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.	3		

AIM:	To implement the Bubble sort algorithm using Recurrence Relations concepts.

THEORY:	**What is Recursion?**
	The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are Towers of Hanoi (TOH), Inorder/Preorder/Postorder Tree Traversals, DFS of Graph, etc.
	
	What is base condition in recursion?
	In the recursive program, the solution to the base case is provided and the solution of the bigger problem is expressed in terms of smaller problems.
	
	int fact(int n)
	{
	  if (n < = 1) // base case
	  return 1;
	  else
	  return n*fact(n-1);
	}
	In the above example, base case for n < = 1 is defined and larger value of number can be solved by converting to smaller one till base case is reached.
	
	What is the difference between direct and indirect recursion?
	A function fun is called direct recursive if it calls the same

function fun. A function fun is called indirect recursive if it calls another function say fun_new and fun_new calls fun directly or indirectly. Difference between direct and indirect recursion has been illustrated in Table 1.

```
// An example of direct recursion
void directRecFun()
{
   // Some code....
   directRecFun();
   // Some code...
}

// An example of indirect recursion
void indirectRecFun1()
{
   // Some code...
   indirectRecFun2();
   // Some code...
}
void indirectRecFun2()
{
   // Some code...
   indirectRecFun1();
   // Some code...
}
```

Recurrence Relation

A recurrence is an equation or inequality that describes a function in terms of its values on smaller inputs. To solve a Recurrence Relation means to obtain a function defined on the natural numbers that satisfy the recurrence.

For Example, the Worst Case Running Time T(n) of the MERGE SORT Procedures is described by the recurrence.

$T(n) = \theta(1)$ if $n=1$

$2T\left(\frac{n}{2}\right) + \theta(n)$ if $n>1$

What is Bubble Sort?

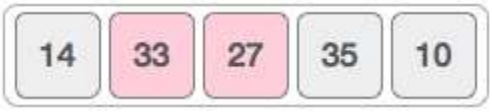
Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of $O(n^2)$ where **n** is the number of items.

How Bubble Sort Works?
We take an unsorted array for our example. Bubble sort takes $O(n^2)$ time so we're keeping it short and precise.

Bubble sort starts with very first two elements, comparing them to check which one is greater.

In this case, value 33 is greater than 14, so it is already in sorted locations. Next, we compare 33 with 27.

We find that 27 is smaller than 33 and these two values must be swapped.



The new array should look like this –

Next we compare 33 and 35. We find that both are in already sorted positions.

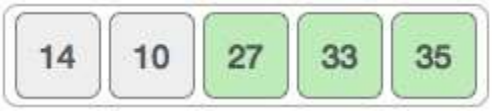
Then we move to the next two values, 35 and 10.

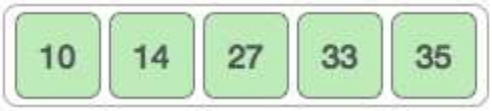
<table>
<tr>
<td></td>
<td>


We know then that 10 is smaller 35. Hence they are not sorted.


We swap these values. We find that we have reached the end of the array. After one iteration, the array should look like this –


To be precise, we are now showing how an array should look like after each iteration. After the second iteration, it should look like this –


Notice that after each iteration, at least one value moves at the end.


And when there's no swap required, bubble sorts learns that an array is completely sorted.


Now we should look into some practical aspects of bubble sort.
</td>
</tr>
<tr>
<td>PSEUDOCODE:</td>
<td>
<pre>begin BubbleSort(int arr[],int n):
    for i=0 to i< (n-1)
        if arr[i] is greater than arr[i+1]
            swap arr[i] and arr[i+1]

    if n-1 is greater than 1 return BubbleSort(arr,n-1)</pre>
</td>
</tr>
</table>

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:	Bubble Sort code:
	import java.util.Arrays;
	
	public class BubbleSort {
	
	int counter = 0;
	int swapNum = 0;
	static int maxPos = 0;
	String swaps = "Swaps: |";
	
	// swap function
	public static void swap(int[] arr, int i, int j) {
	int temp = arr[i];
	arr[i] = arr[j];
	arr[j] = temp;
	}
	
