Bharatiya Vidya Bhavan's

Sardar Patel Institute of Technology

Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Name	**Pratik Pujari**		
UID no.	**2020300054**	**Class:**	**Comps C Batch**
Experiment No.	2		

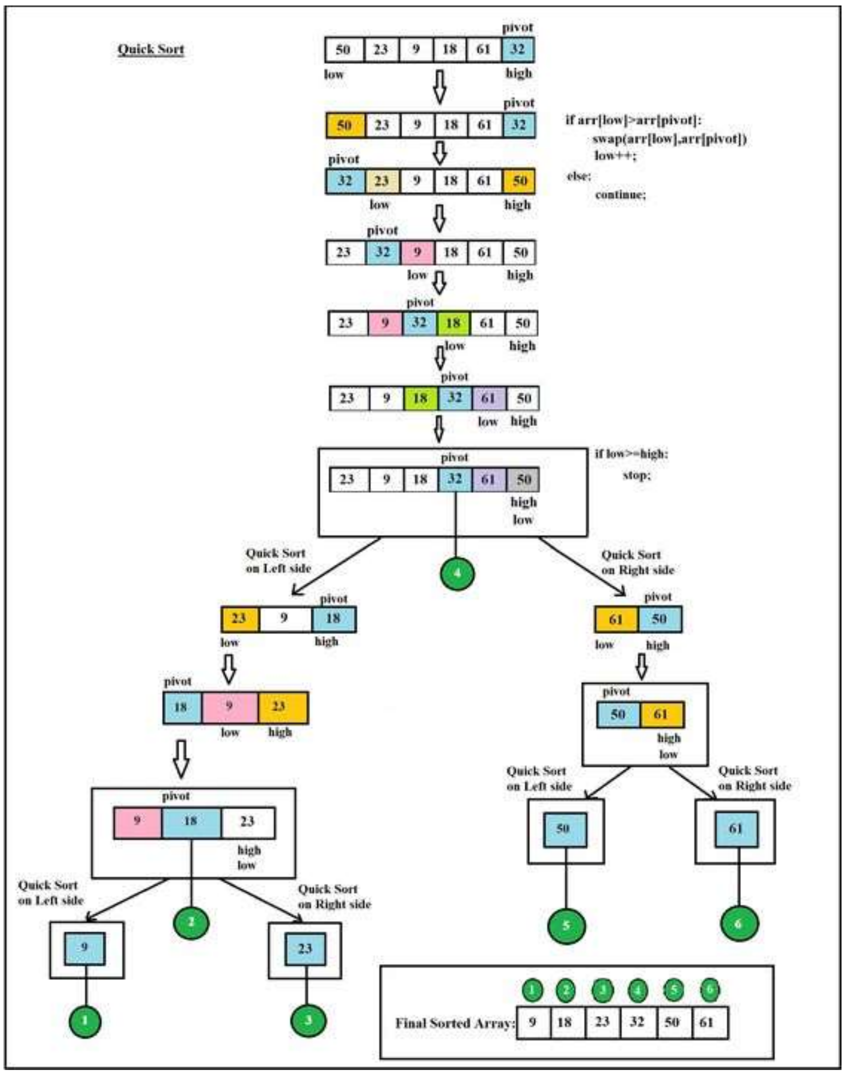
AIM:	To implement divide and conquer sorting algorithms
THEORY:	**What is QuickSort ?** Quick Sort is a divide and conquer algorithm. It creates two empty arrays to hold elements less than the pivot value and elements greater than the pivot value, and then recursively sort the sub arrays. There are two basic operations in the algorithm, swapping items in place and partitioning a section of the array. **Important Characteristics of Quick Sort:** • Quick Sort is useful for sorting arrays. • In efficient implementations Quick Sort is not a stable sort, meaning that the relative order of equal sort items is not preserved. • Overall time complexity of Quick Sort is O(nLogn). In the worst case, it makes O(*n*2) comparisons, though this behavior is rare. • The space complexity of Quick Sort is O(nLogn). It is an in-place sort (i.e. it doesn't require any extra storage) **Problem Statement** Consider the following array: 50, 23, 9, 18, 61, 32. We need to sort this array in the most efficient manner without using extra place (inplace sorting).

	Solution **Step 1**: • **Make any element as pivot:** Decide any value to be the pivot from the list. For convenience of code, we often select the rightmost index as pivot or select any at random and swap with rightmost. Suppose for two values "Low" and "High" corresponding to the first index and last index respectively. ○ In our case low is 0 and high is 5. ○ Values at low and high are 50 and 32 and value at pivot is 32. • **Partition the array on the basis of pivot:** Call for partitioning which rearranges the array in such a way that pivot (32) comes to its actual position (of the sorted array). And to the left of the pivot, the array has all the elements less than it, and to the right greater than it. ○ In the partition function, we start from the first element and compare it with the pivot. Since 50 is greater than 32, we don't make any change and move on to the next element 23. ○ Compare again with the pivot. Since 23 is less than 32, we swap 50 and 23. The array becomes 23, 50, 9, 18, 61, 32 ○ We move on to the next element 9 which is again less than pivot (32) thus swapping it with 50 makes our array as 23, 9, 50, 18, 61, 32. ○ Similarly, for next element 18 which is less than 32, the array becomes 23, 9, 18, 50, 61, 32. Now 61 is greater than pivot (32), hence no changes. ○ Lastly, we swap our pivot with 50 so that it comes to the correct position. Thus the pivot (32) comes at its actual position and all elements to its left are lesser, and all elements to the right are greater than itself. **Step 2**: The main array after the first step becomes 23, 9, 18, 32, 61, 50

Step 3: Now the list is divided into two parts:
1. Sublist before pivot element
2. Sublist after pivot element

Step 4: Repeat the steps for the left and right sublists recursively. The final array thus becomes
9, 18, 23, 32, 50, 61.

The following diagram depicts the workflow of the Quick Sort algorithm which was described above.



```
if arr[low]>arr[pivot]:
    swap(arr[low],arr[pivot])
    low++;
else:
    continue;
```

```
if low>=high:
    stop;
```

ALGORITHM:

1. Find a "pivot" item in the array. This item is the basis for comparison for a single round.
2. Start a pointer (the left pointer) at the first item in the array.
3. Start a pointer (the right pointer) at the last item in the array.
4. While the value at the left pointer in the array is less than the pivot value, move the left pointer to the right (add 1). Continue until the value at the left pointer is greater than or equal to the pivot value.
5. While the value at the right pointer in the array is greater than the pivot value, move the right pointer to the left (subtract 1). Continue until the value at the right pointer is less than or equal to the pivot value.
6. If the left pointer is less than or equal to the right pointer, then swap the values at these locations in the array.
7. Move the left pointer to the right by one and the right pointer to the left by one.
8. If the left pointer and right pointer don't meet, go to step 1

Appliations of quick sort

- Commercial Computing is used in various government and private organizations for the purpose of sorting various data like sorting files by name/date/price, sorting of students by their roll no., sorting of account profile by given id, etc.
- The sorting algorithm is used for information searching and as Quicksort is the fastest algorithm so it is widely used as a better way of searching.
- It is used everywhere where a stable sort is not needed.
- It is tail -recursive and hence all the call optimization can be done.

Bharatiya Vidya Bhavan’s

Sardar Patel Institute of Technology

Bhavan’s Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

EXPERIMENT 1	

CODE:	Program Code
	Quicksort:
	`import java.util.Arrays;`
	
	`public class quickSort {`
	
	`    int totalSwaps = 0;`
	`    int totalComparisons = 0;`
	
	`    // This function takes last element as pivot, places`
	`    public int partition(int arr[], int low, int high) {`
	
	`        int pivot = low;`
	`        low++;`
	`        int comparisons = 0;`
	`        int swaps = 0;`
	`        System.out.print("\n-------------------------------------------------------------");`
	`        System.out.print("\nPivot: " + arr[high] + " Low: " + low + " High: " + high);`
	`        System.out.print("\nBefore Swaps-> Array: " + printArray(arr));`
	`        do {`
	`            // increment low pointer until that element is larger than the pivot element`
	`            while (low < high && arr[low] <= arr[pivot]) {`
	`                low++;`
	`                comparisons++;`
	`            }`
	`            // decrement high pointer until that element is smaller than the pivot element`
	`            while (high > pivot && arr[high] > arr[pivot]) {`
	`                high--;`
	`                comparisons++;`
	`            }`

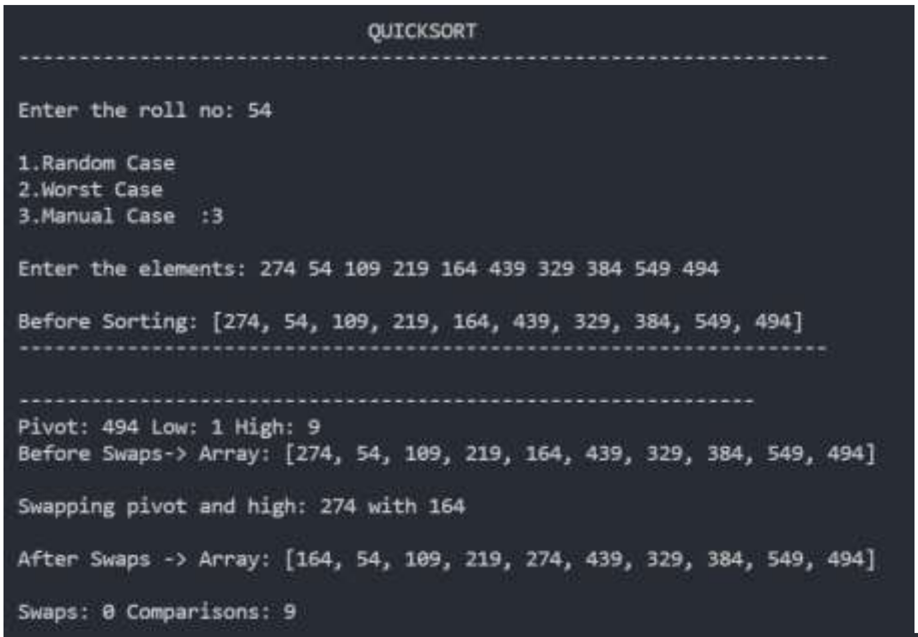
```
          // if the low and high pointers cross each other swap
the corresponding elements
          if (low < high) {
             int temp = arr[low];
             arr[low] = arr[high];
             arr[high] = temp;
             swaps++;

          }
       } while (low < high);
       // swap the pivot element and the element pointed by
high
       System.out.print("\n\nSwapping pivot and high: " +
arr[pivot] + " with " + arr[high]);
       int temp = arr[high];
       arr[high] = arr[pivot];
       arr[pivot] = temp;
       totalComparisons += comparisons;
       totalSwaps += swaps;
       System.out.println("\n\nAfter Swaps -> Array: " +
printArray(arr));
       System.out.print("\nSwaps: " + swaps + "
Comparisons: " + comparisons);
       return high;
    }

    // swap function
    public void swap(int arr[], int i, int j) {
       int temp = arr[i];
       arr[i] = arr[j];
       arr[j] = temp;
    }

    public void sort(int arr[], int low, int high) {
       if (low < high) {
          /*
           * pi is partitioning index, arr[pi] is
           * now at right place
           */
```

```
            int pi = partition(arr, low, high);

            // Recursively sort elements before
            // partition and after partition
            sort(arr, low, pi - 1);
            sort(arr, pi + 1, high);
        }
    }

    /* A utility function to print array of size n */
    public String printArray(int arr[]) {
        return Arrays.toString(arr);
    }
}

Main Class:
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;

public class Driver {
    public static void main(String[] args) {
        // User Input
        Scanner input = new Scanner(System.in);
        System.out.print("\n                              QUICKSORT");
        System.out.print("\n--------------------------------------
---------------------------\n");
        System.out.print("\nEnter the roll no: ");
        int rollNo = input.nextInt();

        int array[];
        ArrayList<Integer> list = new ArrayList<Integer>();

        // Case input
        System.out.print("\n1.Random Case\n2.Worst
Case\n3.Manual Case  :");
        int choice = input.nextInt();
        if (choice == 2) {
            // Worst Case
```

