Pandit Deendayal Energy University, Gandhinagar School of Technology

Department of Computer Science & Engineering

Design & Analysis of Algorithm Lab (20CP209P)

Name: **Padshala Smit Jagdishbhai**

Enrolment No: **21BCP187**

Semester: **IV Division**: **3 (G6)**

Branch: **Computer Science & Engineering**

Practical No. 1 & 2

Aim:

Write a program to demonstrate Insertion Sort, Selection Sort, Quick Sort and Merge Sort and analyze the time complexity with different range of inputs.

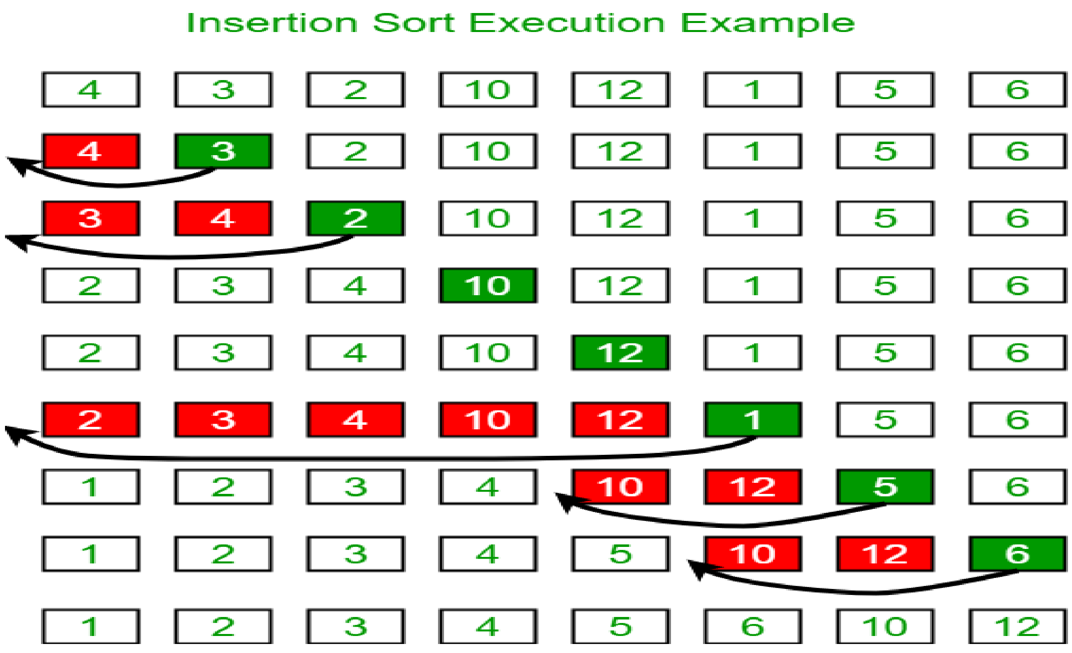
Introduction:

Insertion Sort:

Insertion sort is a sorting algorithm that places an unsorted element at its suitable place in each iteration.

Insertion sort works similarly as we sort cards in our hand in a card game.

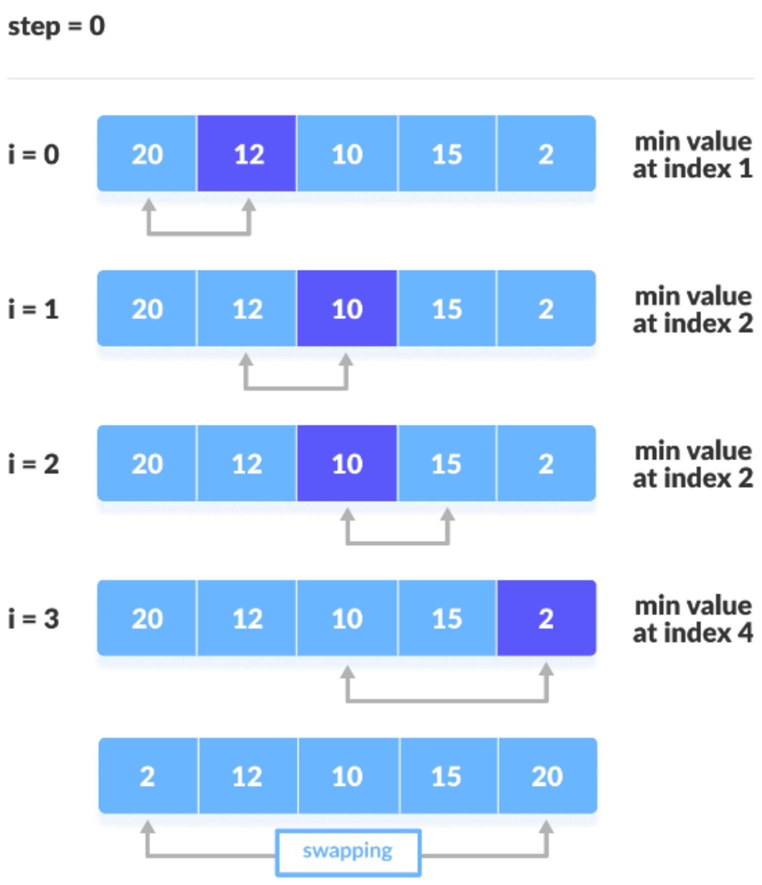
We assume that the first card is already sorted then, we select an unsorted card. If the unsorted card is greater than the card in hand, it is placed on the right otherwise, to the left. In the same way, other unsorted cards are taken and put in their right place.
A similar approach is used by insertion sort.



