Part - A
(1 x 10 = 10 Marks)

Instructions: Answer all

Q. No	Question	Marks	BL	CO	PO	PI Code
1.	**Asymptotic notation describes:** A. The characteristics of a function in the limit B. The correctness of the algorithm **C. Running time of an algorithm** D. All the above	1	1	1	**1-4,10,12**	
2.	Best-case time complexity of binary search is ________. A. O(n) B. O(n+1) **C. O(1)** D. O(log n)	1	1	1	**1-4,10,12**	
3.	Find the algorithm that is not suitable for large data sets as its worst-case complexity is $O(n^2)$ where n is the number of items. A. Insertion Sort **B. Bubble Sort** C. Linear Search D. Binary Search					

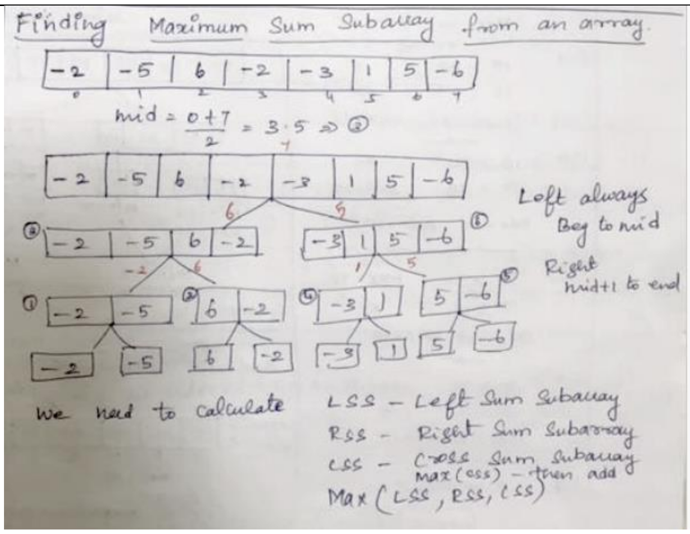
4.	Calculate the time complexity analysis for the following code using step count method.
	F1( )
	{
	for i:= 1 to n do
	print(“DAA”)
	}
	A. n+3
	B. n+1
	C. n+3
	D. 2n
		1	1	1	**1-4,10,12**	

5.	Determine the value of a2 for the recurrence relation an = 17an-1 + 30n with a0=3. A. 4387 B. 5484 C. 238 **D. 1437**	1	1	1	**1-4,10,12**	
6.	Select the technique which is not based on divide-and-conquer programming approach: A. Merge Sort B. Quick Sort					

	C. Binary Search **D. Insertion sort**					
7.	What are a and b in the master theorem? **A. a = Number of subproblems and b = The cost of dividing and merging the subproblems.** B. a = The cost of dividing and merging the subproblems and b = Number of subproblems C. a = Number of problems and b = The cost of merging the subproblems. D. None	1	1	2	**1-4,10,12**	
8.	Find the recurrence relation of merge sort A. T(n)=T(n/2) + 1 B. T(n)=2T(n/2) + 1 C. T(n)=T(n/2) + n **D. T(n)=2T(n/2) + n**	1	1	2	**1-4,10,12**	
9.	Calculate the maximum sum for the given array A={5,-4,-2,6,1} A. 5 B. 4 C. **7** D. 2	1	1	2	**1-4,10,12**	
10.	Compute the value for the given recurrence relation using master theorem: T(n)= 2T(n/2)+1 A. O(n^2) B. O(log n) C. **O(n)** D. O(n log n)	1	1	2	**1-4,10,12**	

Part – B **(5 x 4 = 20 Marks)** **Instructions: Answer All the Questions**						
11	Prove that $f(n)=n^2+2n+3$ is $O(n^2)$ **Proof**: by the Big-Oh definition, T(n) is $O(n^2)$ if T(n) $\le c\cdot n^2$ for some $n \ge n0$. Let us check this condition: if $n^2 + 2n + 3 \le c\cdot n^2$ then $1 + \frac{2}{n^2} + \frac{1}{n^2} \le c$. Therefore, the Big-Oh condition holds for $n \ge n_0 = 1$ and $c \ge 5$ (= 1 + 2 + 3). Larger values of n_0 result in	5	3	1	**1-4,10,12**	1.2.1