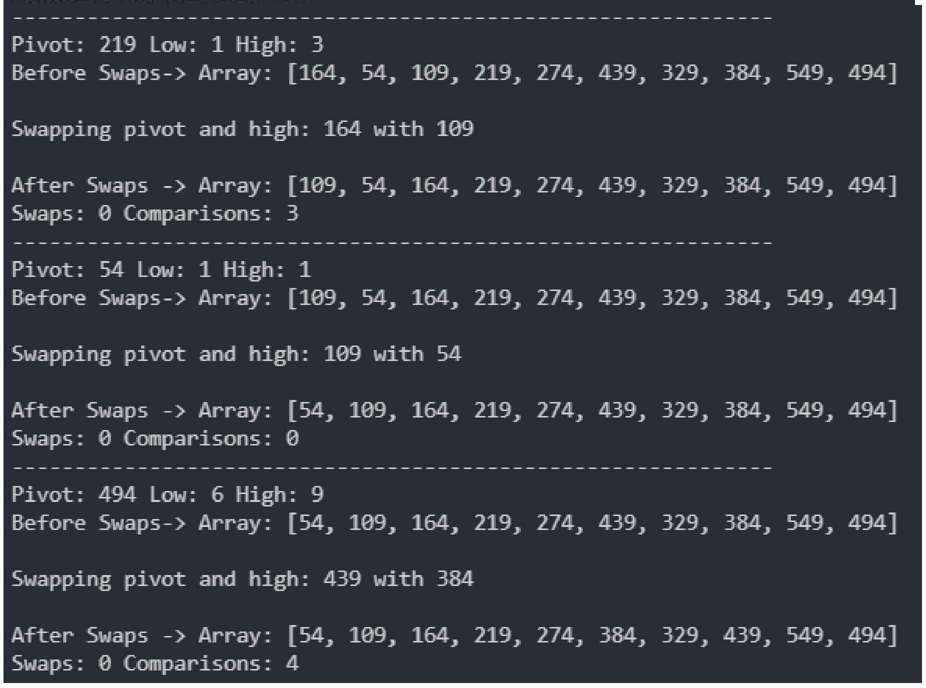
```
          for (int i = 9; i >= 0; i--)
             list.add(rollNo + (rollNo + 1) * i);

       } else if (choice == 1) {
          // Random Case
          for (int i = 0; i < 10; i++)
             list.add(rollNo + (rollNo + 1) * i);

          Collections.shuffle(list);
       } else {
          // Manual Case
          System.out.print("\nEnter the elements: ");
          for (int i = 0; i < 10; i++)
             list.add(input.nextInt());
       }
       array = new int[10];
       for (int i = 0; i < 10; i++)
          array[i] = list.get(i);
       quickSort qSort = new quickSort();
       System.out.print("\nBefore Sorting: " +
qSort.printArray(array));
       System.out.print("\n---------------------------------------
---------------------------\n");
       qSort.sort(array, 0, array.length - 1);
       System.out.print("\n---------------------------------------
---------------------------\n");
       System.out.print("\nAfter Sorting: " +
qSort.printArray(array));
       System.out.print("\n---------------------------------------
---------------------------\n");
       System.out.print("\nTotal swaps: " +
qSort.totalSwaps);
       System.out.print("\nTotal comparisons: " +
qSort.totalComparisons);
       input.close();
    }
}
```

OUTPUT:

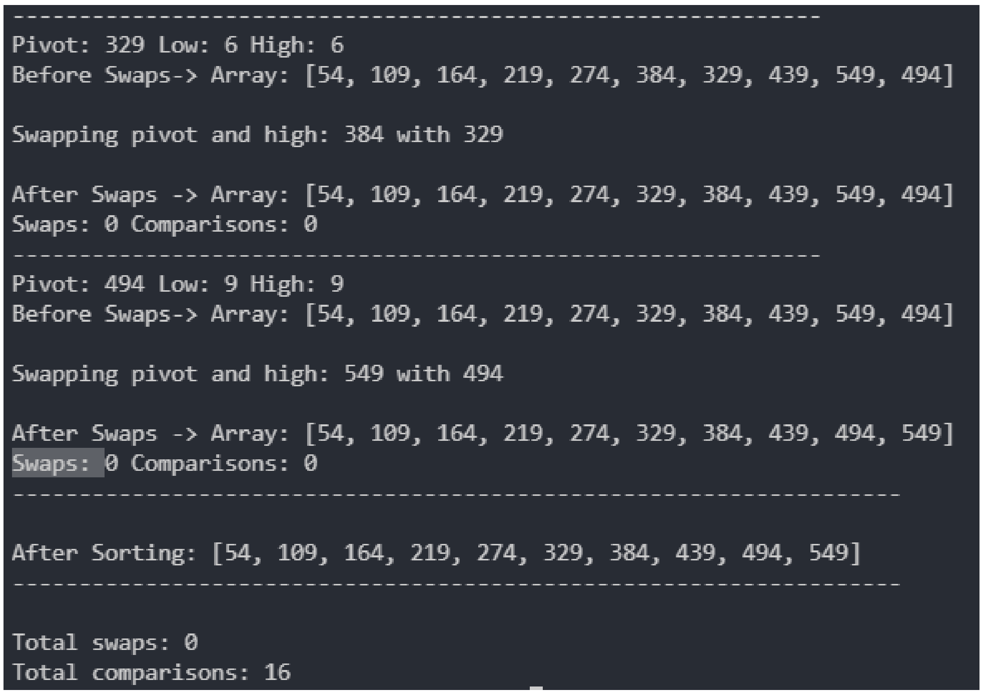
Best case



```
                        QUICKSORT
------------------------------------------------------------------

Enter the roll no: 54

1.Random Case
2.Worst Case
3.Manual Case  :3

Enter the elements: 274 54 109 219 164 439 329 384 549 494

Before Sorting: [274, 54, 109, 219, 164, 439, 329, 384, 549, 494]
------------------------------------------------------------------

------------------------------------------------------------
Pivot: 494 Low: 1 High: 9
Before Swaps-> Array: [274, 54, 109, 219, 164, 439, 329, 384, 549, 494]

Swapping pivot and high: 274 with 164

After Swaps -> Array: [164, 54, 109, 219, 274, 439, 329, 384, 549, 494]

Swaps: 0 Comparisons: 9
```



```
------------------------------------------------------------
Pivot: 219 Low: 1 High: 3
Before Swaps-> Array: [164, 54, 109, 219, 274, 439, 329, 384, 549, 494]

Swapping pivot and high: 164 with 109

After Swaps -> Array: [109, 54, 164, 219, 274, 439, 329, 384, 549, 494]
Swaps: 0 Comparisons: 3
------------------------------------------------------------
Pivot: 54 Low: 1 High: 1
Before Swaps-> Array: [109, 54, 164, 219, 274, 439, 329, 384, 549, 494]

Swapping pivot and high: 109 with 54

After Swaps -> Array: [54, 109, 164, 219, 274, 439, 329, 384, 549, 494]
Swaps: 0 Comparisons: 0
------------------------------------------------------------
Pivot: 494 Low: 6 High: 9
Before Swaps-> Array: [54, 109, 164, 219, 274, 439, 329, 384, 549, 494]

Swapping pivot and high: 439 with 384

After Swaps -> Array: [54, 109, 164, 219, 274, 384, 329, 439, 549, 494]
Swaps: 0 Comparisons: 4
```



```
-------------------------------------------------------------
Pivot: 329 Low: 6 High: 6
Before Swaps-> Array: [54, 109, 164, 219, 274, 384, 329, 439, 549, 494]

Swapping pivot and high: 384 with 329

After Swaps -> Array: [54, 109, 164, 219, 274, 329, 384, 439, 549, 494]
Swaps: 0 Comparisons: 0
-------------------------------------------------------------
Pivot: 494 Low: 9 High: 9
Before Swaps-> Array: [54, 109, 164, 219, 274, 329, 384, 439, 549, 494]

Swapping pivot and high: 549 with 494

After Swaps -> Array: [54, 109, 164, 219, 274, 329, 384, 439, 494, 549]
Swaps: 0 Comparisons: 0
-------------------------------------------------------------------

After Sorting: [54, 109, 164, 219, 274, 329, 384, 439, 494, 549]
-------------------------------------------------------------------

Total swaps: 0
Total comparisons: 16
```

Time Complexity of Quick sort

- **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:

