Merge Sort Algorithm

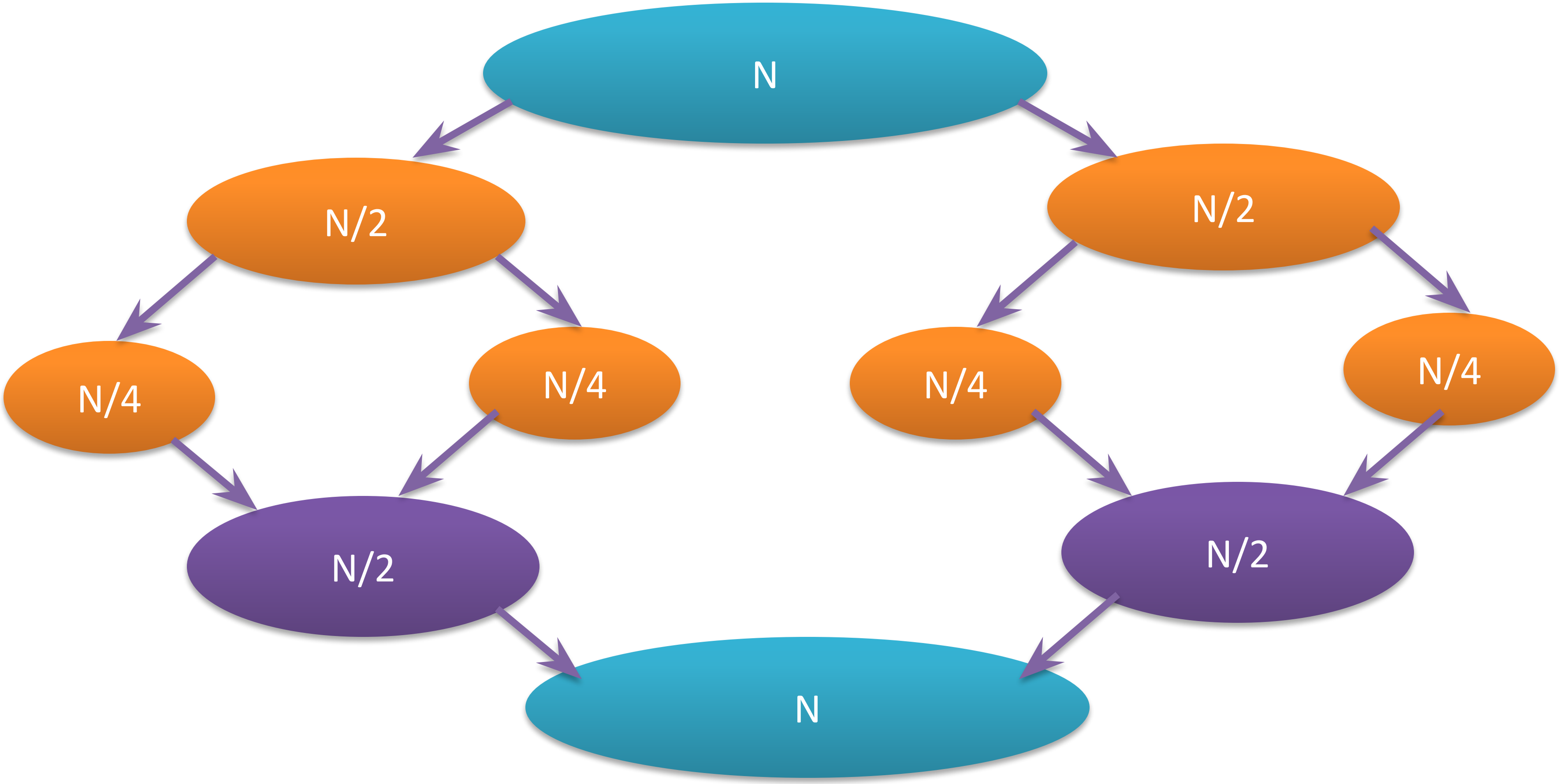
Problem Definition: Implement MergeSort Algorithm to sort a given set of elements and determine the time required to sort the elements. Plot the graph of Computing V/s Problem size.