Selection Sort:

A selection sort algorithm sorts the elements by iterating over the entire array. It selects the smallest element from the unsorted array and swaps it with the element present at the first index.

It again finds the next smallest element from the unsorted array and swaps it with the element at the second index. This keeps going on until we achieve our resultant sorted array.

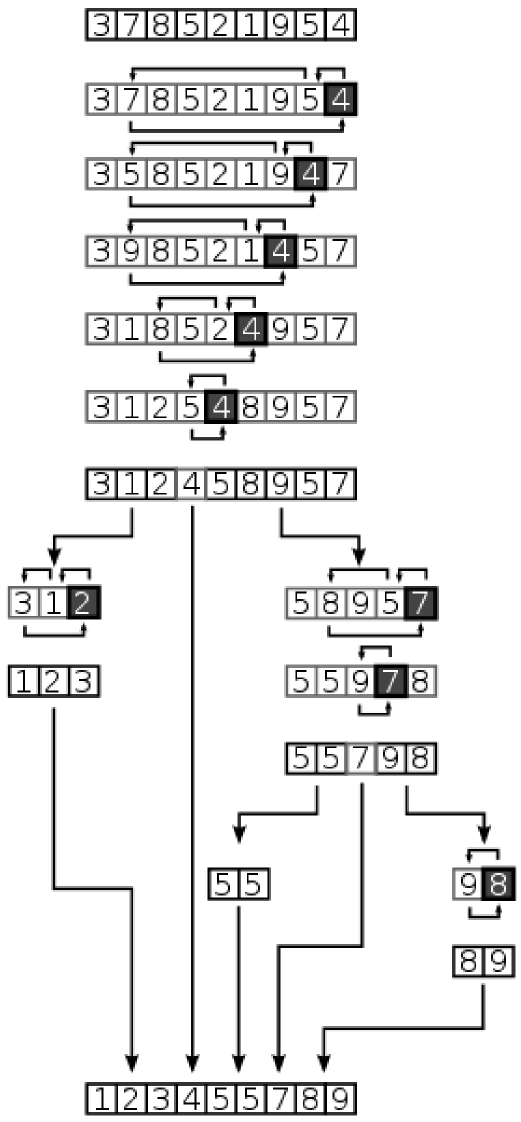


Quick Sort:

Quick Sort is a sorting algorithm based on the divide and conquer approach.

A large array is partitioned into two arrays one of which holds values smaller than the specified value, say pivot, based on which the partition is made and another array holds values greater than the pivot value.

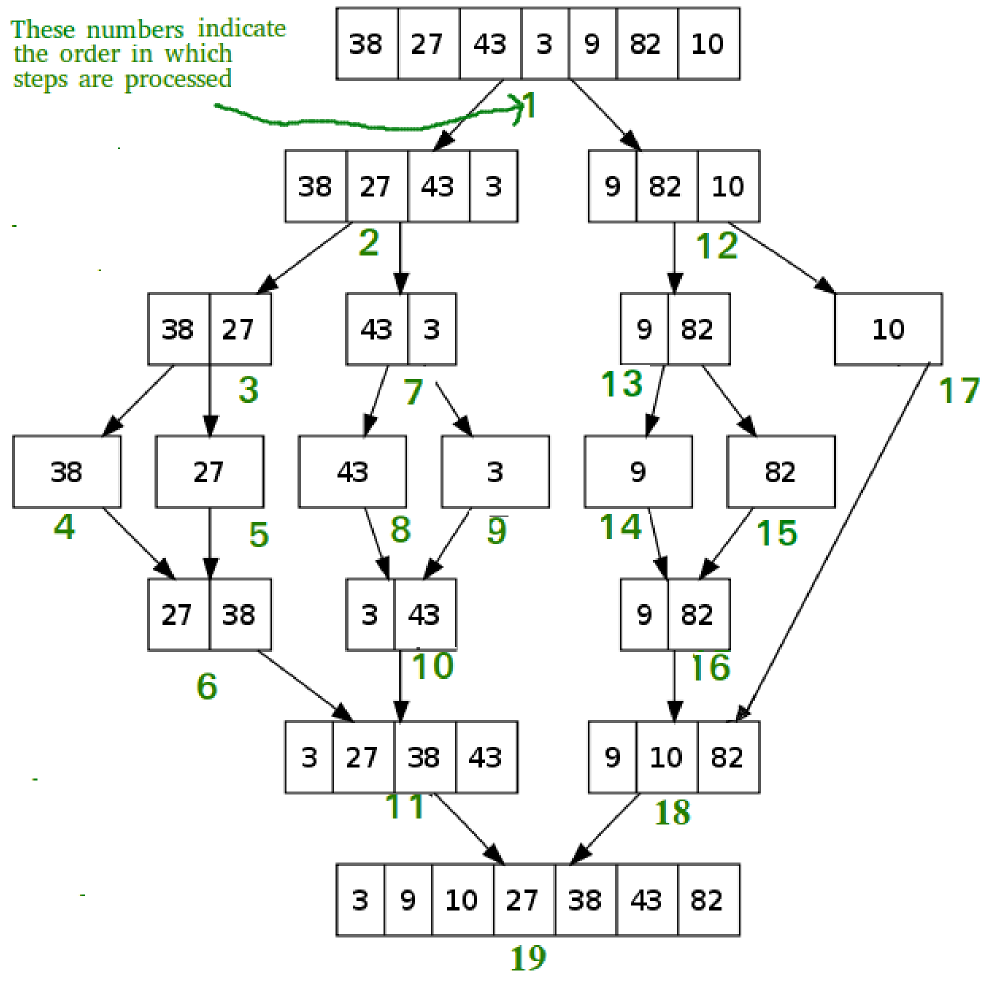
Quicksort partitions an array and then calls itself recursively twice to sort the two resulting subarrays. This algorithm is quite efficient for large-sized data sets as its average and worst-case complexity are O(n2), respectively.