	smaller factors c, but in any case, the above statement is valid.					
12	In the book shop, eight new books were purchased. Mr. Kailash wants to check the book details of the book id “206”. Use linear search algorithm and apply that search technique to retrieve the book details from the given Data: 201, 202, 203, 204, 205, 206, 207, 208. How many comparisons will the algorithm take to find the book id 206. Analyse the worst-case time complexity for the above scenario. **How many comparisons will the algorithm take to find the book id 206.** Let's count the number of comparisons: Compare 201 with 206 - No match Compare 202 with 206 - No match Compare 203 with 206 - No match Compare 204 with 206 - No match Compare 205 with 206 - No match Compare 206 with 206 - **Match found** So, the linear search algorithm will take a total of 6 comparisons to find the book ID "206" in this particular list. **Analyse the worst-case time complexity for the above scenario.** **Worst-Case Time Complexity: O(n)** - If the target element is not present in the list, the algorithm needs to iterate through all n elements before determining that the element is not there.	5	4	1	**1-4,10,12**	2.6.4
13	Calculate maximum sum of the given array: A= [-2, -5,6, -2, -3,1,5, -6]	5	2	2	**1-4,10,12**	1.2.1



Finding Maximum Sum Subarray from an array.

-2	-5	6	-2	-3	1	5	-6
0	1	2	3	4	5	6	7

mid = $\frac{0+7}{2}$ = 3.5 ⇒ 3

Left always Beg to mid

Right mid+1 to end

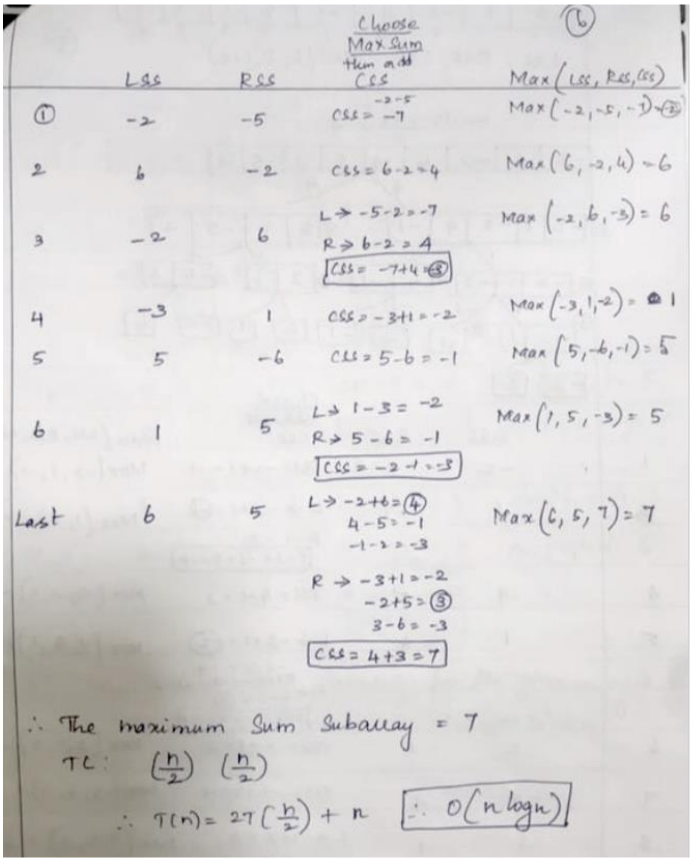
We need to calculate LSS – Left Sum Subarray

RSS – Right Sum Subarray

CSS – Cross Sum Subarray

Max(css) – then add

Max(LSS, RSS, CSS)