Objectives of the Experiment:

1. To introduce the divide and conquer strategy
2. Present the working of MergeSort
3. Analyze the Algorithm & Estimate computing time

Theoretical Background of the Experiment

Concept : Divide and Conquer - Problem Solving Design strategy

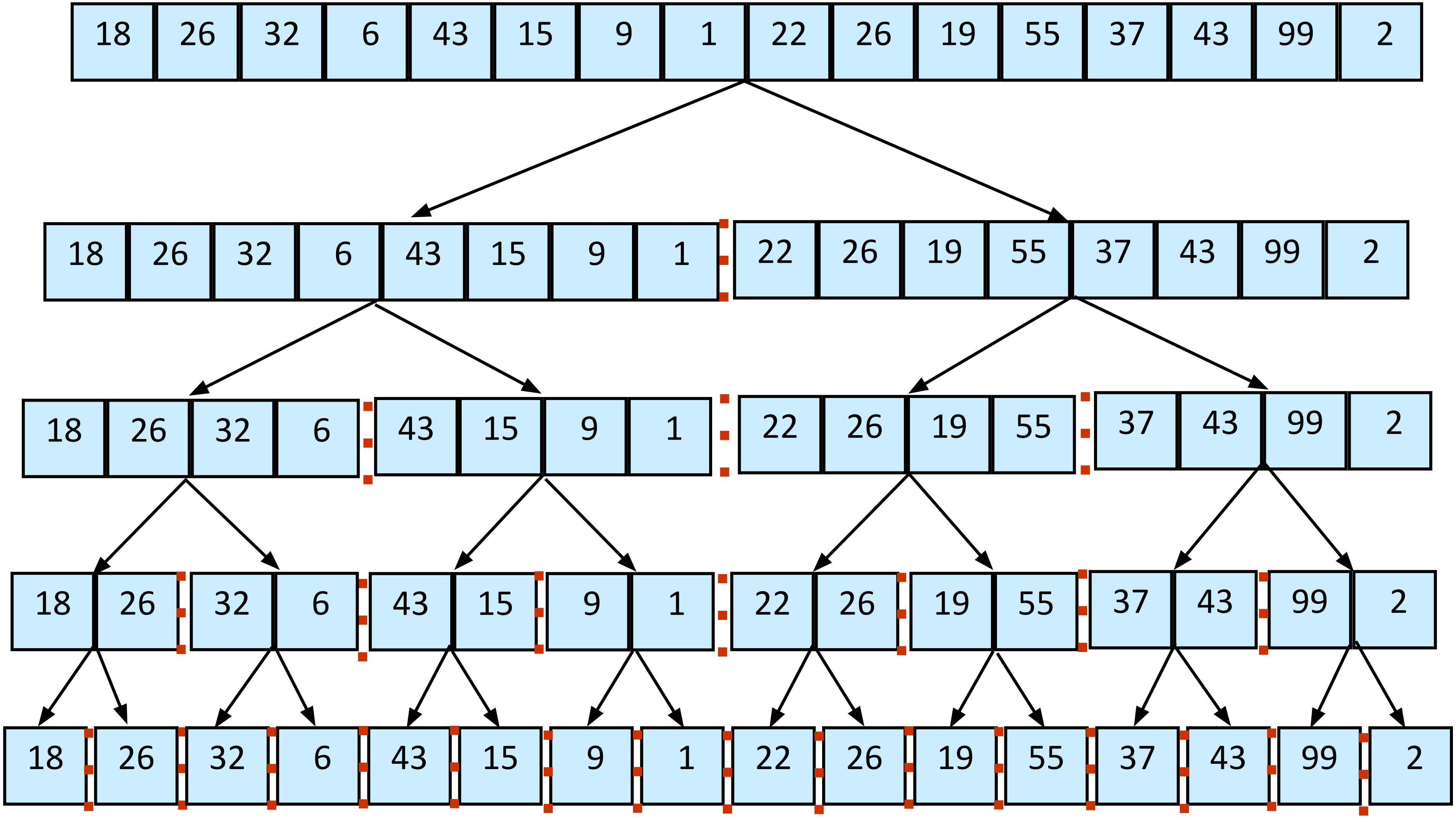


An Example: Merge Sort

***Sorting Problem*:** Sort a sequence of *n* elements into non-decreasing order.

- ***Divide***: Divide the *n*-element sequence to be sorted into two subsequences of *n/2* elements each
- ***Conquer:*** Sort the two subsequences recursively using merge sort.
- ***Combine***: Merge the two sorted subsequences to produce the sorted answer.