Merge Sort:

Merge sort is a sorting algorithm based on the divide and conquer strategy. It works by dividing the unsorted list into n sub-lists, each containing one element, and then repeatedly merging sub-lists to produce new sorted sub-lists until there is only one sub-list remaining.

The algorithm has a time complexity of O(n log n) and is considered one of the most efficient and widely used sorting algorithms. It is a stable sorting algorithm, which means that it maintains the relative order of equal elements in the sorted list. The algorithm uses a recursive approach to sort the sub-lists.



-

Algorithm:

Insertion Sort:

```
for j=2 to arr.length
key = arr[j]
i=j-1
while (arr[i] > key and i > 0)
arr[i+1]=arr[i]
i=i-1
arr[i+1]=key
```

Selection Sort:

```
For i=0 to arr.length-1
Min=I;
For j=i+1 to arr.length
If(arr[j] < arr[min])
Min=j;
If(min != i)
Swap arr[min] and arr[i]
```

Quick Sort:

```
PARTITION (array A, start, end) {
  pivot = A[end]
  i = start-1
  for j = start to end -1 {
  do if (A[j] < pivot) {
  then i = i + 1
  swap A[i] with A[j]
   }}
  swap A[i+1] with A[end]
  return i+1
}

QUICKSORT (array A, start, end)
{
```

```
if (start < end)    {
p = partition(A, start, end)
QUICKSORT (A, start, p - 1)
QUICKSORT (A, p + 1, end)
}   }
```

Merge Sort:

```
MERGE_SORT(arr, beg, end)
if beg < end
set mid = (beg + end)/2
MERGE_SORT(arr, beg, mid)
MERGE_SORT(arr, mid + 1, end)
MERGE (arr, beg, mid, end)
```

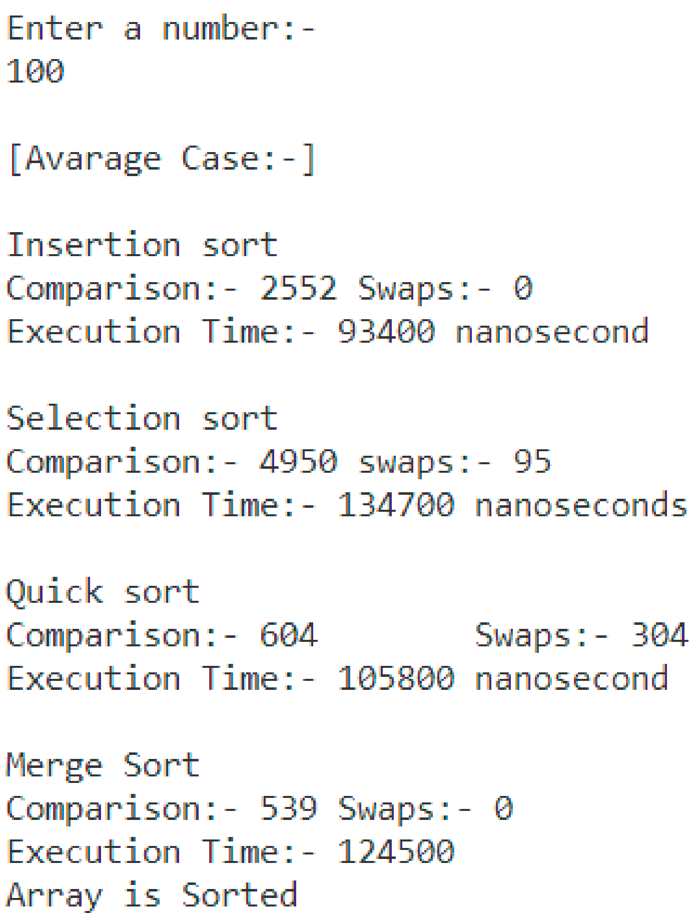
Program:

```
/*
This code sorts an array of integers using four different
algorithms:
Insertion sort, Selection sort, Quick sort, and Merge sort.
For each sorting algorithm, it displays the execution time in
nanoseconds.
*/

import java.util.Arrays;
import java.util.Random;
import java.util.Scanner;

// Insertion Sort Class
class Insertion{
    public void Insertion(int arr[]){
        // Method to sort the array using Insertion sort
        display di = new display();
        System.out.println("\nInsertion sort");
        // call method to display initial array
        di.bdisplay(arr);
        long comp=0;
        long swap=0;
        // Start the timer
        long starti = System.nanoTime();
        for (int j = 1; j < arr.length; j++) {
            int key = arr[j];
            int i = j-1;
             // Shift the elements of the array until the correct position
for key is found
            while ( (i > -1) && ( arr[i] > key ) ) {
                arr[i+1] = arr[i];
                i--;
                comp++;
            }
            arr[i+1] = key;
            comp++;
        }
        long endi = System.nanoTime();
        // End the timer
        System.out.println("Comparison:- "+comp+" Swaps:-
"+swap);
        System.out.println("Execution Time:- " + (endi-starti)+ "
nanosecond");
        di.adisplay(arr);
        // Call method to display sorted array
    }
}


// Selection Sort Class
class Selection{
    public void Selection(int arr[]){
        // Method to sort the array using Selection Sort
        display ds = new display();
        System.out.println("\nSelection sort");
        ds.bdisplay(arr);
        int comp=0,swap=0;
        long start = System.nanoTime();
        for(int i=0;i<arr.length-1;i++){
            int min=i;
            for(int j=min+1;j<arr.length;j++){
                comp++;
                if(arr[j] < arr[min]){
                    min = j;
                }
            }
            if(min!=i){
            // Swap the elements to place the minimum element in its
correct position

            swap++;
            arr[min] = arr[min] + arr[i];
            arr[i] = arr[min] - arr[i];
            arr[min] = arr[min] - arr[i];
            // int temp = arr[min];
            // arr[min] = arr[i];
            // arr[i] = temp;
            }
        }
        long end = System.nanoTime();
        System.out.println("Comparison:- "+comp+" swaps:-
"+swap);
        System.out.println("Execution Time:- "+ (end-start)+"
nanoseconds");
        ds.adisplay(arr);
    }

}
```