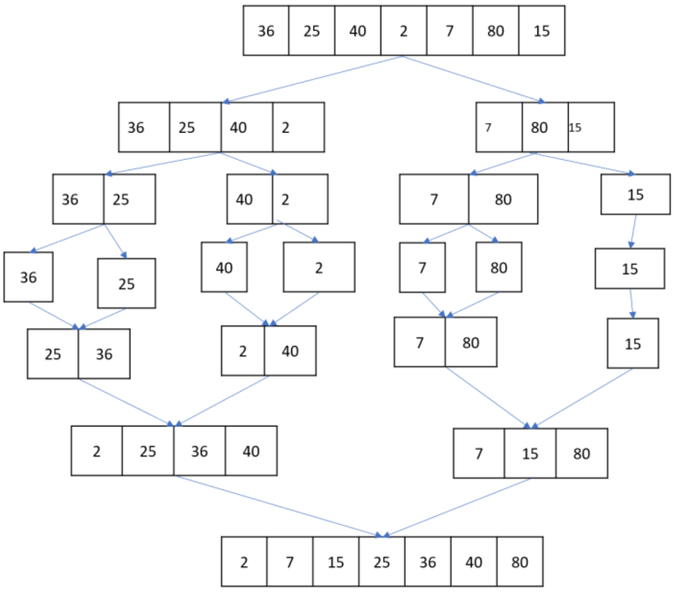
	LSS	RSS	Choose Max Sum then add CSS	Max(LSS, RSS, CSS)
1	-2	-5	CSS = -2-5 = -7	Max(-2, -5, -7) = -2
2	6	-2	CSS = 6-2 = 4	Max(6, -2, 4) = 6
3	-2	6	L → -5-2 = -7 R → 6-2 = 4 $\boxed{CSS = -7+4 = -3}$	Max(-2, 6, -3) = 6
4	-3	1	CSS = -3+1 = -2	Max(-3, 1, -2) = 1
5	5	-6	CSS = 5-6 = -1	Max(5, -6, -1) = 5
6	1	5	L → 1-3 = -2 R → 5-6 = -1 $\boxed{CSS = -2-1 = -3}$	Max(1, 5, -3) = 5
Last	6	5	L → -2+6 = 4 4-5 = -1 -1-2 = -3 R → -3+1 = -2 -2+5 = 3 3-6 = -3 $\boxed{CSS = 4+3 = 7}$	Max(6, 5, 7) = 7

∴ The maximum Sum Subarray = 7

TC: $\left(\frac{n}{2}\right)$ $\left(\frac{n}{2}\right)$

∴ $T(n) = 2T\left(\frac{n}{2}\right) + n$ $\boxed{\therefore O(n \log n)}$

14	Consider the following example of an unsorted array, sort the elements using Merge Sort algorithm. A1= (36,25,40,2,7,80,15). Write the recurrence relation and perform the worst-case analysis.	5	2	2	**1-4,10,12**	2.6.4



The recurrence relation for the Merge Sort algorithm can be expressed as follows:
T(n)=2T($\frac{n}{2}$)+O(n)

- The term 2T(2n) accounts for the two recursive calls on arrays of size $\frac{n}{2}$
- The term O(n) represents the time complexity of merging two sorted arrays of size $\frac{n}{2}$

This recurrence relation is in the form of the Master Theorem, and for Merge Sort, it falls into Case 2 of the Master Theorem.

The Master Theorem states the form:

$T(n)=aT(\frac{n}{b}) + f(n)$, where a, b, and $f(n)$ are constants.

In Case 2, if f(n) = $\Theta(n^c \log^k n)$ with $c = \log_b a$, then the time complexity is $\Theta(n^c \log^{k+1} n)$.

For merge sort, a=2, b=2, c=1 and k=0.

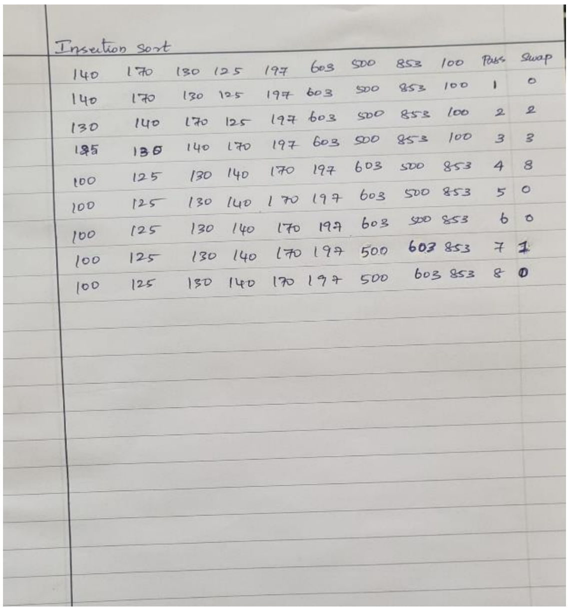
Since $c=\log_2 2$, the recurrence falls into case 2. Therefore, the worst-case complexity of merge sort is: T(n)= Θ (n log n).

