}{2} \qquad \text{and} \qquad T(n) \le kn\ \log_e n.$$

So the average-case time complexity of the quick-sort is $O(n \log n)$.

Worst-case time requirement

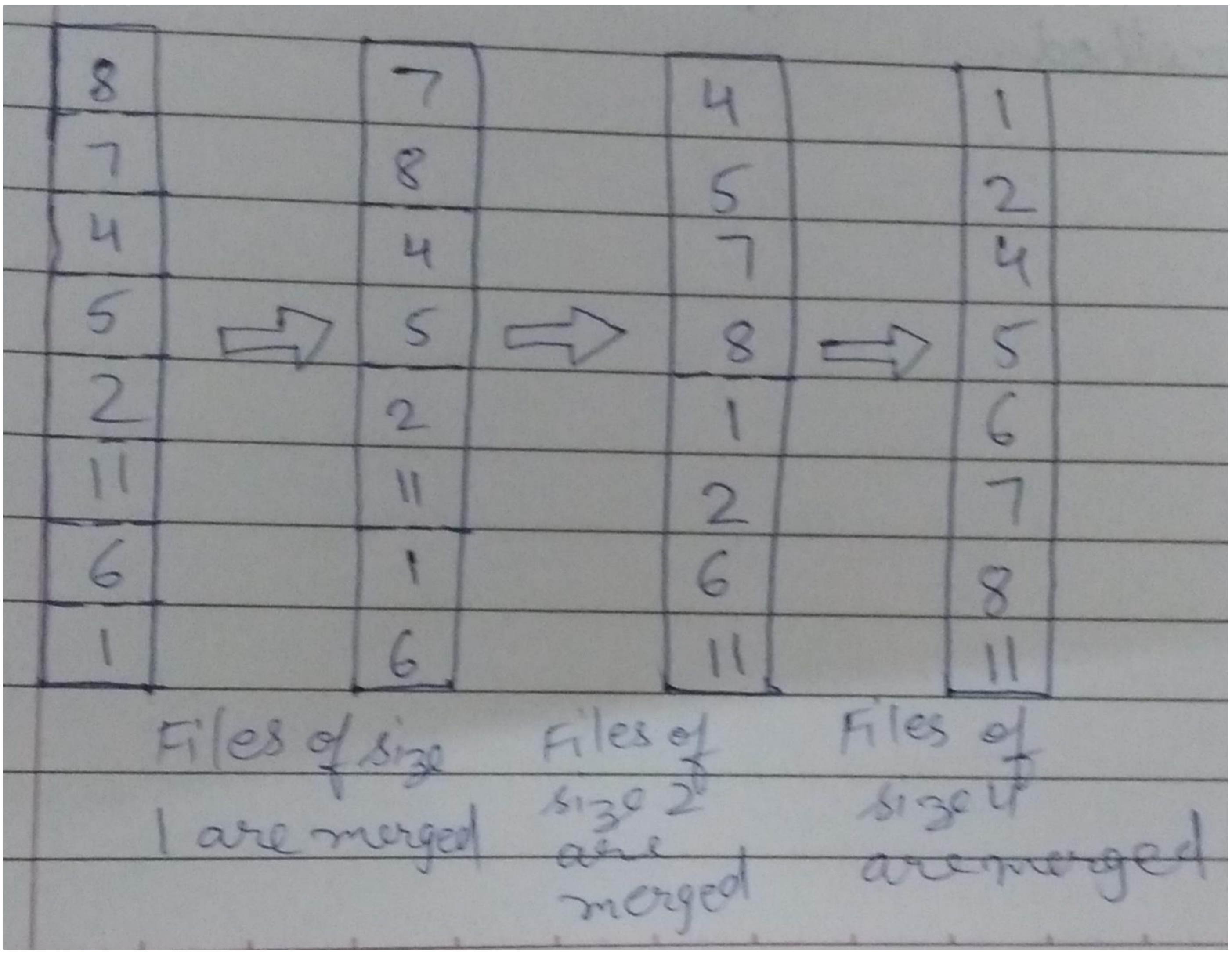
In the worst case, one sub-file (A_1 or A_2) is empty after every partitioning. Hence, t complexity for the worst case may be written as:

$$\begin{aligned}
T(n) &= cn + T(n-1)\\
&= cn + c(n-1) + T(n-2)\\
&= cn + c(n-1) + c(n-2) + T(n-3)\\
&= cn + c(n-1) + c(n-2) + \dots c.1 + T(1)\\
&= c(1 + 2 + \cdots + n) + T(1)\\
&= c\left\{\frac{n(n+1)}{2}\right\} + b, \quad \therefore\ T(n) \in O(n^2)
\end{aligned}$$