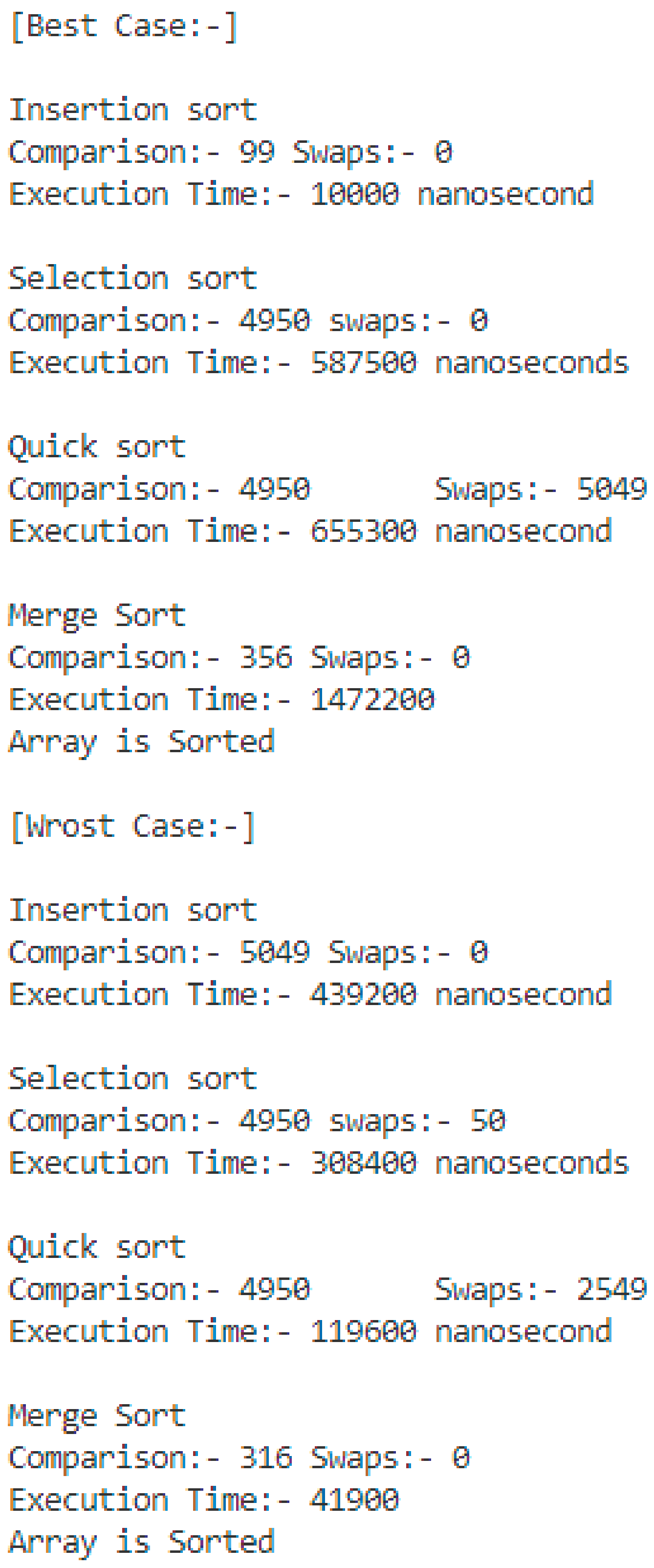
```
// Quick Sort Class
class QuickSort{

    static long comparisonCount;
    static long swapCount;
    // Partition method to divide the array into two sub part
    public static int partition(int[] arr, int low, int high) {
        // Chooose the last element as the pivot
        int pivot = arr[high];
        int i = (low-1);
        for (int j=low; j<high; j++) {
            comparisonCount++;
            // If the current element is smaller or equal to the pivot,
swap  it with the element at index i+1
            if (arr[j] <= pivot) {
                i++;
                // swap(arr, i, j);
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
                swapCount++;
            }
        }
        // swap(arr, i+1, high);
        int temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;
        // swapCount++;
        return i+1;
    }

    // Quick Sort Algorithm
    public static void quicksort(int[] arr, int left, int right) {
        //        // Recursively sort the sub-arrays on the left and right
side of the pivot
```

```
        if(left<right){
            int pi=partition(arr, left, right);
            quicksort(arr, left, pi-1);
            quicksort(arr, pi+1, right);
        }
    }

    // Quick Sort the given array and display the results
    public void qsort(int[] arr) {
        display dq = new display();
        System.out.println("\nQuick sort");
        dq.bdisplay(arr);
        comparisonCount=0;
        swapCount=0;
        long start = System.nanoTime();
        quicksort(arr, 0, arr.length-1);
        long end = System.nanoTime();
        System.out.println("Comparison:-
"+comparisonCount+"\tSwaps:- "+swapCount);
        System.out.println("Execution Time:- " + (end-start)+ "
nanosecond");
        dq.adisplay(arr);
    }
}

// Merge Sort Class
class MergeSort{

    static long comparisons=0;
    static long swaps=0;

    // Merge Sort Algorithm
    public static void mergeSort(int[] array, int left, int right) {
        // Recursively divide the array into smaller sub-arrays and
sort them
        if (left < right) {
            int middle = (left + right) / 2;
            mergeSort(array, left, middle);
            mergeSort(array, middle + 1, right);
            merge(array, left, middle, right);
        }
    }

    // Merge two sub array into a single sorted array
    public static void merge(int[] array, int left, int middle, int
right) {
        // Calculate the size of left and right array
        int leftSize = middle - left + 1;
        int rightSize = right - middle;

        //  Create array for left and right subarray
        int[] leftArray = new int[leftSize];
        int[] rightArray = new int[rightSize];
        for (int i = 0; i < leftSize; i++) {
            leftArray[i] = array[left + i];
        }
        for (int i = 0; i < rightSize; i++) {
            rightArray[i] = array[middle + 1 + i];
        }

        // Merge the sub array back into the initial array
        int i = 0, j = 0, k = left;
        while (i < leftSize && j < rightSize) {
            // Compare the elements from left and right subarray
            if (leftArray[i] <= rightArray[j]) {
                array[k] = leftArray[i];
                i++;
            } else {
                array[k] = rightArray[j];
                j++;
            }
            comparisons++;
            k++;
        }
        // Add any remaining elements from the left subarray to the
original array
        while (i < leftSize) {
            array[k] = leftArray[i];
            i++;
            k++;
        }

         // Add any remaining elements from the right subarray to
the original array
        while (j < rightSize) {