Bharatiya Vidya Bhavan’s

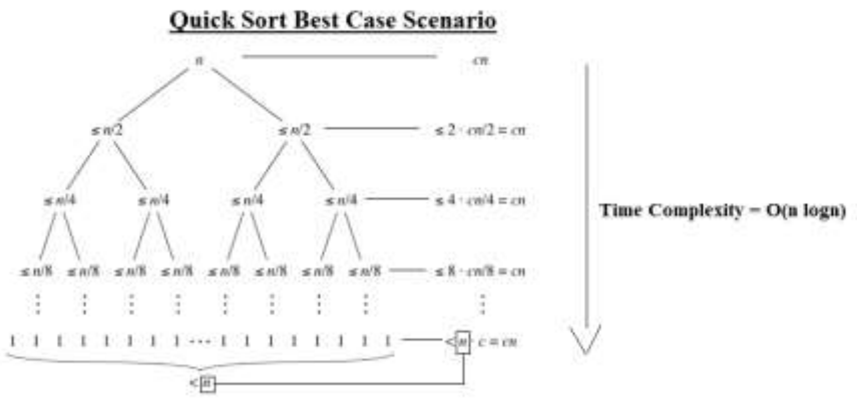
Sardar Patel Institute of Technology

Bhavan’s Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

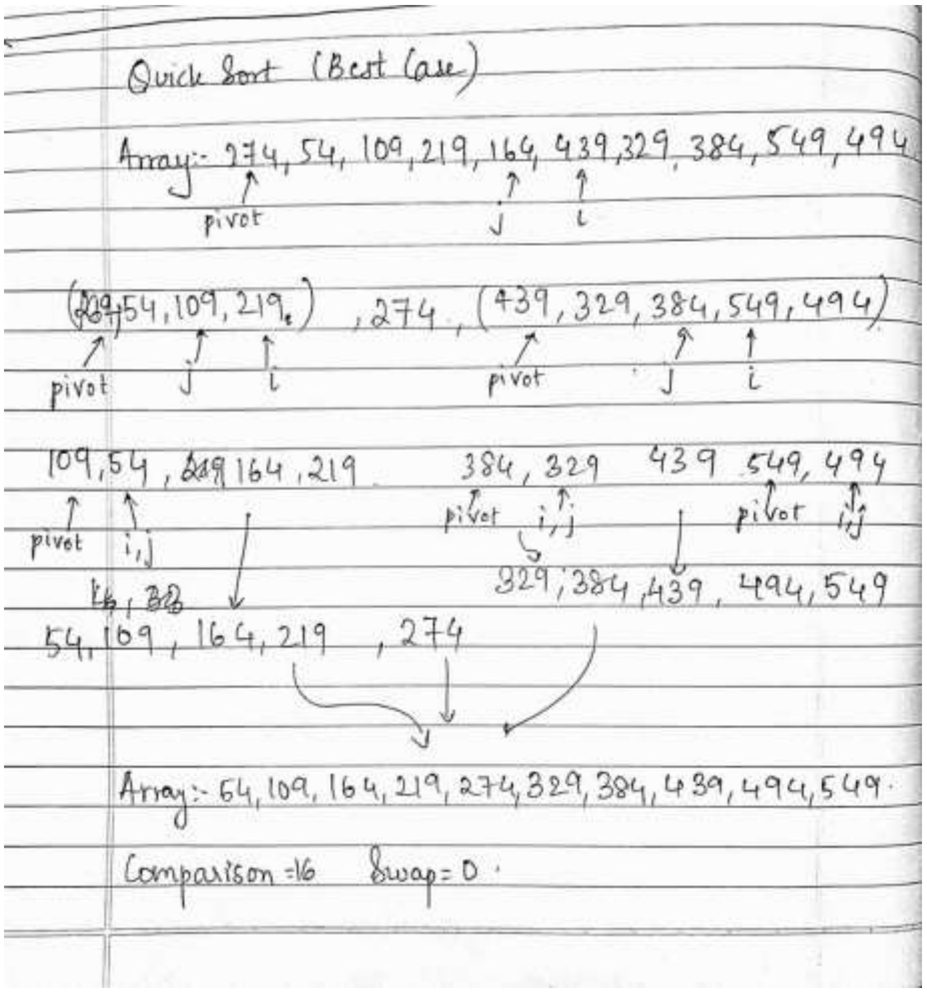
Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA



The best-case complexity of the quick sort algorithm is O(n logn)



Quick Sort (Best Case)

Array:- 274, 54, 109, 219, 164, 439, 329, 384, 549, 494
pivot j i

(164, 54, 109, 219) , 274 , (439, 329, 384, 549, 494)
pivot j i pivot j i

109, 54, 164, 219 384, 329 439 549, 494
pivot i, j pivot i, j pivot i, j

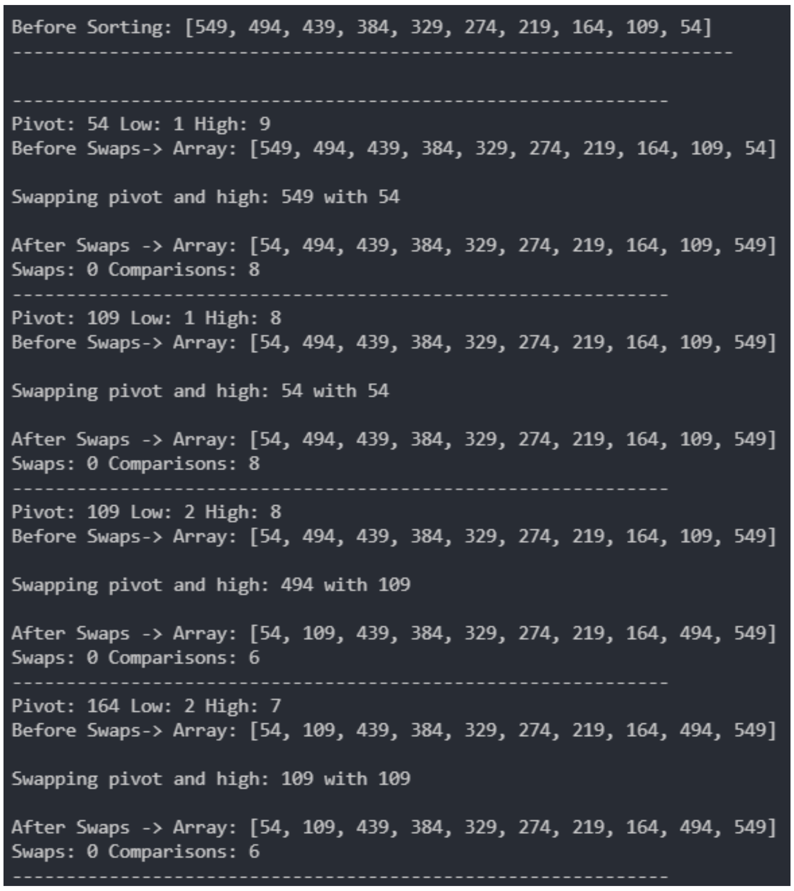
54, 109 329, 384, 439, 494, 549

54, 109, 164, 219 , 274

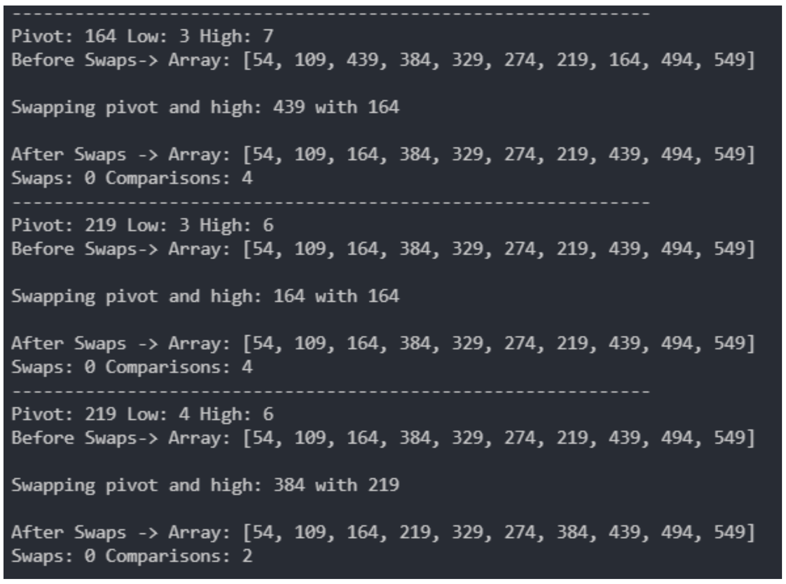
Array:- 54, 109, 164, 219, 274, 329, 384, 439, 494, 549.

Comparison =16 Swap= 0.

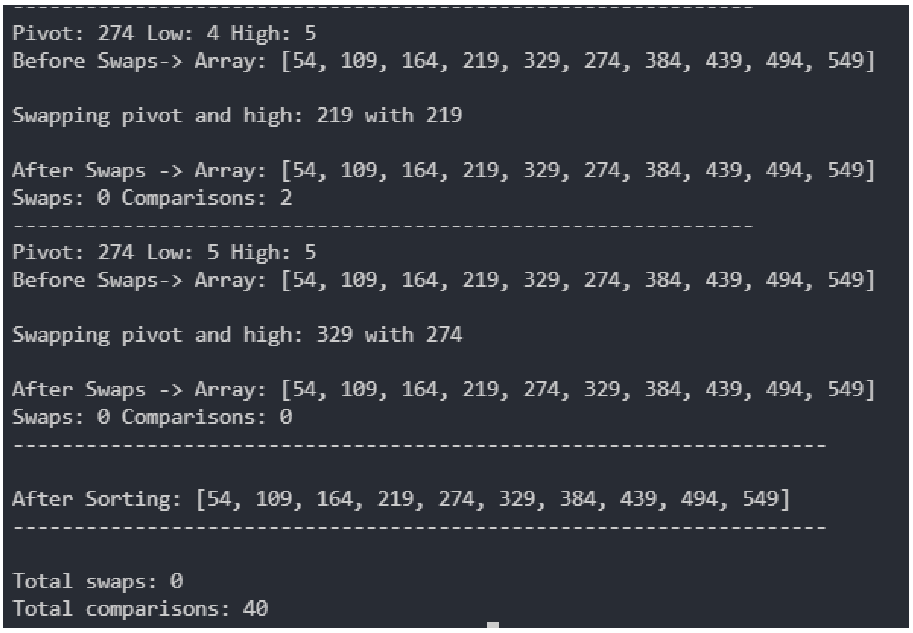
Worst Case:



```
Before Sorting: [549, 494, 439, 384, 329, 274, 219, 164, 109, 54]
-------------------------------------------------------------------

-------------------------------------------------------------
Pivot: 54 Low: 1 High: 9
Before Swaps-> Array: [549, 494, 439, 384, 329, 274, 219, 164, 109, 54]

Swapping pivot and high: 549 with 54

After Swaps -> Array: [54, 494, 439, 384, 329, 274, 219, 164, 109, 549]
Swaps: 0 Comparisons: 8
-------------------------------------------------------------
Pivot: 109 Low: 1 High: 8
Before Swaps-> Array: [54, 494, 439, 384, 329, 274, 219, 164, 109, 549]

Swapping pivot and high: 54 with 54

After Swaps -> Array: [54, 494, 439, 384, 329, 274, 219, 164, 109, 549]
Swaps: 0 Comparisons: 8
-------------------------------------------------------------
Pivot: 109 Low: 2 High: 8
Before Swaps-> Array: [54, 494, 439, 384, 329, 274, 219, 164, 109, 549]

Swapping pivot and high: 494 with 109

After Swaps -> Array: [54, 109, 439, 384, 329, 274, 219, 164, 494, 549]
Swaps: 0 Comparisons: 6
-------------------------------------------------------------
Pivot: 164 Low: 2 High: 7
Before Swaps-> Array: [54, 109, 439, 384, 329, 274, 219, 164, 494, 549]

Swapping pivot and high: 109 with 109

After Swaps -> Array: [54, 109, 439, 384, 329, 274, 219, 164, 494, 549]
Swaps: 0 Comparisons: 6
-------------------------------------------------------------
```