	// bubble sort function
	public void bubbleSortIteration(int arr[]) {
	int n = arr.length;
	System.out.printf("Iteration\tSwap\t\tArray\n");
	for (int i = 0; i < n - 1; i++) {
	for (int j = 0; j < n - i - 1; j++) {
	++counter;
	if (arr[j] > arr[j + 1]) {
	// swap arr[j+1] and arr[j]
	System.out.printf("%d\t\t%d<-
	>%d\t\t%s\n", counter, arr[j], arr[j + 1], printArr(arr));
	
	// swappging the elements
	int temp = arr[j];
	arr[j] = arr[j + 1];
	arr[j + 1] = temp;
	} else {
	System.out.printf("%d\t\t----\t\t%s\n",
	counter, printArr(arr));

```
                }
            }
        }
    }

    public void bubbleSortRec(int[] arr, int n) {

        for (int i = 0; i < n - 1; i++) {
            if (arr[i] > arr[i + 1]) {
                ++swapNum;
                // swap arr[i+1] and arr[i]
                swaps += arr[i] + "<->" + arr[i + 1] + " | ";
                // swappging the elements
                swap(arr, i, i + 1);
            }
        }

        if (n - 1 > 1) {
            System.out.print("\n\n" + swaps);
            System.out.print(
                    "\nUnsorted + Sorted => " + printArr(arr, 0,
n - 1) + " " + printArr(arr, n - 1, arr.length)
                            + "\n");
            swaps = "Swaps: |";
            bubbleSortRec(arr, n - 1);
        }
    }

    public int getSwapCount() {
        return swapNum;
    }

    public int[] bubbleSort(int i, int[] arr) {
        if (arr[i] > arr[i + 1] && (i + 1) < arr.length - 1) {
            swap(arr, i, i + 1);
            maxPos++;
        }
        if (i < arr.length - 1) {
```

```
            System.out.print("\n----------------------------------
-------------------");
            System.out.println("\nMax Places " + arr[i] + " is
shifted " + maxPos + " times");
            System.out.println("\nArray: " + printArr(arr));
            maxPos = 0;
            bubbleSort(i + 1, arr);
        }
        return arr;
    }

    public String printArr(int arr[]) {
        // printing the array
        if (arr.length > 10)
            return " ... ";
        else
            return Arrays.toString(arr);
    }

    public String printArr(int[] arr, int start, int end) {
        // printing the array
        if (arr.length > 10)
            return " ... ";
        else
            return Arrays.toString(Arrays.copyOfRange(arr,
start, end));
    }
    // They need the array size case

}


Driver Code:
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Scanner;

public class Driver {
```

```
    public static void main(String[] args) throws Exception {

        // Arraylist of Integers
        ArrayList<Integer> list = new ArrayList<Integer>();
        // array for sorting
        int[] array;
        // User input
        Scanner input = new Scanner(System.in);
        System.out.print("\n1.Random Array\n2.Roll.no
Input    : ");
        int choice = input.nextInt();
        if (choice == 1) {
            System.out.print("\nEnter the size of the array : ");
            int size = input.nextInt();
            for (int i = 0; i < size; i++) {
                list.add((int) (i + (Math.random() * 100)));
            }
            Collections.sort(list);
        } else {
            System.out.print("\nEnter the roll no: ");
            int roll = input.nextInt();
            for (int i = 0; i < 10; i++) {
                list.add(roll + (roll + 1) * i);
            }
        }
        System.out.print("\n1.Best Case\n2.Worst
Case\n3.Average Case\n4.Manual Choice\nEnter your
choice: ");
        int newChoice = input.nextInt();
        int listSize = list.size();
        array = new int[listSize];
        switch (newChoice) {
            case 1:
                for (int i = 0; i < listSize; i++) {
                    array[i] = list.get(i);
                }
                break;
            case 2:
                for (int i = listSize - 1; i >= 0; i--) {
```

```
                array[(listSize - 1) - i] = list.get(i);
            }
            break;
        case 3:
            Collections.shuffle(list);
            for (int i = 0; i < listSize; i++) {
                array[i] = list.get(i);
            }
            break;
        case 4:
            if (choice == 1) {
                System.out.print("\nAre u sure u want to
enter the array manually?(y/n): ");
                String choice1 = input.next().toLowerCase();
                if (choice1.equals("y")) {
                    System.out.print("Enter the size of the
array: ");
                    int size = input.nextInt();
                    list.clear();
                    System.out.print("Enter the elements of
the array(with space): ");
                    for (int i = 0; i < size; i++) {
                        list.add(input.nextInt());
                    }
                }
            }
            break;
        default:
            System.out.println("Invalid choice");
            break;
    }
    int arrLen = array.length;
    // Array segregation
    switch (arrLen) {
        case 0:
            input.close();
            throw new Exception("Array is empty");
        case 1:
            input.close();
```

```
                throw new Exception("Array has only one
element");
            default:
                System.out.print("\nArray: " +
Arrays.toString(array));
                break;
        }
        System.out
                .print("\n*****************
****************************\n");
        System.out.print("\nBubble Sort\n");
        // Bubble Sort
        System.out.print("\n1.Iteration Buuble
Sort\n2.Recursive Buuble Sort\n3.Bubble Sort (without for
loops)   : ");
        int choice2 = input.nextInt();
        // Create bubble sort object
        BubbleSort bs = new BubbleSort();
        // Call bubble sort function
        System.out.print("\nBefore sorting: " +
Arrays.toString(array));
        System.out.print("\n---------------------
----------------------\n");
        if (choice2 == 1) {
            bs.bubbleSortIteration(array);
        } else {
            bs.bubbleSortRec(array, listSize);
            System.out.print("\nTotal Swaps: " +
bs.getSwapCount());
        }

        System.out.print("\n---------------------
----------------------\n");
        System.out.print("\nFinal Array: " +
Arrays.toString(array) + " \n");
        // input closing method
        input.close();
    }
}
```