            array[k] = rightArray[j];
            j++;
            k++;
        }
    }

    // Merge Sort the given array and display the results
    public void msort(int[] array) {
        display dm =new display();
        System.out.println("\nMerge Sort");
        dm.bdisplay(array);
        comparisons = 0;
        swaps = 0;
        long start = System.nanoTime();
        mergeSort(array, 0, array.length - 1);
        long end = System.nanoTime();
        System.out.println("Comparison:- "+comparisons+" Swaps:-
"+swaps);
        System.out.println("Execution Time:- "+(end-start));
        dm.adisplay(array);
    }
}

class display{
    public void bdisplay(int arr[]) {
        if(arr.length<=10){
        System.out.println("Before Sort");
        for(int i:arr){
            System.out.print(i+" ");
        }
        System.out.println();
        }
    }

    public void adisplay(int arr[]) {
        if(arr.length<=10){
        System.out.println("After Sort");
        for(int i:arr){
            System.out.print(i+" ");
        }
        System.out.println();
        }
    }

}

// Main Class
public class allsort {

    // Reverse method for reverse the array
    public static void reverse(int arr[]){
        for(int i=0;i<arr.length/2;i++){
            int temp = arr[i];
            arr[i] = arr[arr.length-1-i];
            arr[arr.length-1-i] = temp;
        }
    }

    // This method for array is sorted or not
    public static void isSorted(int[] a) {
        boolean s;
        for (int i=1; i<a.length; i++) {
            if (a[i] < a[i-1]) {
                System.out.println("Array is not Sorted");
                break;
            }
        }
        System.out.println("Array is Sorted");
    }

    // This method implements four sorting algorithm
    public static int[] sort(int[] arr) {
        // Creating objects of Insertion, Selection, QuickSort, and
MergeSort classes
        Insertion i = new Insertion();
        Selection s = new Selection();
        QuickSort q = new QuickSort();
        MergeSort m = new MergeSort();

        // Calling the Insertion sort method and passing a copy of the
input array
        i.Insertion(Arrays.copyOf(arr, arr.length));
        s.Selection(Arrays.copyOf(arr, arr.length));
        q.qsort(Arrays.copyOf(arr, arr.length));
        // m.msort(Arrays.copyOf(arr, arr.length));

        // Calling the MergeSort sort method and passing the input
array
        m.msort(arr);

        // Checking if the input array is sorted or not
        isSorted(arr);

        // Returning the sorted input array
        return arr;
    }

    public static void main(String[] args) {
        display d = new display();
        Random random = new Random();
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter a number:- ");
        int n=sc.nextInt();
        int[] arr1 = new int[n];
        // Generate unique random numbers
        for(int i=0;i<n;i++){
            arr1[i] = random.nextInt(n);
            for(int j=0;j<i;j++){
                if(arr1[i] == arr1[j]){
                    i--;
                    break;
                }
            }
        }

        System.out.println("\n[Avarage Case:-]");
        // d.bdisplay(arr1);
        sort(arr1);
        // d.adisplay(arr1);

        // Best Case
        /* Best case input is avg case output */
        System.out.println("\n[Best Case:-]");
        // d.bdisplay(arr1);
        sort(arr1);
        // d.adisplay(arr1);
        // Wrost case
        /* Wrost case input is reverse of best case output */
        System.out.println("\n[Wrost Case:-]");
        reverse(arr1);
        // d.bdisplay(arr1);
        sort(arr1);
        // d.adisplay(arr1);
    }
}
```