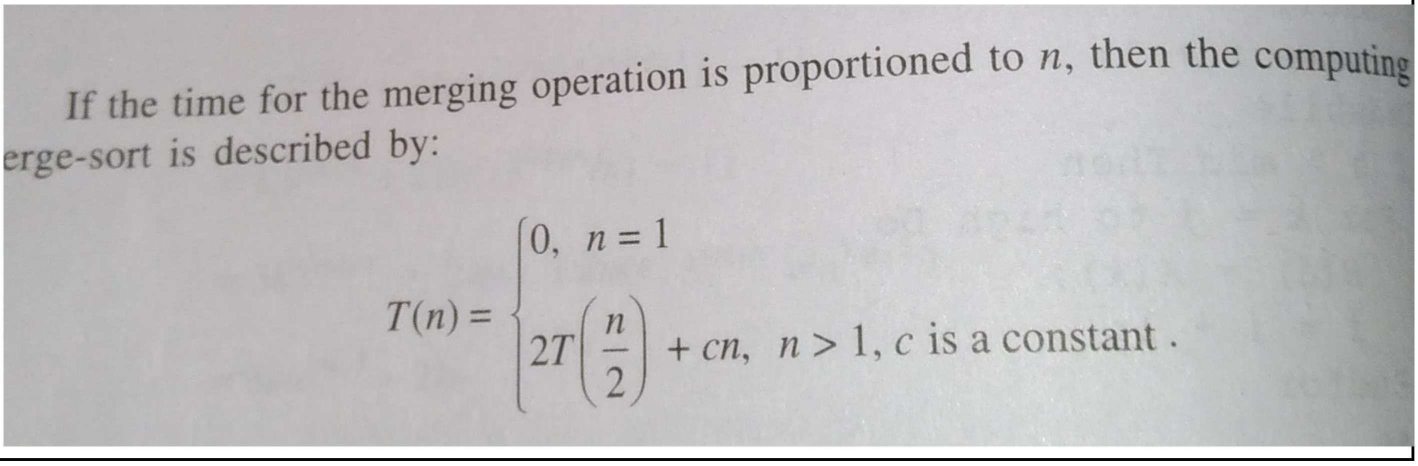
In the worst-case, the quick-sort requires quadratic time.

c) Apply Merge sort algorithm to sort the following numbers. Show each step clearly. 8, 7, 4, 5, 2, 11, 6, 1. Derive the complexity of Merge sort.