OUTPUT:

Best Case:

```
1.Random Array
2.Roll.no Input    : 1

Enter the size of the array : 6

1.Best Case
2.Worst Case
3.Average Case
4.Manual Choice
Enter your choice: 1

Array: [11, 16, 20, 37, 78, 90]
*********************************************

Bubble Sort

1.Iteration Buuble Sort
2.Recursive Buuble Sort    : 2

Before sorting: [11, 16, 20, 37, 78, 90]
---------------------------------------------


Swaps: |
Unsorted + Sorted => [11, 16, 20, 37, 78] [90]


Swaps: |
Unsorted + Sorted => [11, 16, 20, 37] [78, 90]


Swaps: |
Unsorted + Sorted => [11, 16, 20] [37, 78, 90]


Swaps: |
Unsorted + Sorted => [11, 16] [20, 37, 78, 90]

Total Swaps: 0
---------------------------------------------

Final Array: [11, 16, 20, 37, 78, 90]
```

Best Case :-

Array = [11, 16, 20, 37, 78, 80]

For First Pass

Array = [11, 16, 20, 37, 78, 80] ⇒ 5 comparison

Swaps : 0

For Second Pass

Array :- [11, 16, 20, 37, 78] + [80] ⇒ 4 comparisons (sorted)

Swaps = 0

For Third Pass

Array : [11, 16, 20, 37] + [78, 80] ⇒ 3 comparison

Swap = 0

For Fourth Pass

Array = [11, 16, 20] + [37, 78, 80] ⇒ 2 comparison

⋮

For Last Pass Array = [11, 16, 20, 37, 78, 80]

Swaps = 0 , Comparison = 15

Average Case:

```
1.Best Case
2.Worst Case
3.Average Case
4.Manual Choice
Enter your choice: 3

Array: [99, 55, 5, 36, 6, 97, 98]
**********************************************
```

```
Bubble Sort

1.Iteration Buuble Sort
2.Recursive Buuble Sort    : 3

Before sorting: [99, 55, 5, 36, 6, 97, 98]
--------------------------------------------


Swaps: |99<->55 | 99<->5 | 99<->36 | 99<->6 | 99<->97 | 99<->98 |
Unsorted + Sorted => [55, 5, 36, 6, 97, 98] [99]


Swaps: |55<->5 | 55<->36 | 55<->6 |
Unsorted + Sorted => [5, 36, 6, 55, 97] [98, 99]


Swaps: |36<->6 |
Unsorted + Sorted => [5, 6, 36, 55] [97, 98, 99]


Swaps: |
Unsorted + Sorted => [5, 6, 36] [55, 97, 98, 99]


Swaps: |
Unsorted + Sorted => [5, 6] [36, 55, 97, 98, 99]

Total Swaps: 10
--------------------------------------------

Final Array: [5, 6, 36, 55, 97, 98, 99]
```