Part – C
(2 x 10 = 20 Marks)
Instructions: Answer All the Questions

15.A	Assume a sequence of books with the following labels: A= {140,170,130,125,197,603,500,853,100} in unsorted order. Write the suitable algorithm to arrange the books in desired order and also write how many comparisons and swaps are required for arranging these books in this instance.	10	3	1	**1-4,10,12**	2.6.4

Insertion Sort

```
InsertionSort(arr):
    n = length(arr)
    for i from 1 to n-1:
        current_element = arr[i]
        j = i - 1
        while j >= 0 and arr[j] > current_element:
            arr[j + 1] = arr[j]
            j = j - 1
        arr[j + 1] = current_element
```


Insertion sort

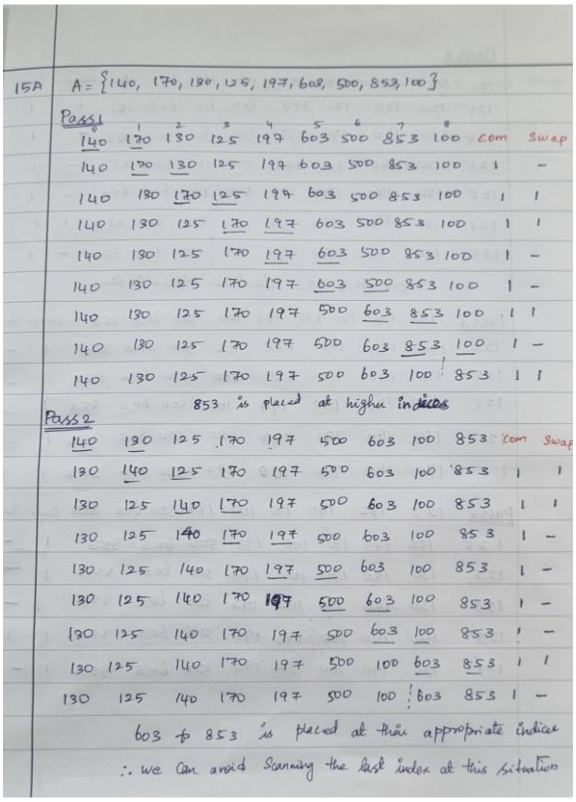
140	170	130	125	197	603	500	853	100	Pass	Swap
140	170	130	125	197	603	500	853	100	1	0
130	140	170	125	197	603	500	853	100	2	2
125	130	140	170	197	603	500	853	100	3	3
100	125	130	140	170	197	603	500	853	4	8
100	125	130	140	170	197	603	500	853	5	0
100	125	130	140	170	197	603	500	853	6	0
100	125	130	140	170	197	500	603	853	7	1
100	125	130	140	170	197	500	603	853	8	0

- In the worst-case scenario, when the array is sorted in reverse order, each element must be compared and swapped with each preceding element until it reaches its correct position.
- So, for an array of n elements in the worst case:
- Comparisons: $\frac{n}{2}$(n−1) in the worst case
- Swaps: $\frac{n}{2}$(n−1) in the worst case

Worst Case Complexity - It occurs when the array elements are required to be sorted in reverse order. That means suppose you have to sort the array elements in ascending order, but its elements are in descending order. The worst-case time complexity of insertion sort is O(n2).

Bubble sort

```
begin BubbleSort(arr)
  for all array elements
    if arr[i] > arr[i+1]
      swap(arr[i], arr[i+1])
    end if
  end for
  return arr
end BubbleSort
```



15A A = {140, 170, 130, 125, 197, 603, 500, 853, 100}

Pass 1

0	1	2	3	4	5	6	7	8	Com	Swap
140	170	130	125	197	603	500	853	100		
140	170	130	125	197	603	500	853	100	1	–
140	130	170	125	197	603	500	853	100	1	1
140	130	125	170	197	603	500	853	100	1	1
140	130	125	170	197	603	500	853	100	1	–
140	130	125	170	197	603	500	853	100	1	–
140	130	125	170	197	500	603	853	100	1	1
140	130	125	170	197	500	603	853	100	1	–
140	130	125	170	197	500	603	100	853	1	1