Output:-

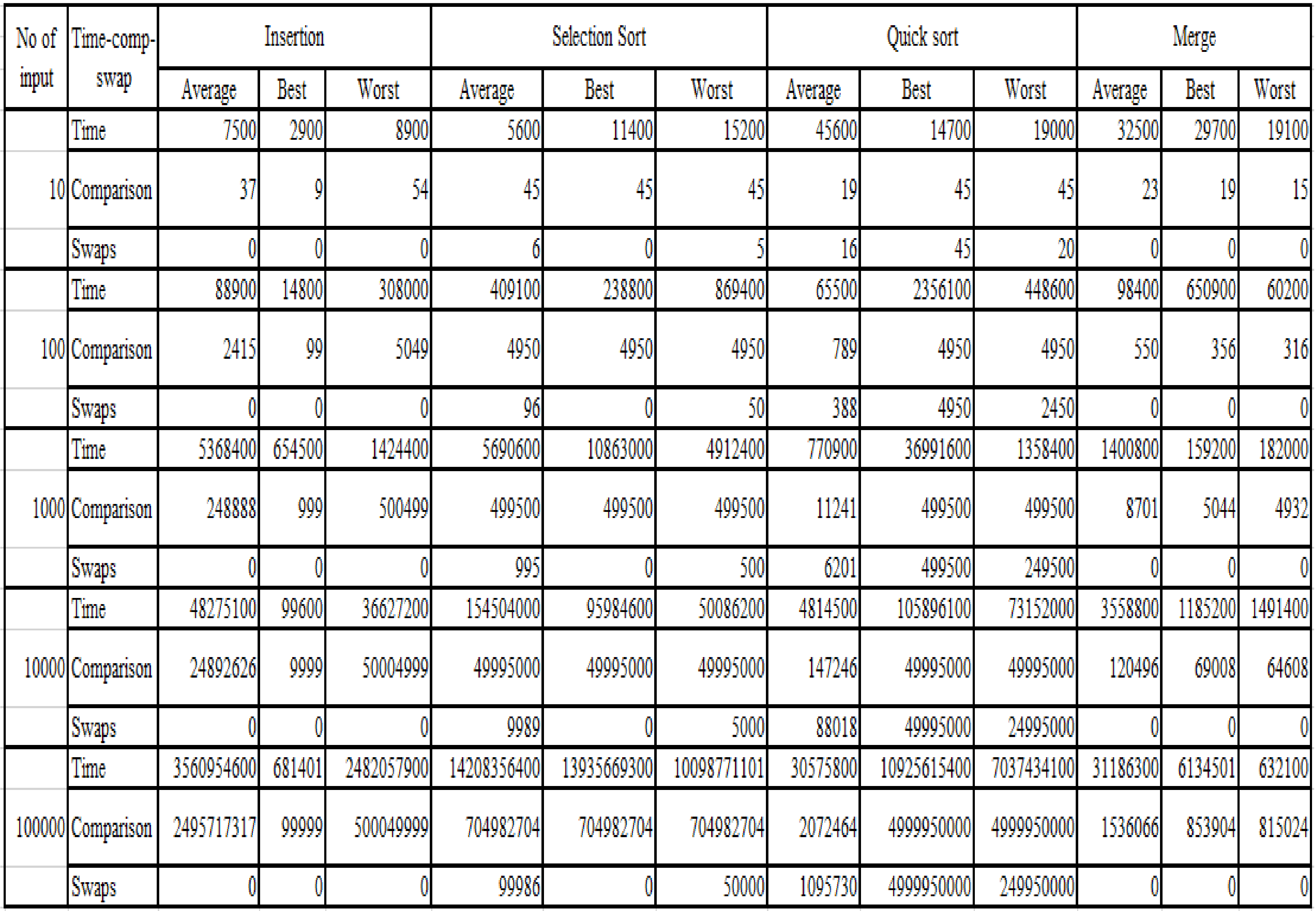


```
Enter a number:-
100

[Avarage Case:-]

Insertion sort
Comparison:- 2552 Swaps:- 0
Execution Time:- 93400 nanosecond

Selection sort
Comparison:- 4950 swaps:- 95
Execution Time:- 134700 nanoseconds

Quick sort
Comparison:- 604        Swaps:- 304
Execution Time:- 105800 nanosecond

Merge Sort
Comparison:- 539 Swaps:- 0
Execution Time:- 124500
Array is Sorted
```



```
[Best Case:-]

Insertion sort
Comparison:- 99 Swaps:- 0
Execution Time:- 10000 nanosecond

Selection sort
Comparison:- 4950 swaps:- 0
Execution Time:- 587500 nanoseconds

Quick sort
Comparison:- 4950       Swaps:- 5049
Execution Time:- 655300 nanosecond

Merge Sort
Comparison:- 356 Swaps:- 0
Execution Time:- 1472200
Array is Sorted

[Wrost Case:-]

Insertion sort
Comparison:- 5049 Swaps:- 0
Execution Time:- 439200 nanosecond

Selection sort
Comparison:- 4950 swaps:- 50
Execution Time:- 308400 nanoseconds

Quick sort
Comparison:- 4950       Swaps:- 2549
Execution Time:- 119600 nanosecond

Merge Sort
Comparison:- 316 Swaps:- 0
Execution Time:- 41900
Array is Sorted
```