Merge Sort – Example



18 26 32 6 43 15 9 1 22 26 19 55 37 43 99 2

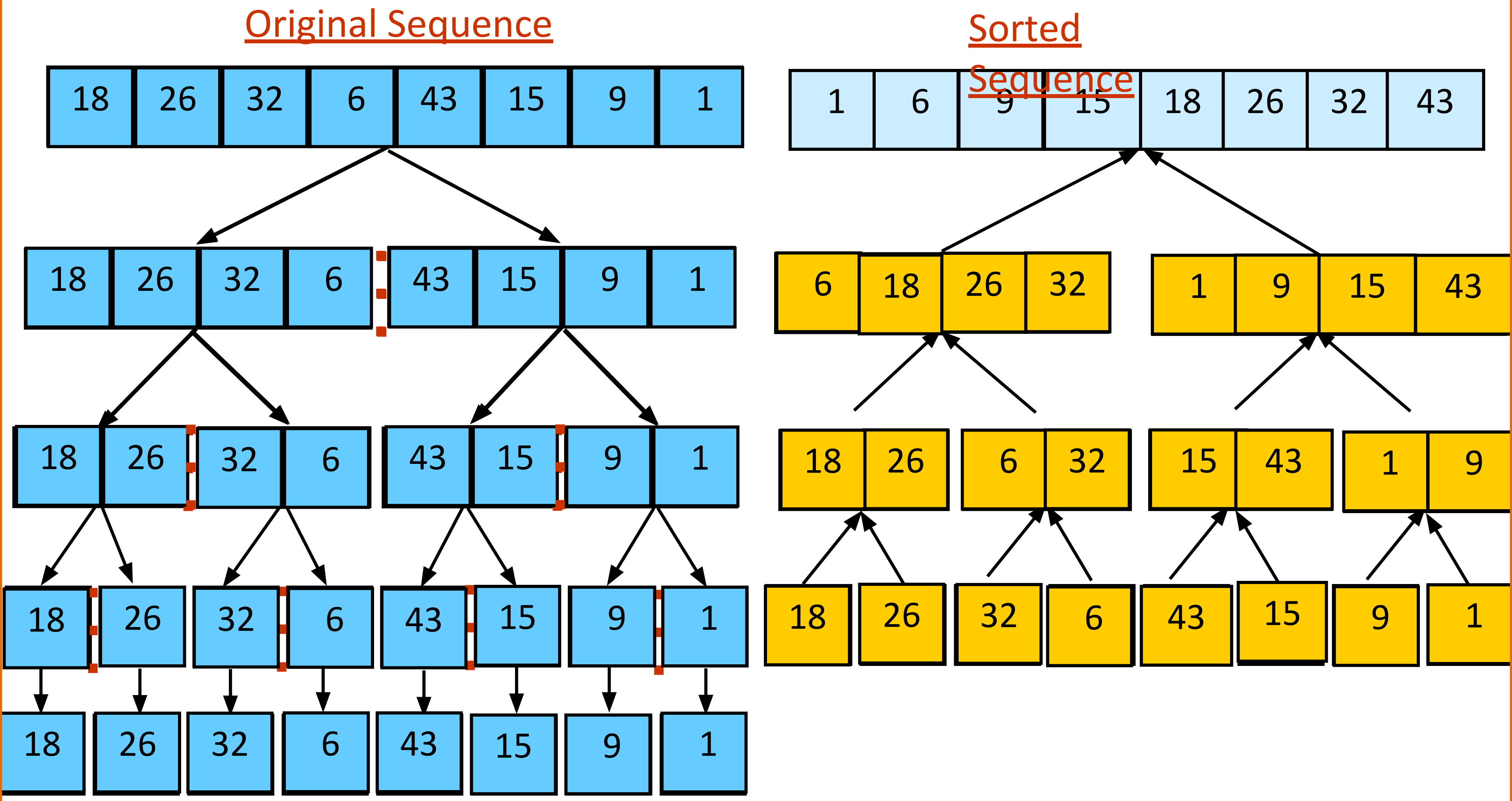
18 26 32 6 43 15 9 1 | 22 26 19 55 37 43 99 2

18 26 32 6 | 43 15 9 1 | 22 26 19 55 | 37 43 99 2

18 26 | 32 6 | 43 15 | 9 1 | 22 26 | 19 55 | 37 43 | 99 2

18 | 26 | 32 | 6 | 43 | 15 | 9 | 1 | 22 | 26 | 19 | 55 | 37 | 43 | 99 | 2

Merge Sort – Example



Original Sequence

18 26 32 6 43 15 9 1

18 26 32 6 | 43 15 9 1

18 26 | 32 6 | 43 15 | 9 1

18 | 26 | 32 | 6 | 43 | 15 | 9 | 1

18 26 32 6 43 15 9 1

Sorted Sequence

1 6 9 15 18 26 32 43

6 18 26 32 | 1 9 15 43

18 26 | 6 32 | 15 43 | 1 9

18 | 26 | 32 | 6 | 43 | 15 | 9 | 1

Merge-Sort (A, left, right)

Input : a sequence of *n* numbers stored in array A

Output : an ordered sequence of *n* numbers

```
Begin                         // sort A[left..right] by divide & conquer
    if left < right
       then mid ← $\lfloor (left+right)/2 \rfloor$
            MergeSort (A, left, mid)
            MergeSort (A, mid+1, right)
            Merge (A, left, mid, right) // merges A[left..mid] with A[md+1..right]
End
```

Initial Call: MergeSort(*A*, 0, *n-1*)

Merge (A, left, mid, right)