853 is placed at higher indices

Pass 2

									Com	Swap
140	130	125	170	197	500	603	100	853		
130	140	125	170	197	500	603	100	853	1	1
130	125	140	170	197	500	603	100	853	1	1
130	125	140	170	197	500	603	100	853	1	–
130	125	140	170	197	500	603	100	853	1	–
130	125	140	170	197	500	603	100	853	1	–
130	125	140	170	197	500	603	100	853	1	–
130	125	140	170	197	500	100	603	853	1	1
130	125	140	170	197	500	100	603	853	1	–

603 & 853 is placed at their appropriate indices

∴ we can avoid scanning the last index at this situation

Pass 3

130	125	140	170	197	500	100	603	853	Swap	Com
125	130	140	170	197	500	100	603	853	1	1
125	130	140	170	197	500	100	603	853	-	1
125	130	140	170	197	500	100	603	853	-	1
125	130	140	170	197	500	100	603	853	-	1
125	130	140	170	197	100	500	603	853	1	1
125	130	140	170	197	100	500	603	853	-	1

500, 603, 853 Placed in their index

Pass 4

125	130	140	170	197	100	500	603	853	Comp	swap
125	130	140	170	197	100	500	603	853	1	-
125	130	140	170	197	100	500	603	853	1	-
125	130	140	170	197	100	500	603	853	1	-
125	130	140	170	197	100	500	603	853	1	-
125	130	140	170	100	197	500	603	853	1	1

Pass 5

125	130	140	170	100	197	500	603	853	Comp	Swap
125	130	140	170	100	197	500	603	853	1	-
125	130	140	170	100	197	500	603	853	1	-
125	130	140	170	100	197	500	603	853	1	-
125	130	140	100	170	197	500	603	853	1	1
125	130	140	100	170	197	500	603	853	1	-

Pass 6

125	130	140	100	170	197	500	603	853	Comp	Swap
125	130	140	100	170	197	500	603	853	1	-
125	130	140	100	170	197	500	603	853	1	-
125	130	100	140	170	197	500	603	853	1	1
125	130	100	140	170	197	500	603	853	1	-

Pass 7

125	130	100	140	170	197	500	603	853	Comp	Swap
125	130	100	140	170	197	500	603	853	1	-
125	100	130	140	170	197	500	603	853	1	1

Pass 8

125	100	130	140	170	197	500	603	853	Comp	Swap
100	125	130	140	170	197	500	603	853	1	1

sorted array

A = {100, 125, 130, 140, 170, 197, 500, 603, 853}

	• **Number of Comparisons**: In the worst case, bubble sort will make approximately $\frac{n(n-1)}{2}$ comparisons. This is because, in each pass, it compares each element with its adjacent element, and there are n-1 pairs of adjacent elements in an array of size n. So, the number of comparisons is $\frac{n(n-1)}{2}$ • **Number of Swaps**: Similarly, in the worst case, bubble sort will make approximately $\frac{n(n-1)}{2}$ swaps. This is because, in each pass, it swaps adjacent elements if they are in the wrong order, and there are n-1 pairs of adjacent elements in an array of size n. So, the number of swaps is $\frac{n(n-1)}{2}$ **Worst Case Complexity -** It occurs when the array elements are required to be sorted in reverse order. That means suppose you have to sort the array elements in ascending order, but its elements are in descending order. The worst-case time complexity of bubble sort is **O(n^2).**					
OR						
15.B	Solve the recurrence relation: i. T(n) = T(n-1) + n using forward substitution method	5+5	3	1	**1-4,10,12**	1.2.1