Average case :-

Array: [99, 55, 5, 36, 6, 97, 98]

→ For First Pass :-

Array = [99, 55, 5, 36, 6, 97, 98]

Since 99 is largest, it does 6 swaps and reaches last

Array :- [55, 5, 36, 6, 97, 98, 99]

→ For Second Pass
Array : [55, 5, 36, 6, 97, 98, 99]

55 reaches it 97 and stops ; 3 comparison/swaps

Array = [5, 36, 6, 55, 97, 98, 99]

→ For third Pass
Array : [5, 36, 6, 55, 97, 98, 99] (Already sorted)

→ For fourth Pass
Array : [5, 36, 6, 55, 97, 98, 99]
Swap at 36, 6 = 1 swap

Array : [5, 6, 36, 55, 97, 98, 99]

∴ Array is sorted.

Worst Case:

```
Enter the size of the array : 5

1.Best Case
2.Worst Case
3.Average Case
4.Manual Choice
Enter your choice: 2

Array: [87, 82, 58, 24, 1]
**********************************************
```

```
Bubble Sort

1.Iteration Buuble Sort
2.Recursive Buuble Sort    : 2

Before sorting: [87, 82, 58, 24, 1]
--------------------------------------------


Swaps: |87<->82 | 87<->58 | 87<->24 | 87<->1 |
Unsorted + Sorted => [82, 58, 24, 1] [87]


Swaps: |82<->58 | 82<->24 | 82<->1 |
Unsorted + Sorted => [58, 24, 1] [82, 87]


Swaps: |58<->24 | 58<->1 |
Unsorted + Sorted => [24, 1] [58, 82, 87]

Total Swaps: 10
--------------------------------------------

Final Array: [1, 24, 58, 82, 87]
```

Time Complexity of Bubble Sort:

Worst Case Time Complexity
Θ(N^2) is the Worst Case Time Complexity of Bubble Sort. This is the case when the array is reversely sort
The number of swaps of two elements is equal to the number of comparisons in this case as every element is out of place.

T(N)=C(N)=S(N)=N∗(N−1)2T(N)=C(N)=S(N)=N∗(N−1)2

Therefore, in the worst case:

- Number of Comparisons: O(N^2) time
- Number of swaps: O(N^2) time
-

Best Case Time Complexity

Θ(N) is the Best Case Time Complexity of Bubble Sort.
This case occurs when the given array is already sorted.
T(N)=C(N)=NT(N)=C(N)=N
S(N)=0S(N)=0

Therefore, in the best case:

- Number of Comparisons: N = O(N) time
- Number of swaps: 0 = O(1) time

Average Case Time Complexity

Θ(N^2) is the Average Case Time Complexity of Bubble Sort.
The number of comparisons is constant in Bubble Sort so in average case, there is O(N^2) comparisons. This is because irrespective of the arrangement of elements, the number of comparisons C(N) is same.
For the number of swaps, consider the following points:

- If an element is in index I1 but it should be in index I2, then it will take a minimum of I2-I1 swaps to bring the element to the correct position.
- An element E will be at a distance of I3 from its position in sorted array
- The sum of maximum difference in position across all elements will be:

(N-1) + (N-3) + (N-5) ... + 0 + ... + (N-3) + (N-1)
= N x N - 2 x (1 + 3 + 5 + ... + N/2)
= N^2 - 2 x N^2 / 4
= N^2 - N^2 / 2
= N^2 / 2
Therefore, in average, the number of swaps = O(N^2).
Therefore, in the average case time complexity of Bubble sort:

- Number of Comparisons: O(N^2) time
- Number of swaps: O(N^2) time

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

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

TIME COMPLEXITY:

h = 5

Pratik Pojari 2020300054

```
public void BubbleSort(int arr[], int n){
  for(int i=0; i<(n-1); i++)
  {
    if(arr[i] > arr[i+1]){

      int temp = arr[i];
      arr[i] = arr[i+1];
      arr[i+1] = temp;
    }
  }
  if (n-1) > 1){

  BubbleSort(arr, n-1)
  }

}
```