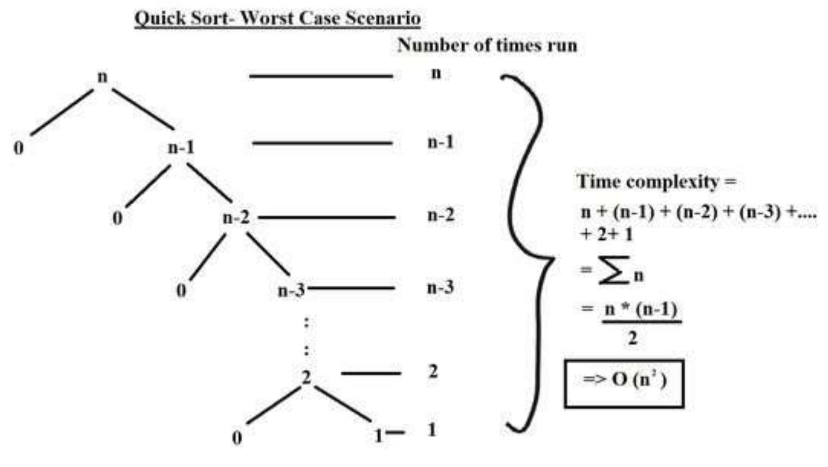


```
-------------------------------------------------------------
Pivot: 164 Low: 3 High: 7
Before Swaps-> Array: [54, 109, 439, 384, 329, 274, 219, 164, 494, 549]

Swapping pivot and high: 439 with 164

After Swaps -> Array: [54, 109, 164, 384, 329, 274, 219, 439, 494, 549]
Swaps: 0 Comparisons: 4
-------------------------------------------------------------
Pivot: 219 Low: 3 High: 6
Before Swaps-> Array: [54, 109, 164, 384, 329, 274, 219, 439, 494, 549]

Swapping pivot and high: 164 with 164

After Swaps -> Array: [54, 109, 164, 384, 329, 274, 219, 439, 494, 549]
Swaps: 0 Comparisons: 4
-------------------------------------------------------------
Pivot: 219 Low: 4 High: 6
Before Swaps-> Array: [54, 109, 164, 384, 329, 274, 219, 439, 494, 549]

Swapping pivot and high: 384 with 219

After Swaps -> Array: [54, 109, 164, 219, 329, 274, 384, 439, 494, 549]
Swaps: 0 Comparisons: 2
```



```
-------------------------------------------------------------
Pivot: 274 Low: 4 High: 5
Before Swaps-> Array: [54, 109, 164, 219, 329, 274, 384, 439, 494, 549]

Swapping pivot and high: 219 with 219

After Swaps -> Array: [54, 109, 164, 219, 329, 274, 384, 439, 494, 549]
Swaps: 0 Comparisons: 2
-------------------------------------------------------------
Pivot: 274 Low: 5 High: 5
Before Swaps-> Array: [54, 109, 164, 219, 329, 274, 384, 439, 494, 549]

Swapping pivot and high: 329 with 274

After Swaps -> Array: [54, 109, 164, 219, 274, 329, 384, 439, 494, 549]
Swaps: 0 Comparisons: 0
-------------------------------------------------------------------

After Sorting: [54, 109, 164, 219, 274, 329, 384, 439, 494, 549]
-------------------------------------------------------------------

Total swaps: 0
Total comparisons: 40
```

- **Worst case scenario:** This happens when we encounter the most unbalanced partitions possible, then the original call takes n time, the recursive call on n-1 elements will take (n-1) time, the recursive call on (n-2) elements will take (n-2) time, and so on. The worst case time complexity of Quick Sort would be **O(n2)**.



Bharatiya Vidya Bhavan's

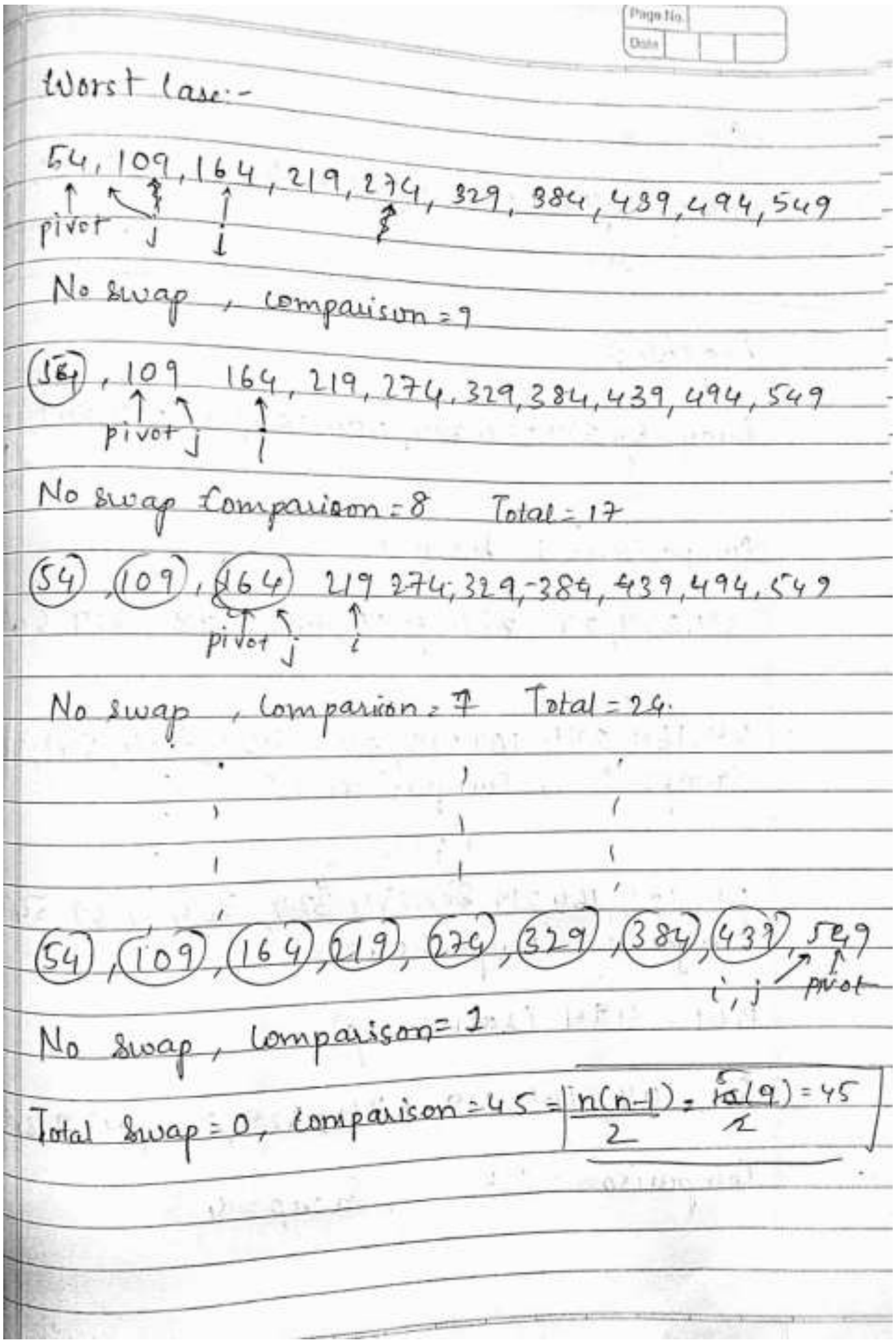
Sardar Patel Institute of Technology

Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department & Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Page No.
Date

Worst Case:-

54, 109, 164, 219, 274, 329, 384, 439, 494, 549

pivot j i

No swap , comparison = 9

54, 109 164, 219, 274, 329, 384, 439, 494, 549

pivot j i

No swap Comparison = 8 Total = 17

54, 109, 164 219 274, 329, 384, 439, 494, 549

pivot j i

No swap , Comparison = 7 Total = 24.

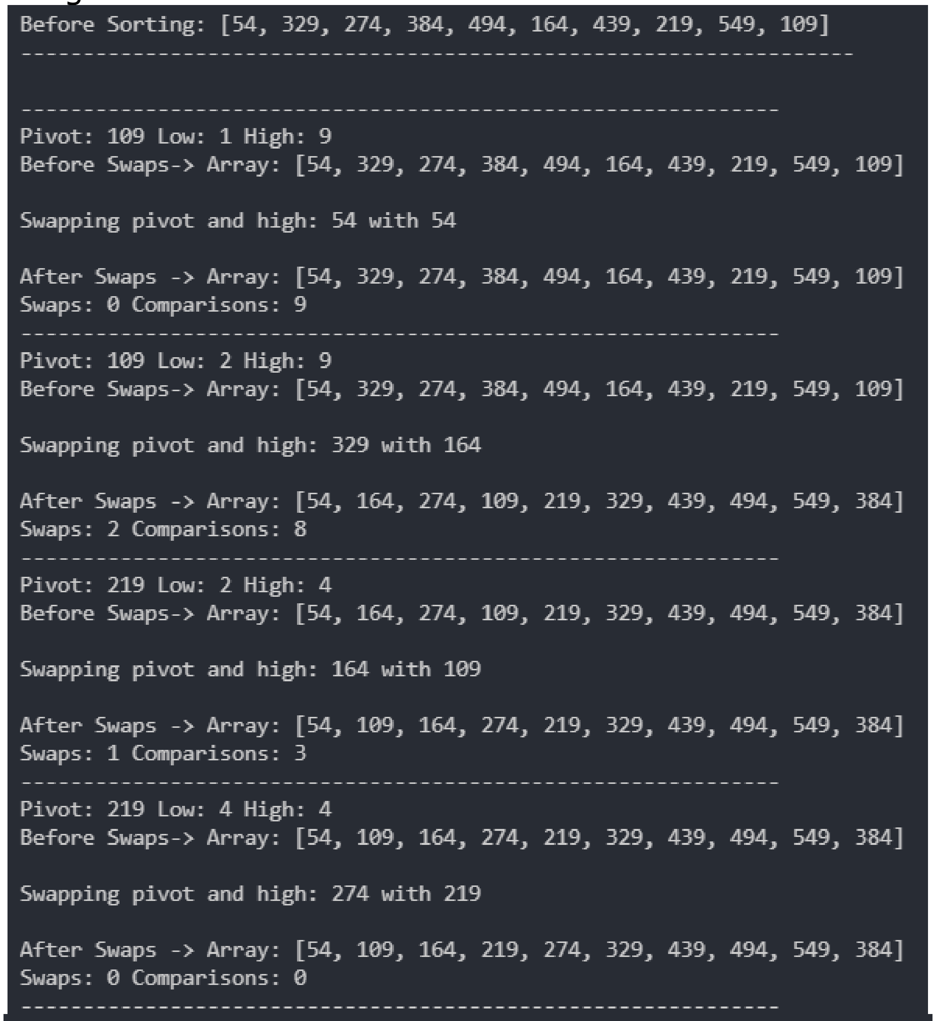
54, 109, 164, 219, 274, 329, 384, 439, 549