	forward Sub (6) $T(n) = T(n-1) + n \quad T(0) = 0$ $T(0) = 0$ $n=1 \quad T(1) = T(1-1) + 1 = T(0) + 1 = 0 + 1 \Rightarrow 1$ $n=2 \quad T(2) = T(2-1) + 2 = T(1) + 2 = 1 + 2 \Rightarrow 3$ $n=3 \quad T(3) = T(3-1) + 3 = T(2) + 3 = 3 + 3 \Rightarrow 6$ $n=4 \quad T(4) = T(4-1) + 4 = T(3) + 4 = 6 + 4 \Rightarrow 10$ $n=5 \quad T(5) = T(4) + 5 = 10 + 5 = 15$ $n=k \quad T(k) = T(k-1) + 1$ $T(0) = 0$, $T(1) = 0 + 1 = 1$, $T(2) = 1 + 1 = 2$, $T(3) = 2 + 1 = 3$, $T(4) = 3 + 1 = 4$ $T(n) = 1 + 2 + 3 + 4 + 5 \ldots k.$ $n + (n+1) + (n+2) + (n+3) + (n+4) \ldots (n+k)$ $\therefore T(n) = \frac{k(k+1)}{2} \Rightarrow \frac{k^2+k}{2}$ $\therefore O(k^2) / O(n^2)$ ii. T(n) = T(n/2) + n using recurrence tree $T(n) = \begin{cases} 1 & n=1 \\ T(n/2)+n & n>1 \end{cases}$ amount of work done in each step $T(n)$: n, $T(n/2)$ — n $n/2$, $T(n/2^2)$ — $n/2$ $n/2^2$, $T(n/2^3)$ — $n/2^2$ $T(n/2^k)$ — $n/2^k$ Now the $T(n) = n + \frac{n}{2} + \frac{n}{2^2} + \frac{n}{2^3} + \cdots + \frac{n}{2^k}$ $= n\left[1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \cdots \frac{1}{2^k}\right]$ $= n\left[\sum_{i=0}^{k} \frac{1}{2^i}\right] \quad \boxed{\because \sum_{i=0}^{k} \frac{1}{2^i} = 1}$ $T(n) = n$ $\therefore O(n)$					
16.A	The following values are given in an array. Assume the first element as pivot and sort the following elements 144,133,111,155,177,190,140,160,199,122. Write the recurrence relation for the above condition and discuss the worst-case time complexity.	10	3	2	**1-4,10,12**	2.6.4

144, 133, 111, 155, 177, 190, 140, 160, 199, 122

144	133	111	155	177	190	140	160	199	122

Pivot → 144

* Rearrange the array such that elements less than the pivot are on the left and elements greater than the pivot are on the right.

→ 144 (i) 133 111 155 177 190 140 160 199 122 (j)

144 133 111 155 (i) 177 190 140 160 199 122 (j) — i – Increment, j – decrement, 155 > 122 ∴ swap

144 133 111 122 (i) 177 190 140 160 199 155 (j) — i – Increment, j – decrement, 177 > 140 ∴ swap

144 133 111 122 177 (i) 190 140 (j) 160 199 155

144 133 111 122 140 (i) 190 170 (j) 160 199 155

144 133 111 122 140 (j) 190 (i) 170 160 199 155 — i – increment, j – decrement, i & j passed each other

* Since i & j passed each other, the location for pivot is fixed at j^{th} location.

∴ Swap pivot and j, this is partioning.

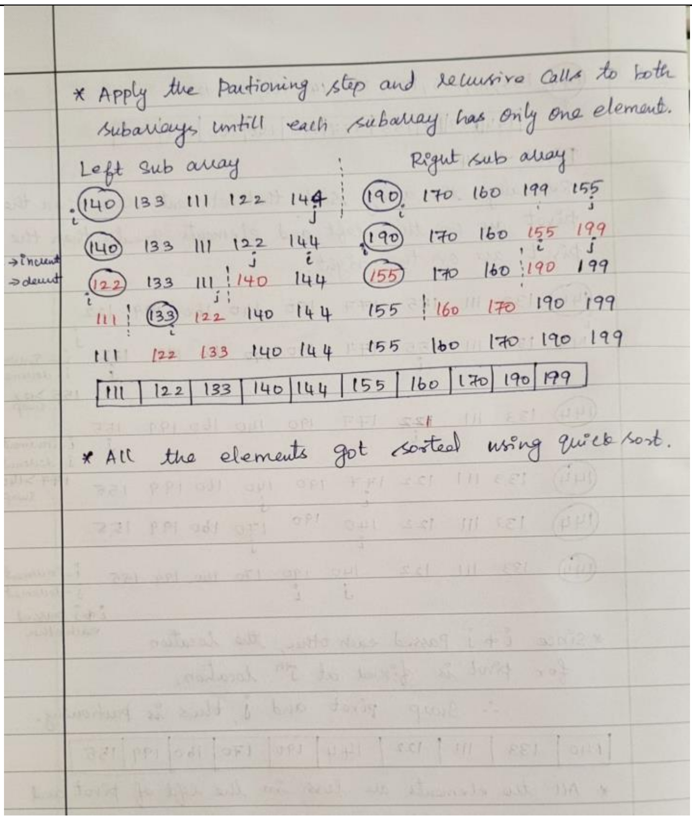
140	133	111	122	144	190	170	160	199	155

* All the elements are less in the left of pivot and all the elements are greater in the right of pivot.