Time complexity of Recursive Bubble Sort

$T(N) = T(N-1) + (N-1)$ → For loop

for return

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

$\vdots$

$T(1) = 1$

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

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

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

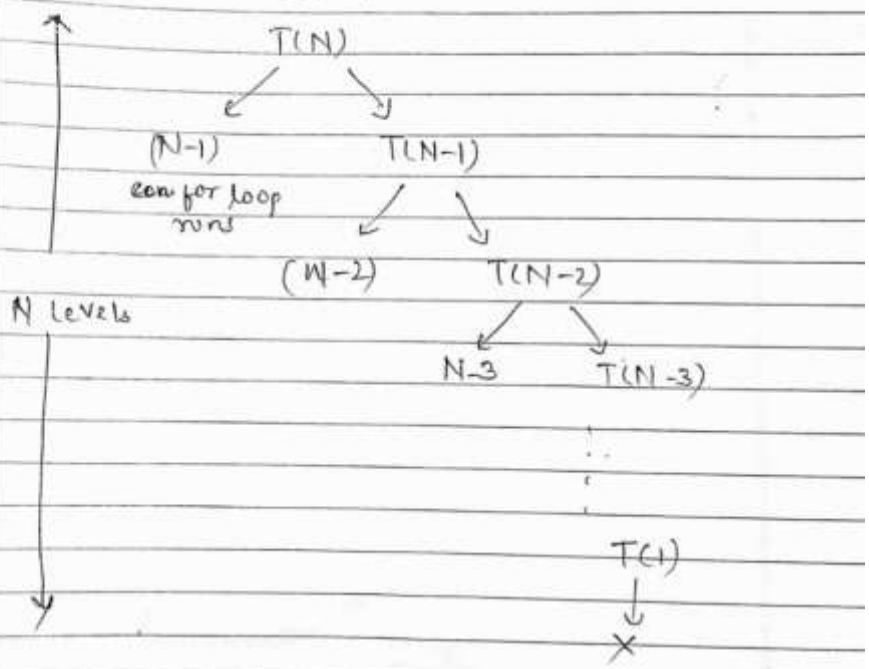
In A.P

Summation of 1, 2, 3 ... n 4) $= \frac{(n+1)n}{2}$

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

$= \frac{N(N-1)}{2} = \frac{N^2}{2} - \frac{N}{2}$

$\therefore$ Time complexity of bubble sort $= O(N^2)$



Time complexity for best case $= 1+1+1+\cdots$ N Times (No comparison)

T(N) = N Times

$\therefore$ O(N) is time complexity for best case of bobble sort

RESULT: Things learnt during procedural programming during solving of the problem

Computer Engineering Department &
Information Technology Engineering Department

Academic Year: 2021-2022

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

- Learnt how to implement bubble sort using iteration and recursion
- Learnt how to use exception in order to show that the array is of invalid length
- Learnt different time complexity cases of bubble sort and how it can be used.
- Learnt bubble sort required more time and space as it go through the array N^2 times

Extra Outputs:

```
Swaps: |67<->41 | 67<->16 | 67<->26 | 83<->34 | 83<->27 | 83<->71 | 83<->49 | 83<->15 | 83<->82 |
 83<->60 | 83<->15 |
Unsorted + Sorted =>  ...   ...


Swaps: |41<->16 | 41<->26 | 67<->34 | 67<->27 | 71<->49 | 71<->15 | 82<->60 | 82<->15 |
Unsorted + Sorted =>  ...   ...


Swaps: |41<->34 | 41<->27 | 67<->49 | 67<->15 | 71<->60 | 71<->15 |
Unsorted + Sorted =>  ...   ...


Swaps: |34<->27 | 49<->15 | 67<->60 | 67<->15 |
Unsorted + Sorted =>  ...   ...


Swaps: |41<->15 | 60<->15 |
Unsorted + Sorted =>  ...   ...


Swaps: |34<->15 | 49<->15 |
Unsorted + Sorted =>  ...   ...


Swaps: |27<->15 | 41<->15 |
Unsorted + Sorted =>  ...   ...


Swaps: |26<->15 | 34<->15 |
Unsorted + Sorted =>  ...   ...


Swaps: |16<->15 | 27<->15 |
Unsorted + Sorted =>  ...   ...


Swaps: |26<->15 |
Unsorted + Sorted =>  ...   ...


Swaps: |16<->15 |
Unsorted + Sorted =>  ...   ...

Total Swaps: 53
-----------------------------------------

Final Array: [15, 15, 16, 26, 27, 34, 41, 49, 60, 67, 71, 82, 83, 86]
```

(If the array is too big it shows the '….')