Input : Two sublists

Output : Ordered List

```
Begin                          // Merge the two sublists

   1    While ( either of the list is not processed )
   2     if( A[i] < A[j] )
           B[k] = A[i]   // Copy smaller of the two sublists in the auxiliary array
           else
            B[k]= A[j]
   3     Copy the remaining elements to B from the sublist
   4     Copy the auxiallary list B to Original list A

   End
```

Sample Input / Output or Test Cases

Sample 1 : n = 9

4, 2, 7, 1, 9, 0, 3, 8, 11

Sample 2 : n = 10

9, 8, 7, 6, 5, 4, 3, 2, 1, 0

Generate the list using Random Function

Time Complexity : T(n) = n*logn (Average Case)

Compute the time using Time function in Java

Performance Comparison

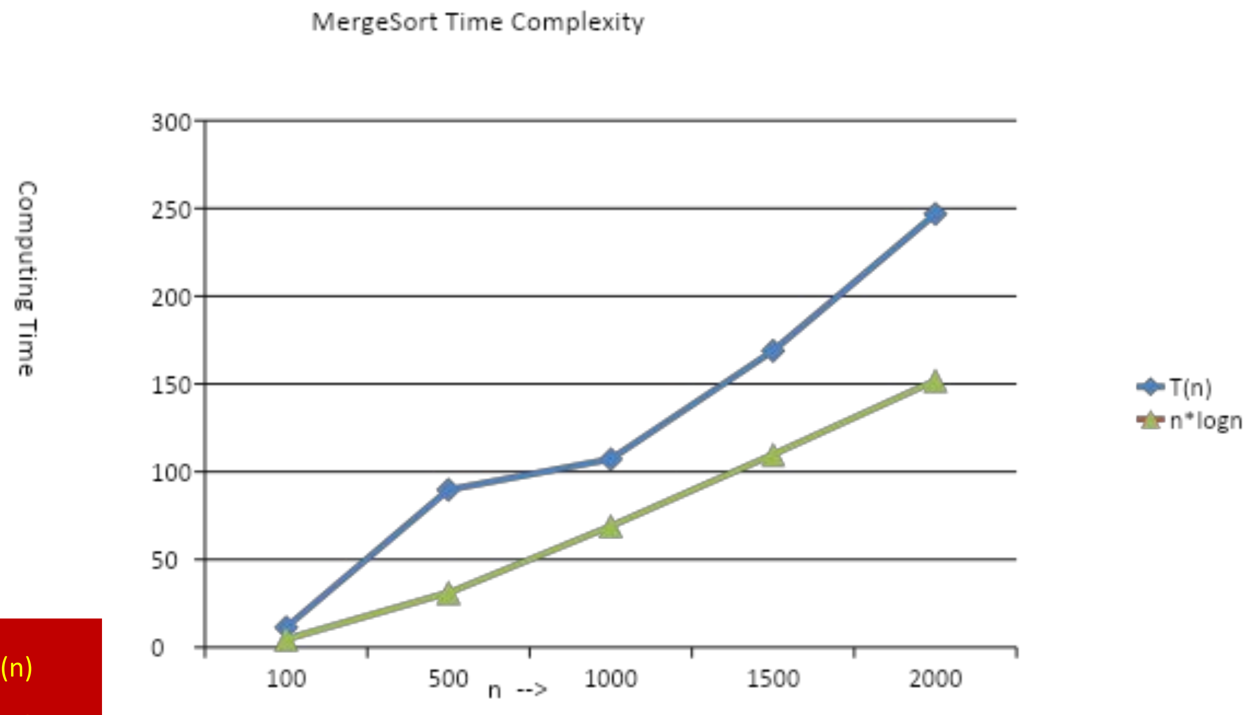
Size	MergeSort	BubbleSort
n	n*logn	n^2
100	4.60517	10000
500	31.07304	250000
1000	69.07755	1000000
1500	109.6983	2250000
2000	152.018	4000000

Generating Input & Estimating Computing time

```
int min=1, max=10000
Scanner sc=new Scanner(System.in);      // Instantiate an Object of Scanner class
System.out.println("Eneter n :");
    n=sc.nextInt();                     // Read the size of the array
    System.out.println("Enter the elements :"); // Read the elements
    for(i=0;i<n;i++)
      a[i]=ThreadLocalRandom.current().nextInt(min,max+1);


final long startTime = System.nanoTime();
Call to MergeSort
final long duration = System.nanoTime() - startTime;
System.out.println(duration);
```

Sample Input / Output or Test Cases



n	T(n)
100	11.3
500	89.7
1000	107.08
1500	168.91
2000	246.86

Learning Outcome of the Experiment and Conclusion

At the end of the session, students should be able to :

1. Explain the working of Divide and Conquer Strategy
2. Demonstrate the working of MergeSort algorithm on a given set of size n
3. Write the program in Java to implement MergeSort Algorithm and estimate the computing time using appropriate time functions.
4. Plot a graph of Computing time V/s Size of the input and draw conclusions