Analysis:



No of input	Time-comp-swap	Insertion			Selection Sort			Quick sort			Merge		
		Average	Best	Worst	Average	Best	Worst	Average	Best	Worst	Average	Best	Worst
	Time	7500	2900	8900	5600	11400	15200	45600	14700	19000	32500	29700	19100
10	Comparison	37	9	54	45	45	45	19	45	45	23	19	15
	Swaps	0	0	0	6	0	5	16	45	20	0	0	0
	Time	88900	14800	308000	409100	238800	869400	65500	2356100	448600	98400	650900	60200
100	Comparison	2415	99	5049	4950	4950	4950	789	4950	4950	550	356	316
	Swaps	0	0	0	96	0	50	388	4950	2450	0	0	0
	Time	5368400	654500	1424400	5690600	10863000	4912400	770900	36991600	1358400	1400800	159200	182000
1000	Comparison	248888	999	500499	499500	499500	499500	11241	499500	499500	8701	5044	4932
	Swaps	0	0	0	995	0	500	6201	499500	249500	0	0	0
	Time	48275100	99600	36627200	154504000	95984600	50086200	4814500	105896100	73152000	3558800	1185200	1491400
10000	Comparison	24892626	9999	50004999	49995000	49995000	49995000	147246	49995000	49995000	120496	69008	64608
	Swaps	0	0	0	9989	0	5000	88018	49995000	24995000	0	0	0
	Time	3560954600	681401	2482057900	14208356400	13935669300	10098771101	30575800	10925615400	7037434100	31186300	6134501	632100
100000	Comparison	2495717317	99999	500049999	704982704	704982704	704982704	2072464	4999950000	4999950000	1536066	853904	815024
	Swaps	0	0	0	99986	0	50000	1095730	4999950000	249950000	0	0	0

Insertion Sort:

Best Case (Sorted Array) takes the least time since it undergoes only (N-1) Shift Checks. (TC ≈ O(N))

Average Case (Random Input Array) takes on an average of the time of Best and Worst Cases. (TC ≈ O (N2 / 4) ≈ O(N2))

Worst Case (Reversed Array) undergoes the greatest number of checks and hence it takes the most time of all of cases.

No. of Shifts = (n-1) + (n-2) + (n-3) + …. 3 + 2 + 1

= (n2 – n) / 2;

TC ≈ O (N2 / 2) ≈ O (N2).

Selection Sort:

- **Worst Case Complexity:** $O(n^2)$

  If we want to sort in ascending order and the array is in descending order then, the worst case occurs.

- **Best Case Complexity:** $O(n^2)$

  It occurs when the array is already sorted

- **Average Case Complexity:** $O(n^2)$

  It occurs when the elements of the array are in jumbled order (neither ascending nor descending).

Quick Sort:

- **Worst Case Complexity:** $O(n^2)$

  It occurs when the pivot element picked is either the greatest or the smallest element.

- **Best Case Complexity:** O(n*log n)

It occurs when the pivot element is always the middle element or near to the middle element.

- **Average Case Complexity:** O(n*log n)
  It occurs when the above conditions do not occur.

Merge Sort:

Best Case Complexity: The merge sort algorithm has a best-case time complexity of O(n*log n) for the already sorted array.

Average Case Complexity: The average-case time complexity for the merge sort algorithm is O(n*log n), which happens when 2 or more elements are jumbled, i.e., neither in the ascending order nor in the descending order.

Worst Case Complexity: The worst-case time complexity is also O(n*log n), which occurs when we sort the descending order of an array into the ascending order.

Applications:

Insertion Sort:

When number of elements is small then insertion sort is very useful because of its time complexity O(n^2).

Insertion sort is also good for sorting linked list, when compared to other sorting algorithm like merge sort which needs extra space for merge operations

Selection Sort:

The selection sort is used when

- a small list is to be sorted
- cost of swapping does not matter
- checking of all the elements is compulsory

- cost of writing to memory matters like in flash memory (number of writes/swaps is O(n) as compared to O(n2) of bubble sort)

Quick Sort:

Quicksort algorithm is used when:

The programming language is good for recursion

Time complexity matters

Space complexity matters

Merge Sort:

Merge Sort is useful for sorting linked lists in O(n Log n) time.