(When the array size is 0/1 it shows error)

```
1.Random Array
2.Roll.no Input    : 1

Enter the size of the array : 0

1.Best Case
2.Worst Case
3.Average Case
4.Manual Choice
Enter your choice: 1
Exception in thread "main" java.lang.Exception: Array is empty
        at Driver.main(Driver.java:78)
```

(User gets a chance to renter array)

```
1.Random Array
2.Roll.no Input    : 1

Enter the size of the array : 5

1.Best Case
2.Worst Case
3.Average Case
4.Manual Choice
Enter your choice: 4

Are u sure u want to enter the array manually?(y/n): y
Enter the size of the array: 5
Enter the elements of the array(with space): 1 3 6 2 7

Array: [1, 3, 6, 2, 7]
**********************************************
```

(Iterative Bubble sort)

```
1.Random Array
2.Roll.no Input    : 1

Enter the size of the array : 6

1.Best Case
2.Worst Case
3.Average Case
4.Manual Choice
Enter your choice: 1

Array: [20, 75, 91, 93, 96, 97]
**********************************************

Bubble Sort

1.Iteration Buuble Sort
2.Recursive Buuble Sort    : 1

Before sorting: [20, 75, 91, 93, 96, 97]
----------------------------------------------
Iteration       Swap            Array
1               ----            [20, 75, 91, 93, 96, 97]
2               ----            [20, 75, 91, 93, 96, 97]
3               ----            [20, 75, 91, 93, 96, 97]
4               ----            [20, 75, 91, 93, 96, 97]
5               ----            [20, 75, 91, 93, 96, 97]
6               ----            [20, 75, 91, 93, 96, 97]
7               ----            [20, 75, 91, 93, 96, 97]
8               ----            [20, 75, 91, 93, 96, 97]
9               ----            [20, 75, 91, 93, 96, 97]
10              ----            [20, 75, 91, 93, 96, 97]
11              ----            [20, 75, 91, 93, 96, 97]
12              ----            [20, 75, 91, 93, 96, 97]
13              ----            [20, 75, 91, 93, 96, 97]
14              ----            [20, 75, 91, 93, 96, 97]
15              ----            [20, 75, 91, 93, 96, 97]

----------------------------------------------

Final Array: [20, 75, 91, 93, 96, 97]
```

(Roll no input)

```
Bubble Sort

1.Iteration Buuble Sort
2.Recursive Buuble Sort    : 2

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


Swaps: |439<->109 | 439<->164 | 439<->329 | 439<->384 | 439<->274 | 439<->219 | 549<->54 | 549
<->494 |
Unsorted + Sorted => [109, 164, 329, 384, 274, 219, 439, 54, 494] [549]


Swaps: |384<->274 | 384<->219 | 439<->54 |
Unsorted + Sorted => [109, 164, 329, 274, 219, 384, 54, 439] [494, 549]


Swaps: |329<->274 | 329<->219 | 384<->54 |
Unsorted + Sorted => [109, 164, 274, 219, 329, 54, 384] [439, 494, 549]


Swaps: |274<->219 | 329<->54 |
Unsorted + Sorted => [109, 164, 219, 274, 54, 329] [384, 439, 494, 549]


Swaps: |274<->54 |
Unsorted + Sorted => [109, 164, 219, 54, 274] [329, 384, 439, 494, 549]


Swaps: |219<->54 |
Unsorted + Sorted => [109, 164, 54, 219] [274, 329, 384, 439, 494, 549]


Swaps: |164<->54 |
Unsorted + Sorted => [109, 54, 164] [219, 274, 329, 384, 439, 494, 549]


Swaps: |109<->54 |
Unsorted + Sorted => [54, 109] [164, 219, 274, 329, 384, 439, 494, 549]

Total Swaps: 20
--------------------------------------------

Final Array: [54, 109, 164, 219, 274, 329, 384, 439, 494, 549]
```