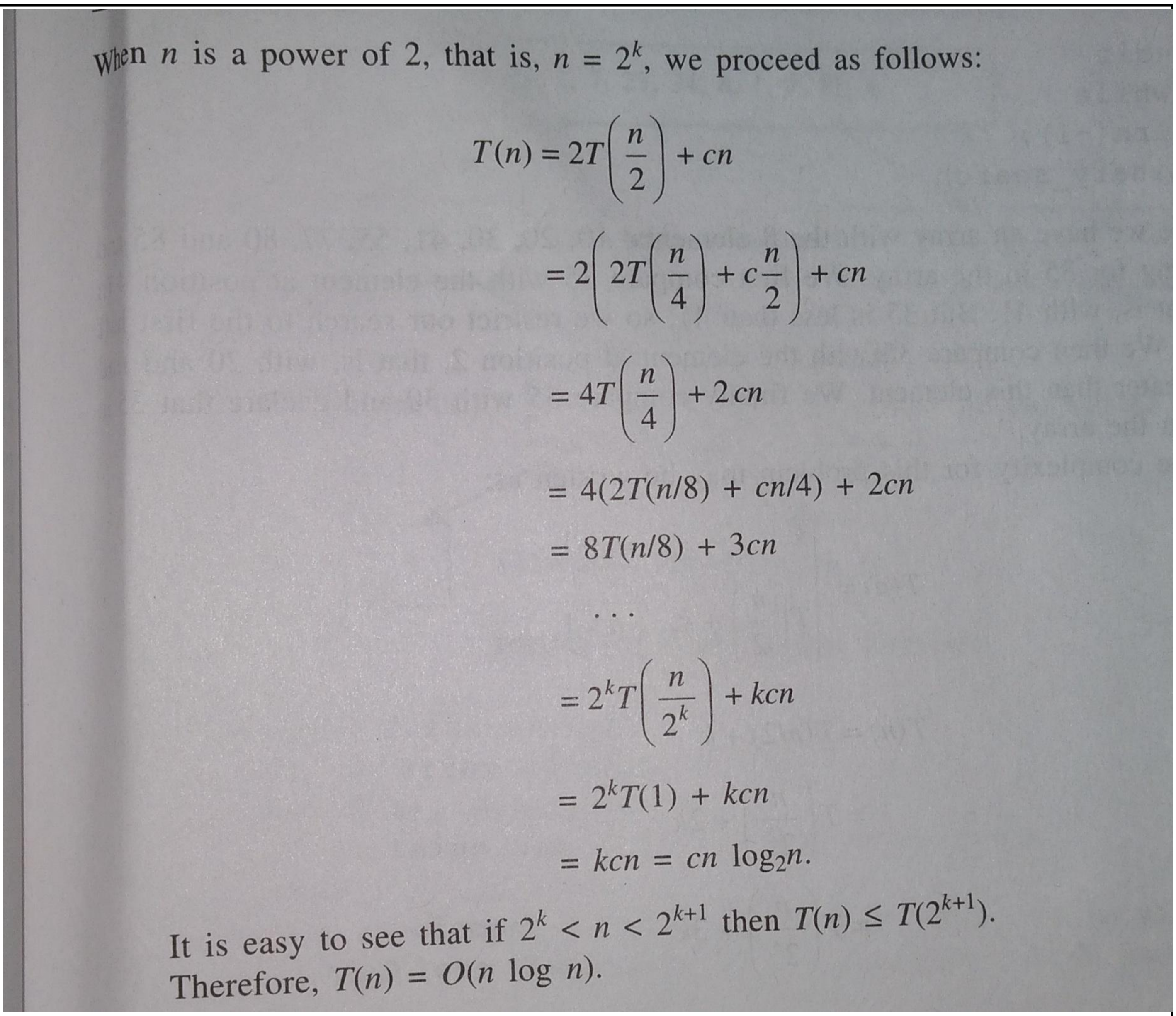
Ans:



8 7 4 5 2 11 6 1
7 8 4 5 2 11 1 6
4 5 7 8 1 2 6 11
1 2 4 5 6 7 8 11
Files of size 1 are merged
Files of size 2 are merged
Files of size 4 are merged


If the time for the merging operation is proportioned to n, then the computing
erge-sort is described by:

$$T(n) = \begin{cases} 0, & n = 1 \\ 2T\left(\dfrac{n}{2}\right) + cn, & n > 1,\ c \text{ is a constant}. \end{cases}$$

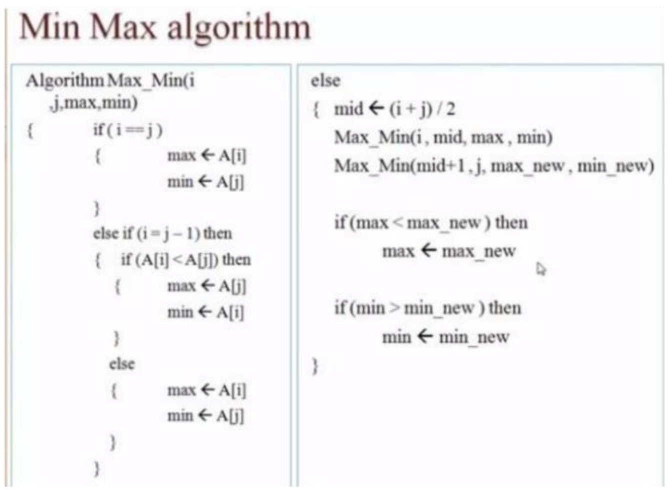
	 When n is a power of 2, that is, $n = 2^k$, we proceed as follows: $$T(n) = 2T\left(\frac{n}{2}\right) + cn$$ $$= 2\left(2T\left(\frac{n}{4}\right) + c\frac{n}{2}\right) + cn$$ $$= 4T\left(\frac{n}{4}\right) + 2cn$$ $$= 4(2T(n/8) + cn/4) + 2cn$$ $$= 8T(n/8) + 3cn$$ $$\ldots$$ $$= 2^kT\left(\frac{n}{2^k}\right) + kcn$$ $$= 2^kT(1) + kcn$$ $$= kcn = cn \log_2 n.$$ It is easy to see that if $2^k < n < 2^{k+1}$ then $T(n) \le T(2^{k+1})$. Therefore, $T(n) = O(n \log n)$.