i, j pivot

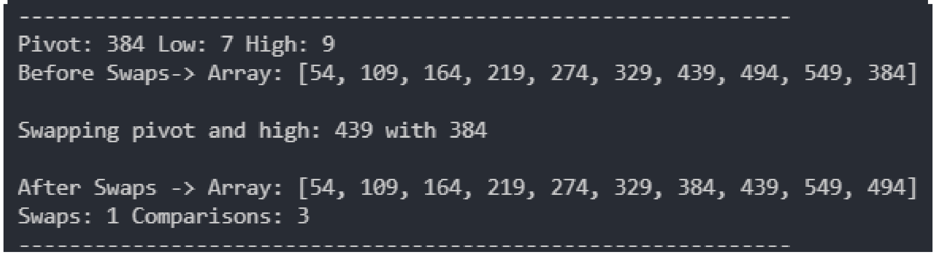
No swap, Comparison = 1

Total swap = 0, comparison = 45 = $\frac{n(n-1)}{2} = \frac{10(9)}{2} = 45$

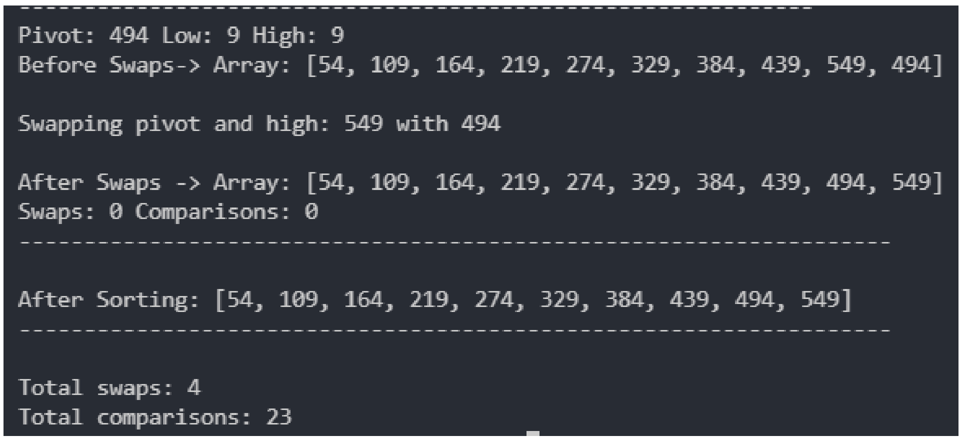
Average Case:



```
Before Sorting: [54, 329, 274, 384, 494, 164, 439, 219, 549, 109]
-------------------------------------------------------------------

------------------------------------------------------------
Pivot: 109 Low: 1 High: 9
Before Swaps-> Array: [54, 329, 274, 384, 494, 164, 439, 219, 549, 109]

Swapping pivot and high: 54 with 54

After Swaps -> Array: [54, 329, 274, 384, 494, 164, 439, 219, 549, 109]
Swaps: 0 Comparisons: 9
------------------------------------------------------------
Pivot: 109 Low: 2 High: 9
Before Swaps-> Array: [54, 329, 274, 384, 494, 164, 439, 219, 549, 109]

Swapping pivot and high: 329 with 164

After Swaps -> Array: [54, 164, 274, 109, 219, 329, 439, 494, 549, 384]
Swaps: 2 Comparisons: 8
------------------------------------------------------------
Pivot: 219 Low: 2 High: 4
Before Swaps-> Array: [54, 164, 274, 109, 219, 329, 439, 494, 549, 384]

Swapping pivot and high: 164 with 109

After Swaps -> Array: [54, 109, 164, 274, 219, 329, 439, 494, 549, 384]
Swaps: 1 Comparisons: 3
------------------------------------------------------------
Pivot: 219 Low: 4 High: 4
Before Swaps-> Array: [54, 109, 164, 274, 219, 329, 439, 494, 549, 384]

Swapping pivot and high: 274 with 219

After Swaps -> Array: [54, 109, 164, 219, 274, 329, 439, 494, 549, 384]
Swaps: 0 Comparisons: 0
------------------------------------------------------------
```

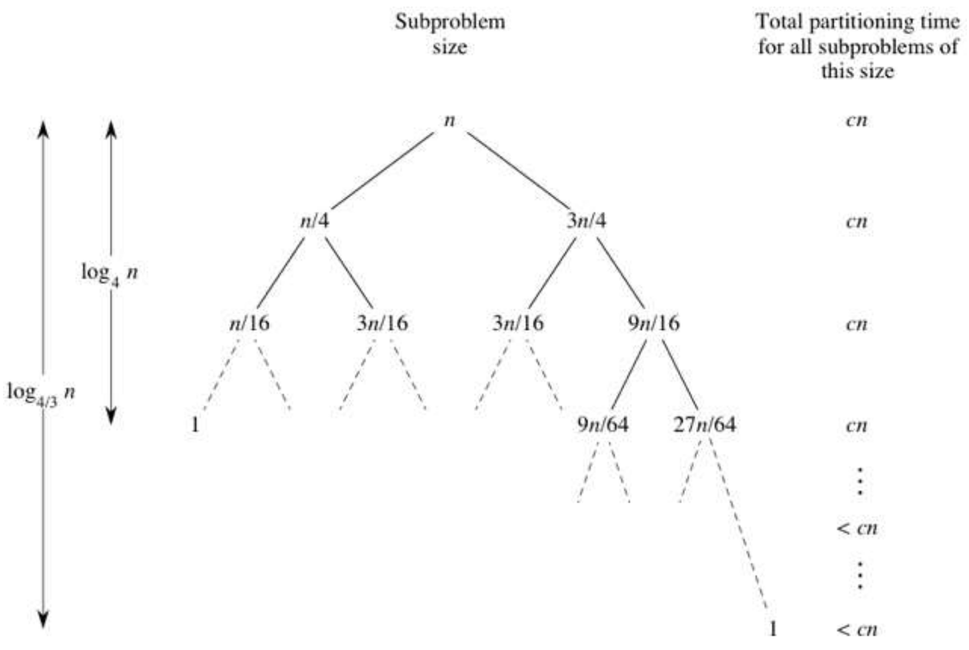


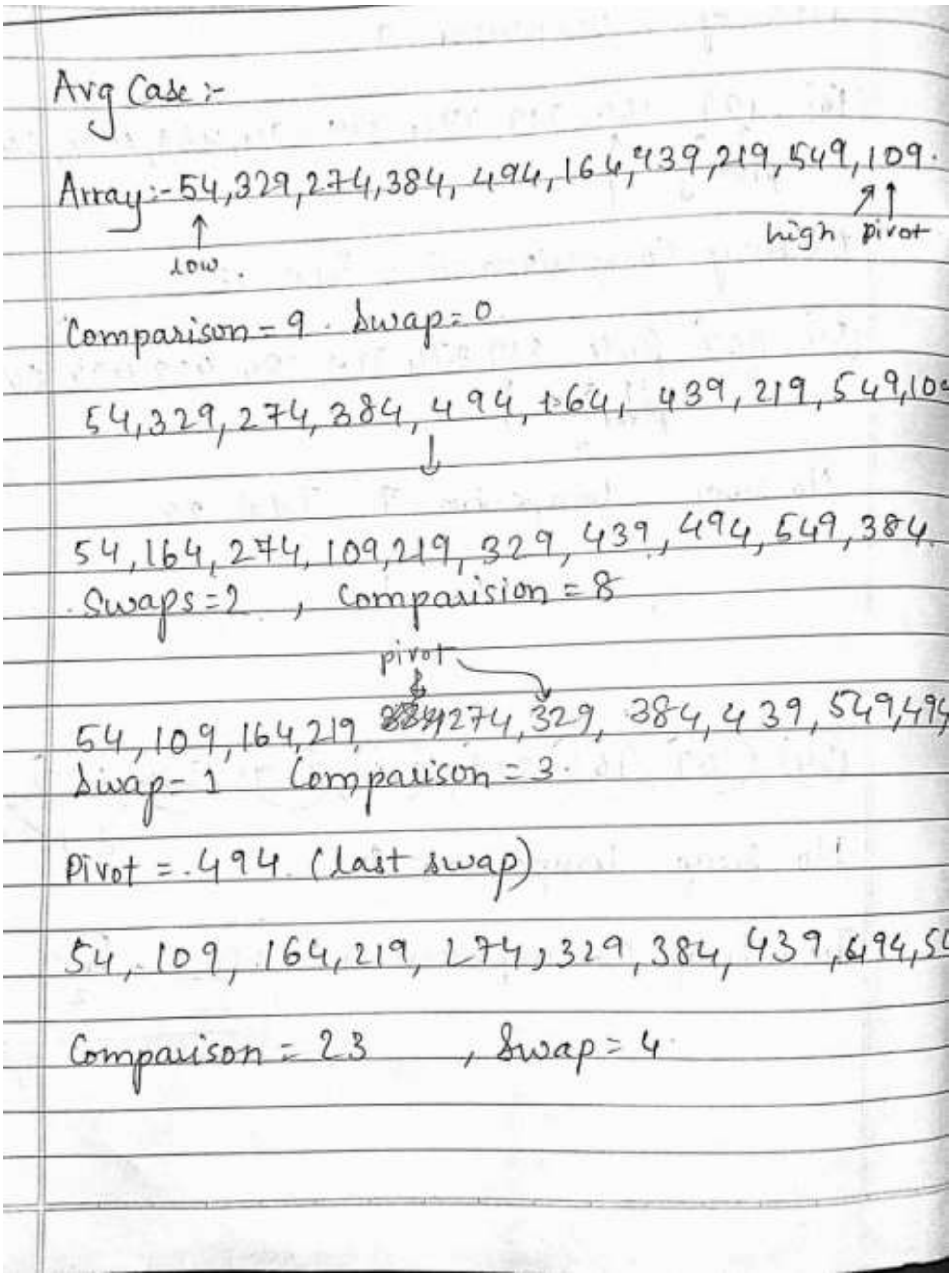
```
------------------------------------------------------------
Pivot: 384 Low: 7 High: 9
Before Swaps-> Array: [54, 109, 164, 219, 274, 329, 439, 494, 549, 384]

Swapping pivot and high: 439 with 384

After Swaps -> Array: [54, 109, 164, 219, 274, 329, 384, 439, 549, 494]
Swaps: 1 Comparisons: 3
------------------------------------------------------------
```



```
------------------------------------------------------------
Pivot: 494 Low: 9 High: 9
Before Swaps-> Array: [54, 109, 164, 219, 274, 329, 384, 439, 549, 494]

Swapping pivot and high: 549 with 494

After Swaps -> Array: [54, 109, 164, 219, 274, 329, 384, 439, 494, 549]
Swaps: 0 Comparisons: 0
------------------------------------------------------------------

After Sorting: [54, 109, 164, 219, 274, 329, 384, 439, 494, 549]
------------------------------------------------------------------

Total swaps: 4
Total comparisons: 23
```

Average Case: The average case run time of quick sort is O(n logn) . This case happens when we dont exactly get evenly balanced partitions. We might get at worst a 3-to-1 split on either side of pivot element.