Left Sub array: | 140 | 133 | 111 | 122 |

Pivot: | 144 |

Right sub array: | 190 | 170 | 160 | 199 | 155 |



* Apply the Partioning step and recursive calls to both subarrays untill each subarray has only one element.

→increment
→decrement

Left Sub array					Right sub array				
140	133	111	122	144	190	170	160	199	155
140	133	111	122	144	190	170	160	155	199
122	133	111	140	144	155	170	160	190	199
111	133	122	140	144	155	160	170	190	199
111	122	133	140	144	155	160	170	190	199

111	122	133	140	144	155	160	170	190	199

* All the elements got sorted using quick sort.

In quicksort, the recurrence relation is based on the partitioning of the array with respect to a chosen pivot element.

Let's denote the size of the array to be sorted as n, and the recurrence relation is given by:

$T(n)=T(k)+T(n -k-1) + O(n)$

where:

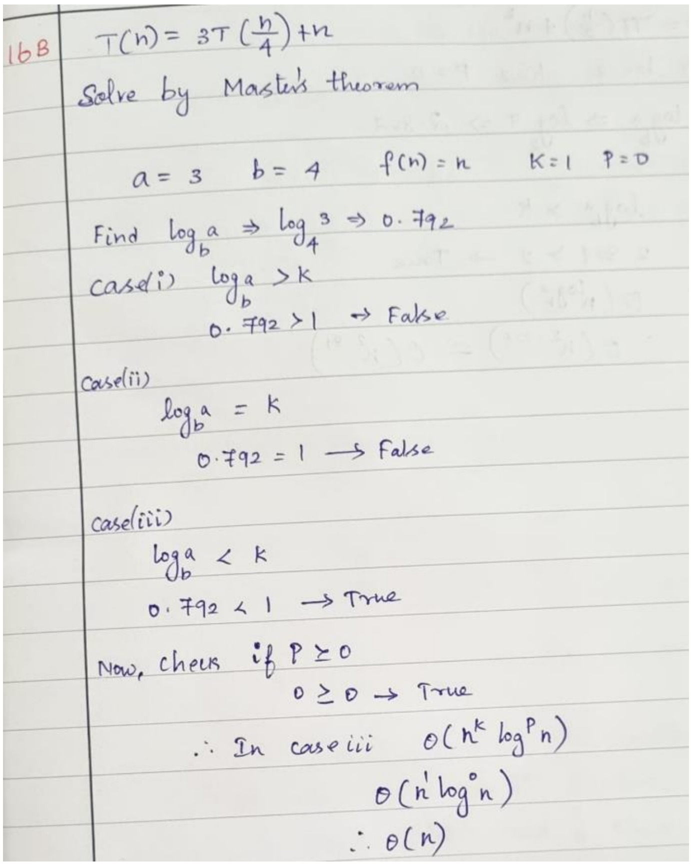
- $T(n)$ is the time complexity for sorting an array of size n.
- k is the position of the pivot after the partitioning step.

The **worst-case time complexity is $O(n^2)$.**

<table>
<tr><td></td><td>In the best and average case, when the pivot divides the array into two roughly equal parts, the recurrence relation becomes:

$T(n)=2T(\frac{n}{2})+O(n)$

The solution to the recurrence relation in the best and average case is $O(n \log n)$, making quick sort an efficient sorting algorithm on average.</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="7">OR</td></tr>
<tr><td>16.B</td><td>Solve the given recurrence relation using Master's Method
i. T(n) = 3T(n/4)+n


16B
$T(n) = 3T(\frac{n}{4})+n$
Solve by Master's theorem
$a = 3$ $b = 4$ $f(n) = n$ $K=1$ $P=0$
Find $\log_b a \Rightarrow \log_4 3 \Rightarrow 0.792$
case(i) $\log_b a > K$
$0.792 > 1 \rightarrow$ False
case(ii)
$\log_b a = K$
$0.792 = 1 \rightarrow$ False
case(iii)
$\log_b a < K$
$0.792 < 1 \rightarrow$ True
Now, check if $P \geq 0$
$0 \geq 0 \rightarrow$ True
$\therefore$ In case iii $\theta(n^k \log^p n)$
$\theta(n^1 \log^0 n)$
$\therefore \theta(n)$

ii. T(n) = 7T(n/2)+n^2</td><td>5+5</td><td>3</td><td>2</td><td>1-4,10,12</td><td>2.5.1</td></tr>
</table>