Q7. Think and Answer

a)

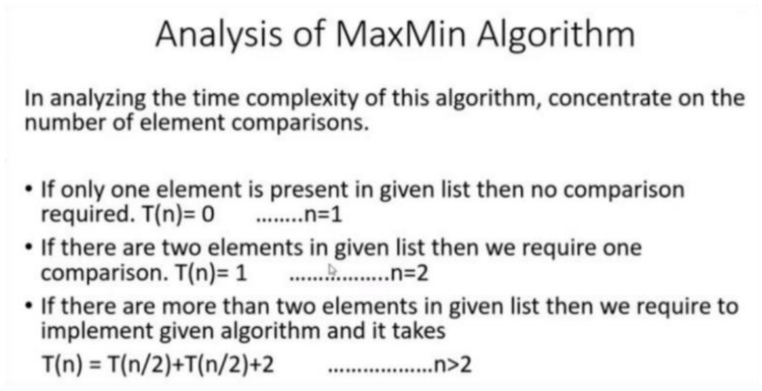
Write an algorithm to find the minimum and maximum elements simultaneously from a given list of elements. You are also required to discuss its running time.

Ans:



Min Max algorithm

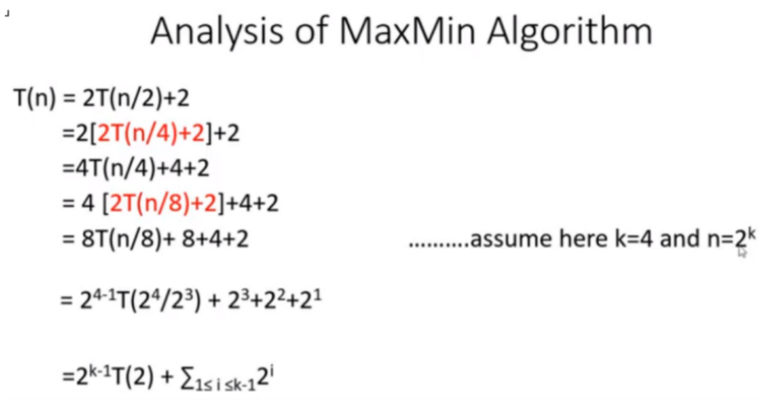
```
Algorithm Max_Min(i
    j,max,min)
{       if ( i == j )
        {       max ← A[i]
                min ← A[j]
        }
        else if (i = j – 1) then
        {   if (A[i] < A[j]) then
            {       max ← A[j]
                    min ← A[i]
            }
            else
            {       max ← A[i]
                    min ← A[j]
            }
        }
else
{   mid ← (i + j) / 2
    Max_Min(i , mid, max , min)
    Max_Min(mid+1 , j, max_new , min_new)

    if (max < max_new ) then
          max ← max_new

    if (min > min_new ) then
          min ← min_new
}
```



Analysis of MaxMin Algorithm

In analyzing the time complexity of this algorithm, concentrate on the number of element comparisons.