Avg Case :-

Array :- 54,329,274,384,494,164,439,219,549,109.

low high pivot

Comparison = 9 . Swap = 0

54,329,274,384,494,164, 439,219,549,10

↓

54,164,274,109,219,329,439,494,549,384.

Swaps = 2 , Comparision = 8

pivot

54,109,164,219, ~~384~~274,329, 384,439,549,49

Swap = 1 Comparison = 3.

Pivot = 494 (last swap)

54, 109, 164,219, 274,329,384,439,494,5

Comparison = 23 , Swap = 4

Bharatiya Vidya Bhavan's

Sardar Patel Institute of Technology

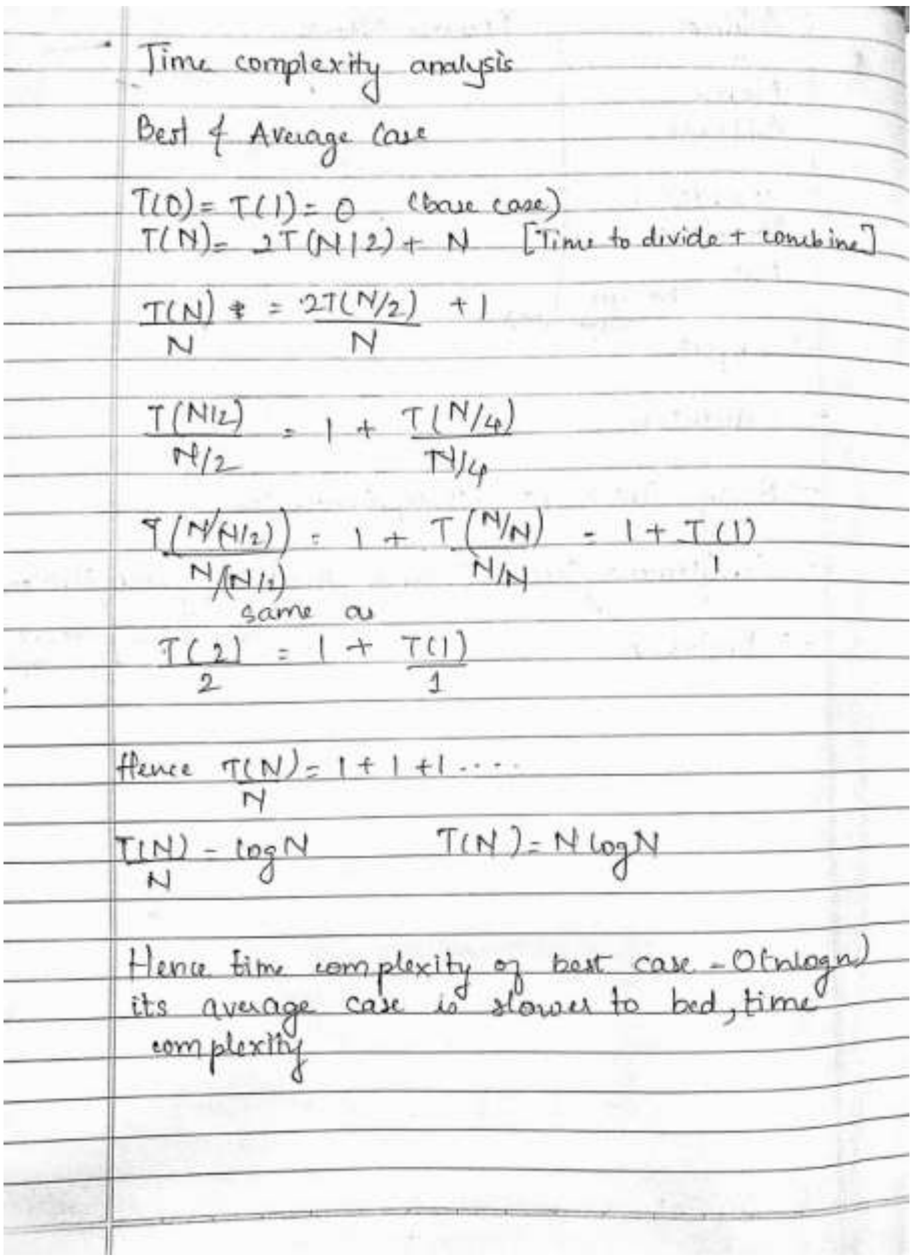
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

Best case and Average case Time complexity analysis



Time complexity analysis

Best & Average Case

$T(0) = T(1) = 0$ (base case)

$T(N) = 2T(N/2) + N$ [Time to divide + combine]

$\frac{T(N)}{N}$ ~~+~~ $= \frac{2T(N/2)}{N} + 1$

$\frac{T(N/2)}{N/2} = 1 + \frac{T(N/4)}{N/4}$

$\frac{T(N/(N/2))}{N/(N/2)} = 1 + \frac{T(N/N)}{N/N} = 1 + \frac{T(1)}{1}$

same as

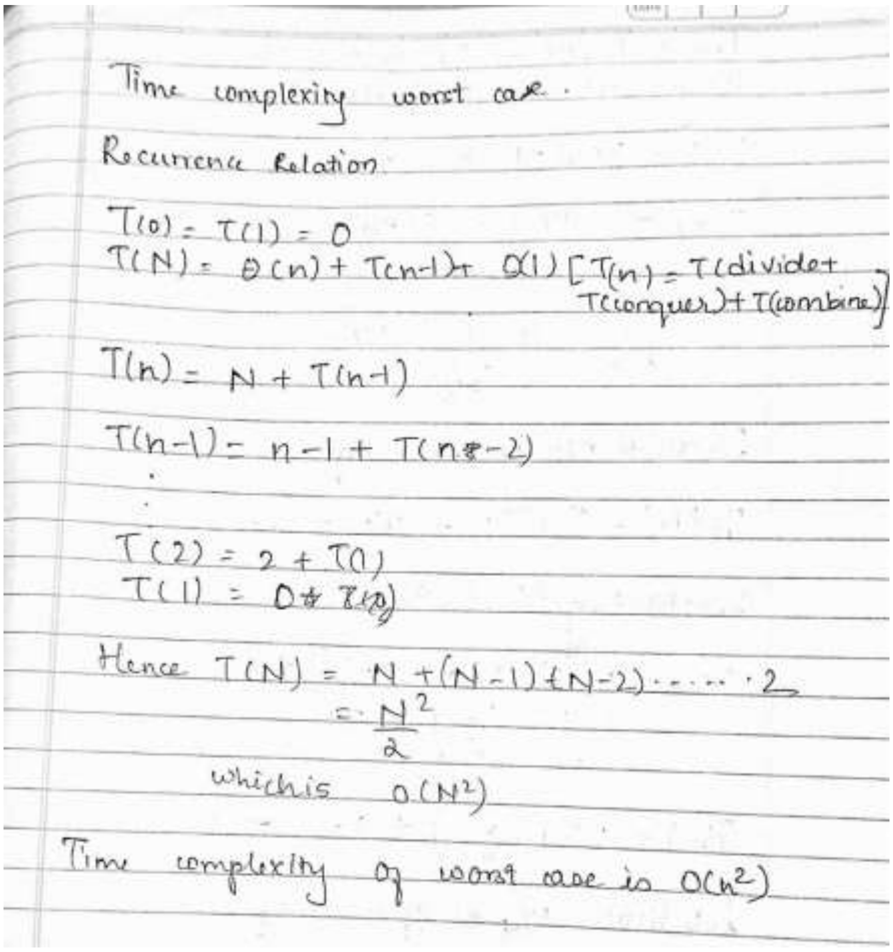
$\frac{T(2)}{2} = 1 + \frac{T(1)}{1}$

Hence $\frac{T(N)}{N} = 1 + 1 + 1 \dots$

$\frac{T(N)}{N} = \log N$ $\quad T(N) = N \log N$

Hence time complexity of best case - O(nlogn)
its average case is slower to bed, time complexity

Worst Case Time Complexity Analysis:



Time complexity worst case.

Recurrence Relation

$T(0) = T(1) = 0$

$T(N) = \Theta(n) + T(n-1) + \Theta(1)$ [$T(n) = T(\text{divide}+T(\text{conquer})+T(\text{combine})$]

$T(n) = N + T(n-1)$

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

.

.

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

$T(1) = 0 + T(0)$

Hence $T(N) = N + (N-1) + (N-2) \cdots\cdots 2$

$= \frac{N^2}{2}$

which is $O(N^2)$

Time complexity of worst case is $O(n^2)$

RESULT: Concepts gained from programming of problem:

- I have learnt about divide and conquer sorting algorithm like quicksort and mergesort
- Since the time complexity of following algo's are less than the O(n) which is O(nlogn) which makes them faster in case of large numbers of elements to sort as nlogn function has relatively constant slope at large values of N
- In quicksort the time complexity of any worst case is O(n2) while in mergesort its O(nlogn) for all cases
- I learnt how to use proper looping in order to avoid the ArrayOutOfBoundsException

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

Class: S.Y.B.Tech **Sem.:** 4 **Course:** DAA

EXPERIMENT (MergeSort)

CODE:

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Scanner;

public class mergeSortClass {
    // Merge two subarrays L and M into arr
    int shift = 0;

    void merge(int arr[], int p, int q, int r) {
        // Create temp array to store the merged subarrays
        int n1 = q - p + 1;
        int n2 = r - q;
        shift = 0;
        int L[] = new int[n1];
        int M[] = new int[n2];

        for (int i = 0; i < n1; i++)
            L[i] = arr[p + i];
        for (int j = 0; j < n2; j++)
            M[j] = arr[q + 1 + j];

        // Maintain current index of sub-arrays and main array
        int i, j, k;
        i = 0;
        j = 0;
        k = p;

        // Until we reach either end of either L or M, pick larger
among
        // elements L and M and place them in the correct
position at A[p..r]
        while (i < n1 && j < n2) {
            if (L[i] <= M[j]) {
                arr[k] = L[i];
                i++;
            } else {
                arr[k] = M[j];
                j++;
                shift++;
            }
            k++;
        }

        // When we run out of elements in either L or M,
        // pick up the remaining elements and put in A[p..r]
        while (i < n1) {
            arr[k] = L[i];
            i++;
            k++;
        }

        while (j < n2) {
            arr[k] = M[j];
            j++;
            k++;
        }
    }