	16B. $T(n) = 7T\left(\frac{n}{2}\right) + n^2$ $a = 7 \quad b = 2 \quad K = 2 \quad P = 0$ Find $\log_b a \Rightarrow \log_2 7 \Rightarrow 2.807$ Case: $\log_b a > K$ $2.807 > 2 \rightarrow$ True $\therefore \Theta(n^{\log_b a})$ $\therefore \Theta(n^{2.807}) \simeq \Theta(n^{2.81})$					

16. a.

11. 44, 33, 11, 55, 77, 90, 40, 60, 99, 22

44	33	11	55	77	90	40	60	99	22
22	33	11	55	77	90	40	60	99	44
22	33	11	55	77	90	40	60	99	44
22	33	11	44	77	90	40	60	99	55
22	33	11	40	77	90	44	60	99	55
22	33	11	40	44	90	77	60	99	55

Now, at this stage all elements present in left side are smaller than 44 and right side elements are greater, Change the pivot for the partition.

22	33	11	40	44 |	90	77	60	99	55
22	33	11	40	44	55	77	60	99	90
22	33	11	40	44	55	77	60	99	90
11	33	22	40	44	55	77	60	99	90
11	33	22	40	44	55	77	60 |	90	99
11	22	33	40	44 |	55	77	60 |	90	99
11	22	33	40	44	55	77	60	90	99
11	22	33	40	44	55 |	77	60 |	90	99
11	22	33	40	44	55	60	77	90	99

The elements are sorted in ascending order.

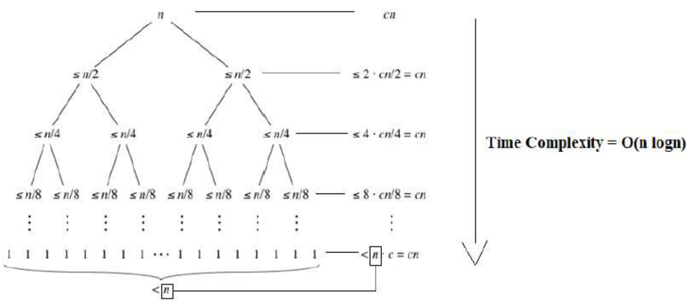
Best case scenario: The best case scenario occurs **when the partitions are as evenly balanced as possible,** i.e their sizes on either side of the pivot element are either are equal or are have size difference of 1 of each other.

•Case 1: The case when sizes of sublist on either side of pivot becomes equal occurs when the subarray has an odd number of elements and the pivot is right in the middle after partitioning. **Each partition will have (n-1)/2 elements.**

•Case 2: The size difference of 1 between the two sublists on either side of pivot happens if the subarray has an even number, n, of elements. **One partition will have n/2 elements with the other having (n/2)-1.**

In either of these cases, **each partition will have at most n/2 elements**, and the tree representation of the subproblem sizes will be as below:

Quick Sort Best Case Scenario



16. B

i. $T(n) = 3T(n/4)+n$

solu $a = 3$ $b = 4$ $f(n) = n$ $k = 1$ $P = 0$

Find $\log_b a = \log_4 3$
$= 0.792$

Check Case1: $\log_b a > k$
$0.792 > 1 \rightarrow$ False

Case2: $\log_b a = k$
$0.792 \neq 1 \longrightarrow$ False

Case 3: $\log_b a < k$
$0.792 < 1 \rightarrow$ True

check
if $P \geq 0$
$0 \geq 0$ – True

$\therefore \Theta(n^k \log^p n)$
$\Theta(n^1 \log^0 n)$
$\boxed{\therefore \Theta(n)}$

ii. T(n) = 7T(n/2)+n^2

$T(n) = 7T\left[\frac{n}{2}\right) + n^2$

$a = 7$ $b = 2$ $K = 2$ $P = 0$

$\log_b a = \log_2 7 = 2.807$

$\because \log_b a > K$
$2.807 > 2$
$\therefore \Theta(n^{\log_b a})$
$\therefore \Theta(n^{2.807})$