- If only one element is present in given list then no comparison required. $T(n)= 0$ ....n=1
- If there are two elements in given list then we require one comparison. $T(n)= 1$ ....n=2
- If there are more than two elements in given list then we require to implement given algorithm and it takes
  $T(n) = T(n/2)+T(n/2)+2$ ....n>2



Analysis of MaxMin Algorithm

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

$=2[2T(n/4)+2]+2$

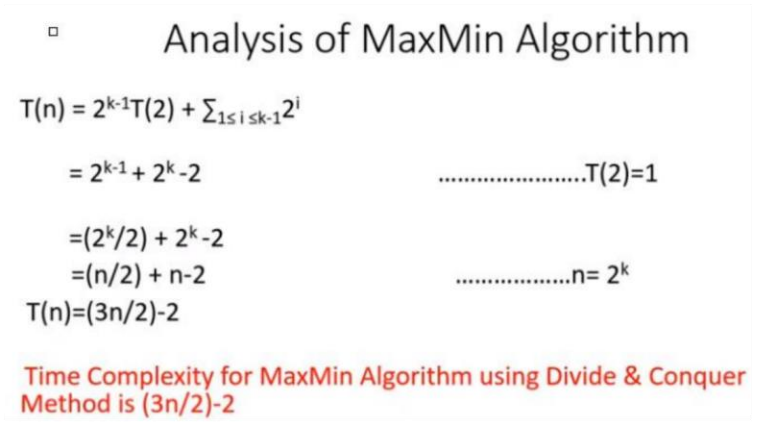
$=4T(n/4)+4+2$

$= 4\ [2T(n/8)+2]+4+2$

$= 8T(n/8)+ 8+4+2$ ....assume here k=4 and $n=2^k$

$= 2^{4-1}T(2^4/2^3) + 2^3+2^2+2^1$

$=2^{k-1}T(2) + \sum_{1\le i\le k-1} 2^i$



Analysis of MaxMin Algorithm

$T(n) = 2^{k-1}T(2) + \sum_{1\le i\le k-1} 2^i$

$= 2^{k-1} + 2^k -2$ ....T(2)=1

$=(2^k/2) + 2^k -2$

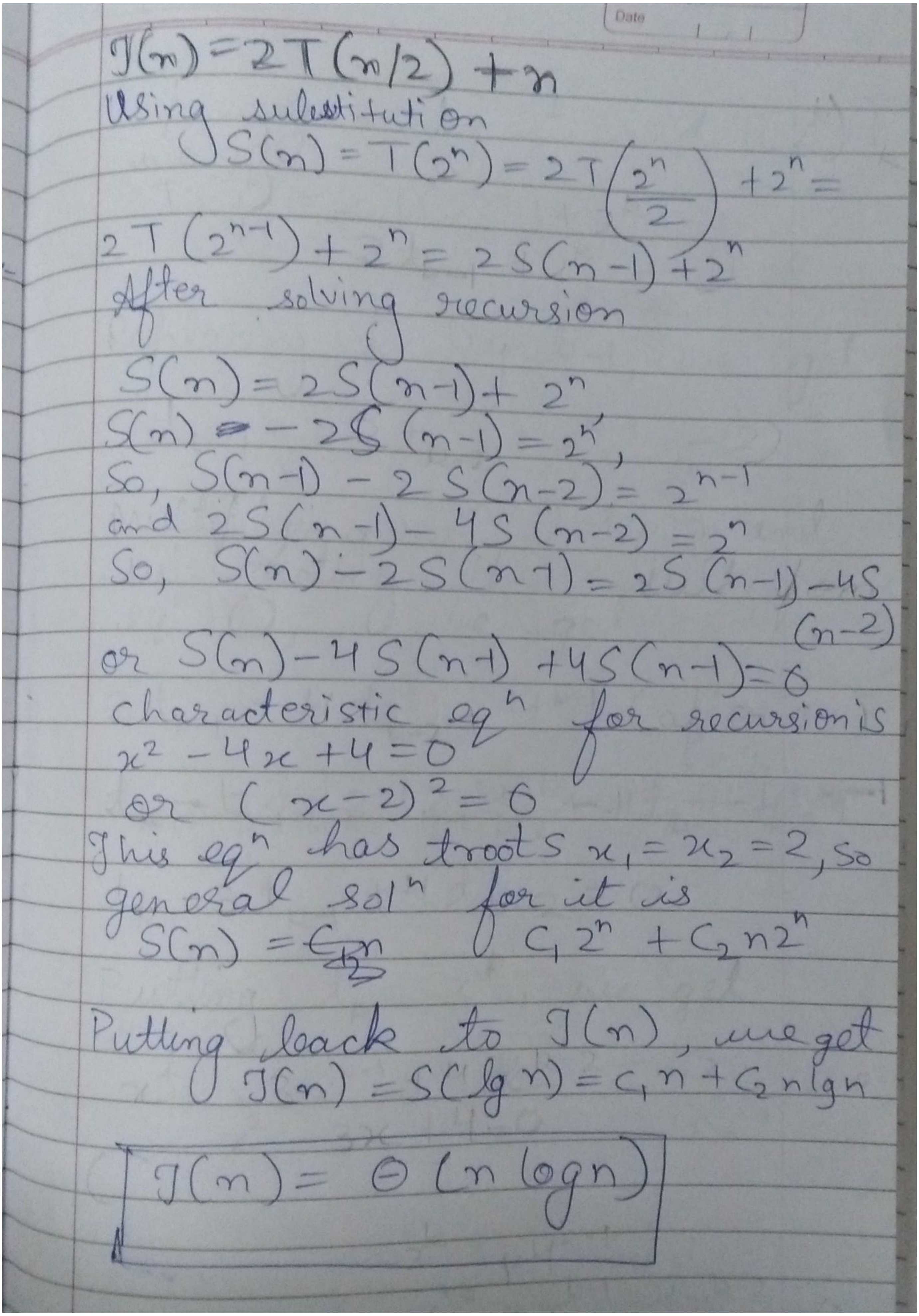
$=(n/2) + n-2$ ....$n= 2^k$

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

Time Complexity for MaxMin Algorithm using Divide & Conquer Method is (3n/2)-2

b) Determine the complexity of the recurrence relation T(n)=2T(n/2) + n

Ans:



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

Using substitution

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

$2T(2^{n-1}) + 2^n = 2S(n-1) + 2^n$

After solving recursion

$S(n) = 2S(n-1) + 2^n$

$S(n) - 2S(n-1) = 2^n,$

So, $S(n-1) - 2S(n-2) = 2^{n-1}$

and $2S(n-1) - 4S(n-2) = 2^n$

So, $S(n) - 2S(n-1) = 2S(n-1) - 4S(n-2)$

or $S(n) - 4S(n-1) + 4S(n-1) = 0$

characteristic eq^n for recursion is

$x^2 - 4x + 4 = 0$

or $(x-2)^2 = 0$

This eq^n has roots $x_1 = x_2 = 2$, so

general sol^n for it is

$S(n) = c_1 2^n + c_2 n 2^n$

Putting back to $T(n)$, we get

$T(n) = S(\lg n) = c_1 n + c_2 n \lg n$

$$\boxed{T(n) = \Theta(n \log n)}$$

Q8.	**My ideas**