    // Divide the array into two subarrays, sort them and
merge them
    void mergeSort(int arr[], int l, int r) {
        if (l < r) {
            System.out.print("\n-----------------------------------
-----------------");
            // m is the point where the array is divided into two
subarrays
            int m = (l + r) / 2;
            System.out.print("\nThe midposition is: " + m);

            System.out.print("\nLeft half: " +
Arrays.toString(Arrays.copyOfRange(arr, l, m)));
            mergeSort(arr, l, m);
```

	(code below)

```
        System.out.print("\nRight half: " +
printArray(Arrays.copyOfRange(arr, m + 1, arr.length)));
        mergeSort(arr, m + 1, r);
        System.out.print("\nTotal Shift: " + shift);
        merge(arr, l, m, r);
        System.out.print("\nMerged: " + printArray(arr));
        System.out.print("\n-----------------------------------
-----------------");
      }
    }

    /* A utility function to print array of size n */
    public String printArray(int arr[]) {
      return Arrays.toString(arr);
    }

    // Driver program
    public static void main(String args[]) {
      Scanner input = new Scanner(System.in);
      System.out.print("\n                              MERGESORT");
      System.out.print("\n-------------------------------------
---------------------------\n");
      System.out.print("\nEnter the roll no: ");
      int rollNo = input.nextInt();

      int array[];
      ArrayList<Integer> list = new ArrayList<Integer>();

      // Case input
      System.out.print("\n1.Random Case\n2.Worst
Case\n3.Manual Case  :");
      int choice = input.nextInt();
      if (choice == 2) {
        // Worst Case
        for (int i = 9; i >= 0; i--)
          list.add(rollNo + (rollNo + 1) * i);

      } else if (choice == 1) {
        // Random Case
```

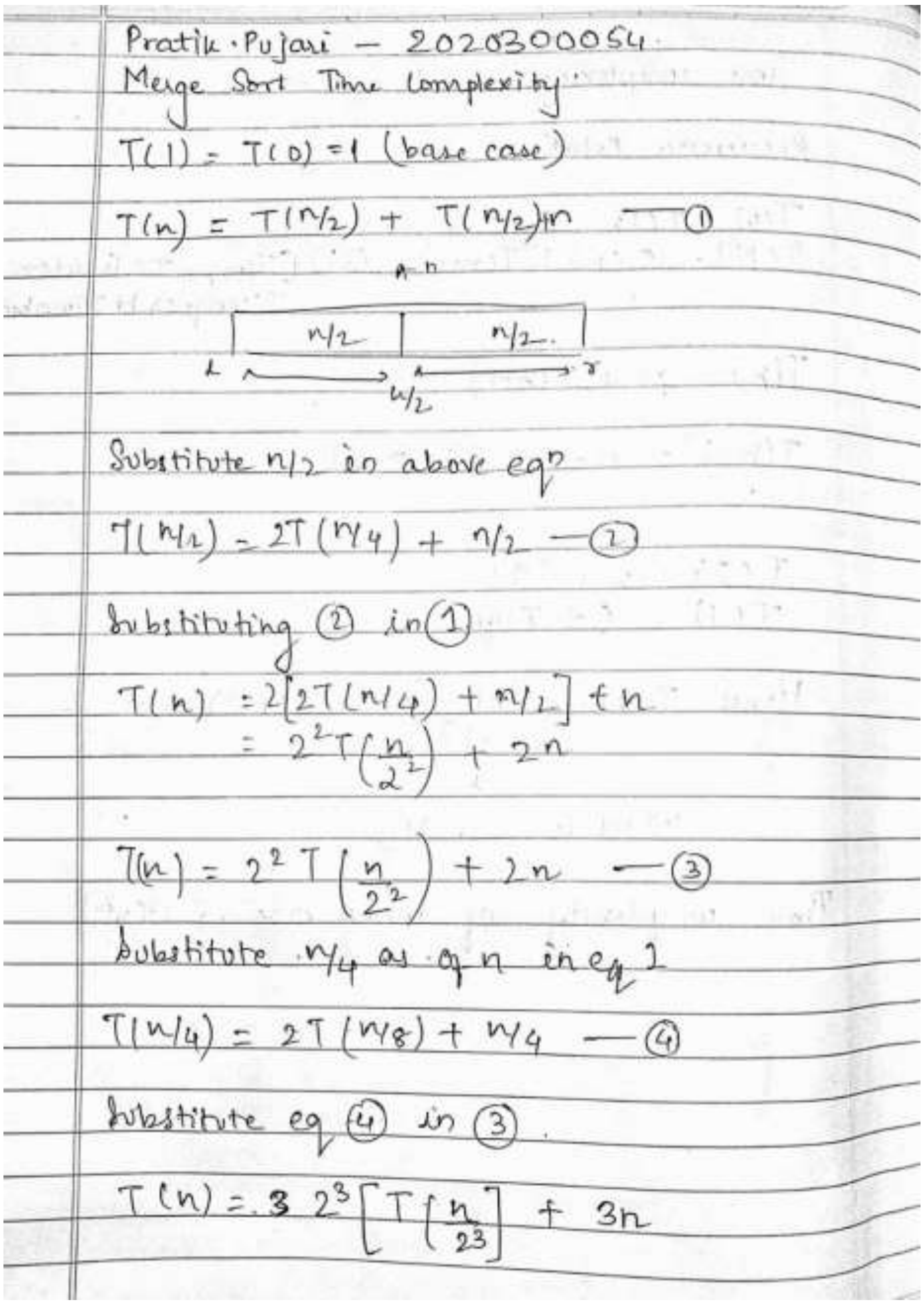
```
            for (int i = 0; i < 10; i++)
                list.add(rollNo + (rollNo + 1) * i);
            Collections.shuffle(list);
        } else {
            // Manual Case
            System.out.print("\nEnter the elements: ");
            for (int i = 0; i < 10; i++)
                list.add(input.nextInt());
        }
        array = new int[10];
        for (int i = 0; i < 10; i++)
            array[i] = list.get(i);
        mergeSortClass ob = new mergeSortClass();
        ob.mergeSort(array, 0, array.length - 1);
        System.out.println("\nSorted array:" +
ob.printArray(array));
        input.close();
    }
}
```

Pseudocode for MergeSort

- Declare left and right var which will mark the extreme indices of the array
- Left will be assigned to 0 and right will be assigned to n-1
- Find mid = (left+right)/2
- Call mergeSort on (left,mid) and (mid+1,rear)
- Above will continue till left<right
- Then we will call merge on the 2 subproblems

Merge sort Algorithm