a)	Explain Quick Sort that pick pivot in different ways. Quick Sort is based on the concept of **Divide and Conquer** algorithm, which is also the concept used in Merge Sort. The difference is, that in quick sort the most important or significant work is done while **partitioning (or dividing)** the array into subarrays, while in case of merge sort, all the major work happens during **merging** the subarrays. For quick sort, the combining step is insignificant. Quick Sort is also called **partition-exchange sort**. This algorithm divides the given array of numbers into three main parts: 1. Elements less than the pivot element 2. Pivot element 3. Elements greater than the pivot element Pivot element can be chosen from the given numbers in many different ways: 1. Choosing the first element 2. Choosing the last element (example shown) 3. Choosing any random element 4. Choosing the median For example: In the array {51, 36, 62, 13, 16, 7, 5, 24}, we take 24 as **pivot** (last element). So after the first pass, the list will be changed like this. {5 7 16 13 **24** 62 36 51} Hence after the first pass, the array is sorted such that all elements less than the chosen pivot are on its left side and all greater elements on its right side. The pivot is finally at the position it is supposed to be in the sorted version of the array. Now the subarrays {5 7 16 13} and {62 36 51} are considered as two separate arrays, and same recursive logic will be applied on them, and we will keep doing this until the complete array is sorted.
b) Ans:	Drawbacks of Merge Sort ● For small datasets, merge sort is slower than other sorting algorithms. ● For the temporary array, mergesort requires an additional space of O (n). ● Even if the array is sorted, the merge sort goes through the entire process.