```
MergeSort(arr, left, right):
    if left > right
        return
    mid = (left+right)/2
    mergeSort(arr, left, mid)
    mergeSort(arr, mid+1, right)
    merge(arr, left, mid, right)
end
```

OUTPUT:	

Pratik · Pujari — 2020300054

Merge Sort Time Complexity

$T(1) = T(0) = 1$ (base case)

$T(n) = T(n/2) + T(n/2) + n$ —①

n

$\boxed{n/2 \mid n/2}$

$l \longrightarrow \longrightarrow r$

$n/2$

Substitute $n/2$ in above eqn

$T(n/2) = 2T(n/4) + n/2$ —②

Substituting ② in ①

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

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

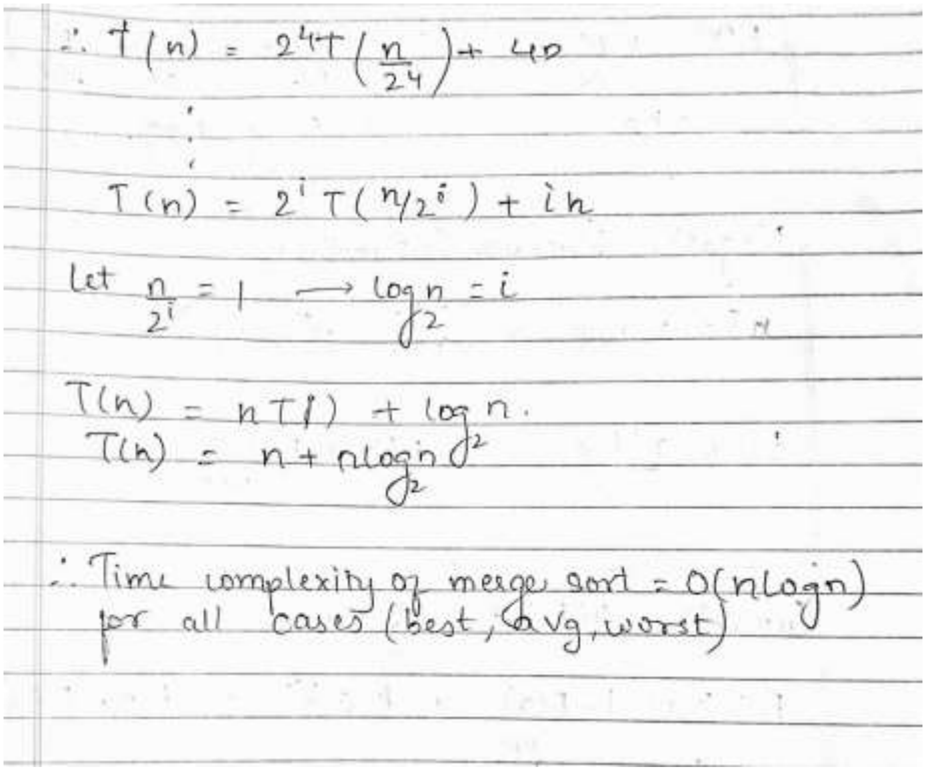
$T(n) = 2^2 T\left(\frac{n}{2^2}\right) + 2n$ —③

Substitute $n/4$ as n in eq 1

$T(n/4) = 2T(n/8) + n/4$ —④

Substitute eq ④ in ③

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



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

$\vdots$

$T(n) = 2^i T(n/2^i) + in$

Let $\frac{n}{2^i} = 1 \longrightarrow \log_2 n = i$

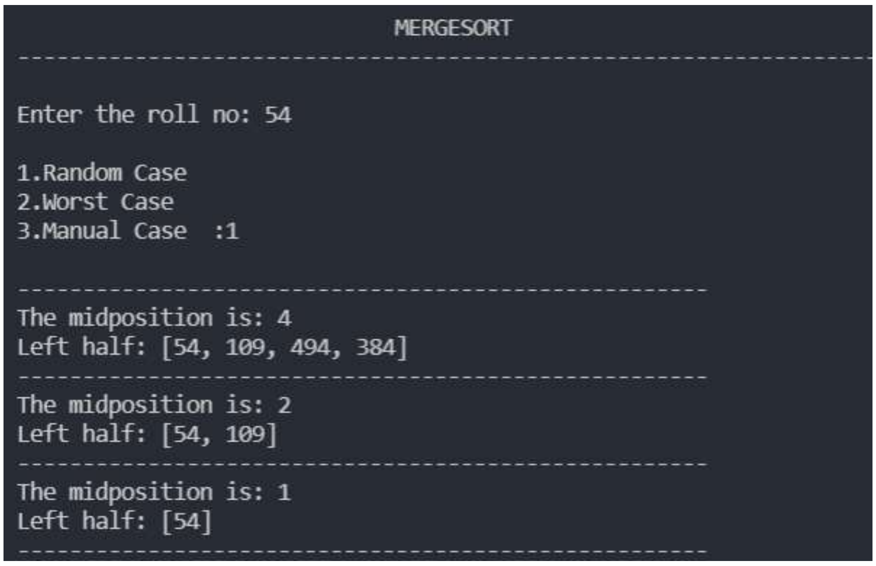
$T(n) = nT(1) + \log_2 n.$

$T(n) = n + n\log_2 n$

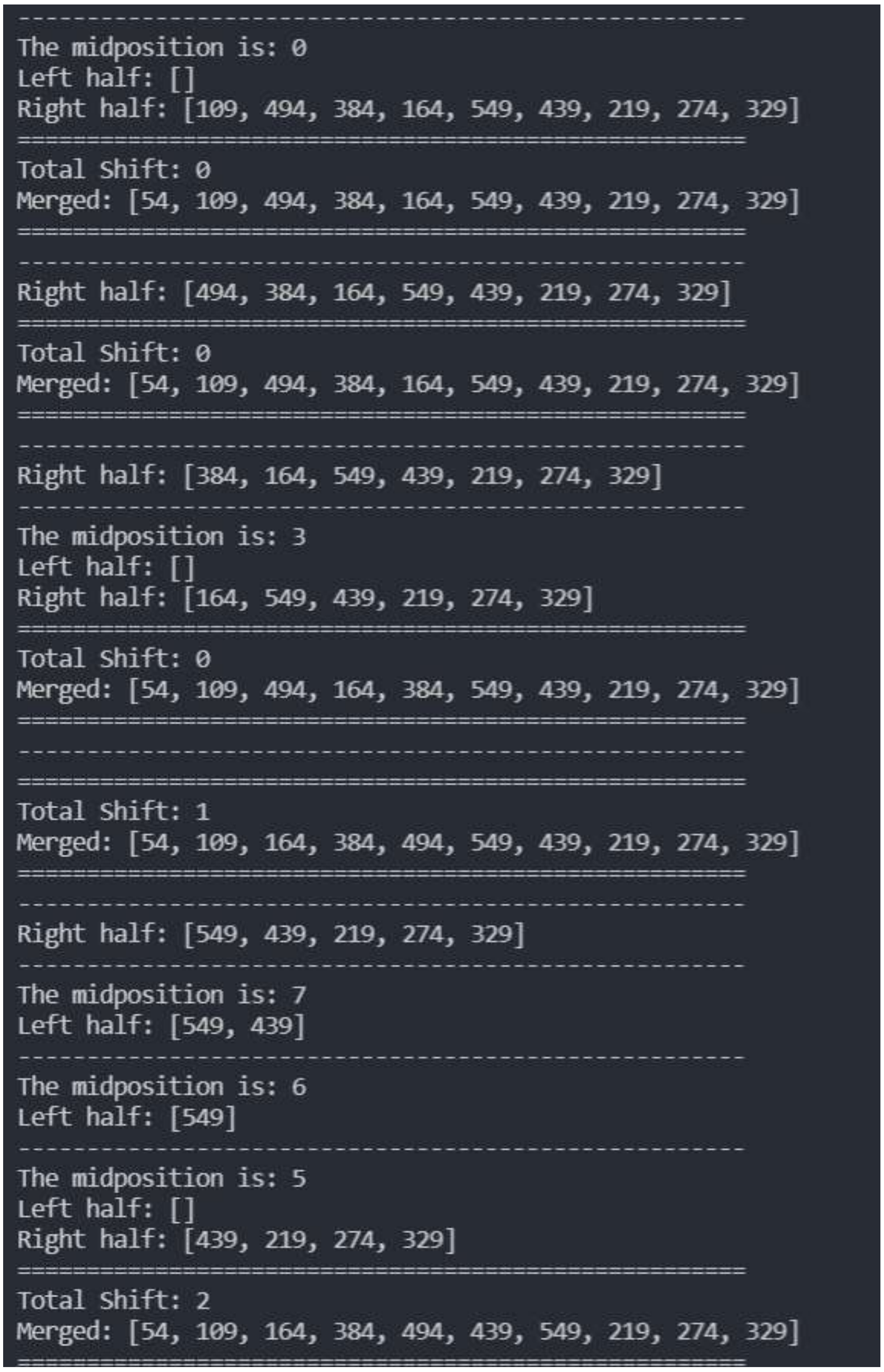
$\therefore$ Time complexity of merge sort $= O(n\log n)$ for all cases (best, avg, worst)

OUTPUT:

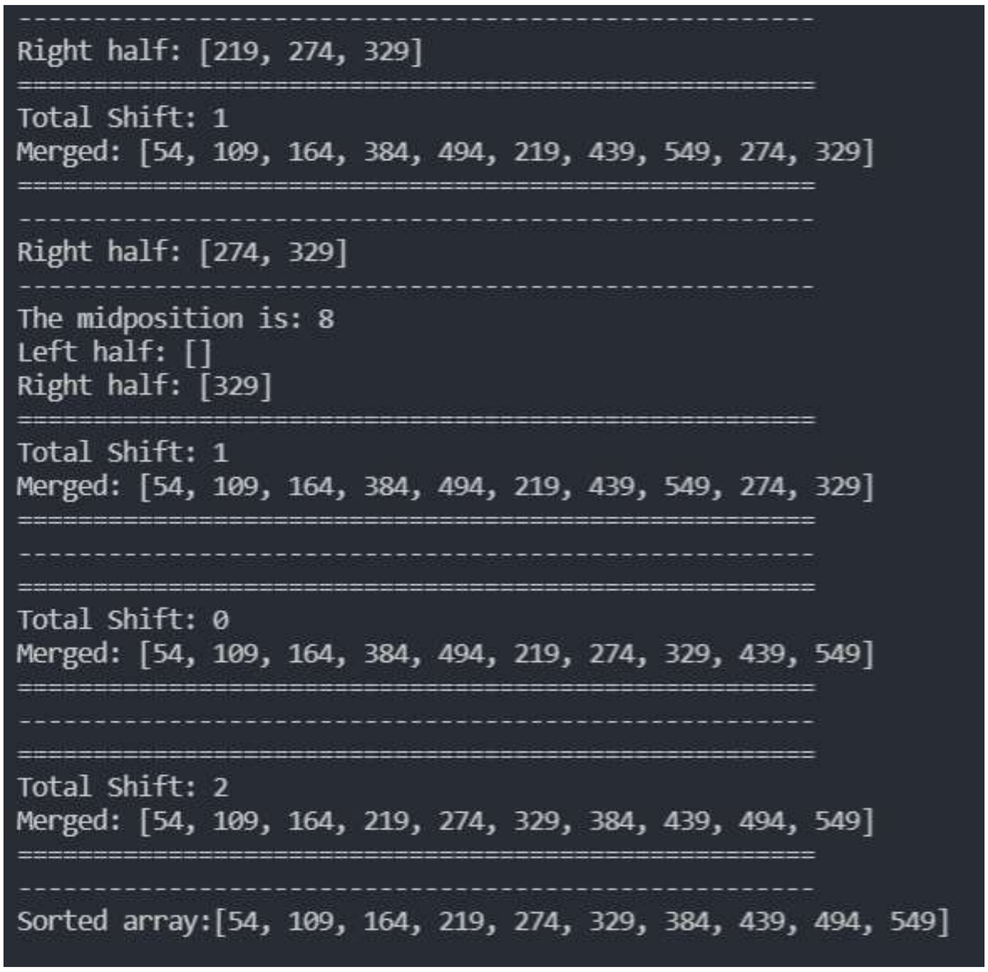
Random Case:



```
                            MERGESORT
------------------------------------------------------------------

Enter the roll no: 54

1.Random Case
2.Worst Case
3.Manual Case  :1

-----------------------------------------------------
The midposition is: 4
Left half: [54, 109, 494, 384]
-----------------------------------------------------
The midposition is: 2
Left half: [54, 109]
-----------------------------------------------------
The midposition is: 1
Left half: [54]
-----------------------------------------------------
```



```
-----------------------------------------------------
The midposition is: 0
Left half: []
Right half: [109, 494, 384, 164, 549, 439, 219, 274, 329]
=====================================================
Total Shift: 0
Merged: [54, 109, 494, 384, 164, 549, 439, 219, 274, 329]
=====================================================
-----------------------------------------------------
Right half: [494, 384, 164, 549, 439, 219, 274, 329]
=====================================================
Total Shift: 0
Merged: [54, 109, 494, 384, 164, 549, 439, 219, 274, 329]
=====================================================
-----------------------------------------------------
Right half: [384, 164, 549, 439, 219, 274, 329]
-----------------------------------------------------
The midposition is: 3
Left half: []
Right half: [164, 549, 439, 219, 274, 329]
=====================================================
Total Shift: 0
Merged: [54, 109, 494, 164, 384, 549, 439, 219, 274, 329]
=====================================================
-----------------------------------------------------
=====================================================
Total Shift: 1
Merged: [54, 109, 164, 384, 494, 549, 439, 219, 274, 329]
=====================================================
-----------------------------------------------------
Right half: [549, 439, 219, 274, 329]
-----------------------------------------------------
The midposition is: 7
Left half: [549, 439]
-----------------------------------------------------
The midposition is: 6
Left half: [549]
-----------------------------------------------------
The midposition is: 5
Left half: []
Right half: [439, 219, 274, 329]
=====================================================
Total Shift: 2
Merged: [54, 109, 164, 384, 494, 439, 549, 219, 274, 329]
=====================================================
```



```
-----------------------------------------------------
Right half: [219, 274, 329]
=====================================================
Total Shift: 1
Merged: [54, 109, 164, 384, 494, 219, 439, 549, 274, 329]
=====================================================
-----------------------------------------------------
Right half: [274, 329]
-----------------------------------------------------
The midposition is: 8
Left half: []
Right half: [329]
=====================================================
Total Shift: 1
Merged: [54, 109, 164, 384, 494, 219, 439, 549, 274, 329]
=====================================================
-----------------------------------------------------
=====================================================
Total Shift: 0
Merged: [54, 109, 164, 384, 494, 219, 274, 329, 439, 549]
=====================================================
-----------------------------------------------------
=====================================================
Total Shift: 2
Merged: [54, 109, 164, 219, 274, 329, 384, 439, 494, 549]
=====================================================
-----------------------------------------------------
Sorted array:[54, 109, 164, 219, 274, 329, 384, 439, 494, 549]
```