Merge sort can be implemented without extra space for linked lists.

Merge sort is used for counting inversions in a list.

Merge sort is used in external sorting

References:

- ➢ Insertion sort
- ➢ Insertion Sort (Image)

- ➢ Selection Sort
- ➢ Selection Sort (Image)

- ➢ Quick Sort
- ➢ Quick Sort (Image)

- ➢ Merge Sort
- ➢ Merge Sort (Image)

Practical No. 3

Aim: -

Use singly linked lists to implement integers of unlimited size. Each node of the list should store one digit of the integer. You should implement addition, subtraction, multiplication, and exponentiation operations. Limit exponents to be positive integers.

Introduction: -

In computer programming, we often need to work with large numbers, which may not fit into the standard integer data types. To overcome this limitation, we can use singly linked lists, where each digit of the number is stored in a separate node of the list.

Algorithm: -

Addition: -

1. Define a function called "Add" that takes two LinkedLists of integers as input.
2. Create a new LinkedList called "ll" to hold the result of the addition.
3. Create two new LinkedLists called "l1" and "l2" as clones of the input LinkedLists "ll1" and "ll2", respectively.
4. If the first element of "l1" is negative and the first element of "l2" is non-negative, then negate the first element of "l1" and call the "Subtract" function with "l2" and "l1" as arguments. Store the result in "ll" and return it.
5. If the first element of "l1" is non-negative and the first element of "l2" is negative, then negate the first element of "l2" and call the "Subtract" function with "l1" and "l2" as arguments. Store the result in "ll" and return it.
6. Create variables "i", "j", and "carry" and set them equal to the size of "l1" minus one, the size of "l2" minus one, and 0, respectively.
7. While "i" and "j" are greater than or equal to 0, do the following:
    - Compute the sum of the absolute values of the "i"-th element of "l1", the "j"-th element of "l2", and "carry". Store the result in "sum".
    - Set "carry" equal to the integer division of "sum" by 10.
    - Set "sum" equal to the remainder of "sum" divided by 10.
    - Add "sum" to the beginning of "ll".
    - Decrement both "i" and "j".
8. While "i" is greater than or equal to 0, do the following:
    - Compute the sum of the absolute value of the "i"-th element of "l1" and "carry". Store the result in "sum".
    - Set "carry" equal to the integer division of "sum" by 10.
    - Set "sum" equal to the remainder of "sum" divided by 10.
    - Add "sum" to the beginning of "ll".

- Decrement "i".

9. While "j" is greater than or equal to 0, do the following:
    - Compute the sum of the absolute value of the "j"-th element of "l2" and "carry". Store the result in "sum".
    - Set "carry" equal to the integer division of "sum" by 10.
    - Set "sum" equal to the remainder of "sum" divided by 10.
    - Add "sum" to the beginning of "ll".
    - Decrement "j".
10. If "carry" is not equal to 0, then add it to the beginning of "ll".
11. If both the first element of "l1" and the first element of "l2" are negative, then negate the last element of "ll".
12. Return "ll".

Subtraction: -

1. Define a function called "Subtract" that takes two LinkedLists of integers as input.
2. Create a new LinkedList called "ll" to hold the result of the subtraction.
3. Create two new LinkedLists called "l1" and "l2" as clones of the input LinkedLists "ll1" and "ll2", respectively.
4. If the first element of "l1" is negative and the first element of "l2" is non-negative, then negate the first element of "l1" and call the "Add" function with "l1" and "l2" as arguments. Store the result in "ll" and return it.
5. If the first element of "l1" is non-negative and the first element of "l2" is negative, then negate the first element of "l2" and call the "Add" function with "l1" and "l2" as arguments. Store the result in "ll" and return it.
6. If both "l1" and "l2" start with a negative sign, remove the negative sign and call the "Subtract" function recursively with "l2" and "l1" as arguments. Negate the result before returning it.
7. If the length of "l2" is greater than the length of "l1", swap "l1" and "l2".
8. Create variables "i" and "j" and set them equal to the size of "l1" minus one and the size of "l2" minus one, respectively.
9. Create a variable called "borrow" and set it to 0.
10. While "i" and "j" are greater than or equal to 0, do the following:
    - If the "i"-th element of "l1" is greater than or equal to the "j"-th element of "l2" minus "borrow", subtract the "j"-th element of "l2" minus "borrow" from the "i"-th element of "l1" and store the result in "ll".
    - Otherwise, add 10 to the "i"-th element of "l1", subtract the "j"-th element of "l2" minus "borrow" from the result, and store the result in "ll". Set "borrow" to 1.
    - Decrement both "i" and "j".
11. While "i" is greater than or equal to 0, do the following:
    - If the "i"-th element of "l1" is greater than or equal to "borrow", subtract "borrow" from the "i"-th element of "l1" and store the result in "ll". Set "borrow" to 0.
    - Otherwise, add 10 to the "i"-th element of "l1" and subtract "borrow" from the result. Store the result in "ll". Set "borrow" to 1.
    - Decrement "i".
12. While the first element of "ll" is equal to 0 and "ll" is not of length 1, remove the first element of "ll".
13. If the first element of "l1" is negative, negate the first element of "ll".
14. Return "ll".

Multiplication: -

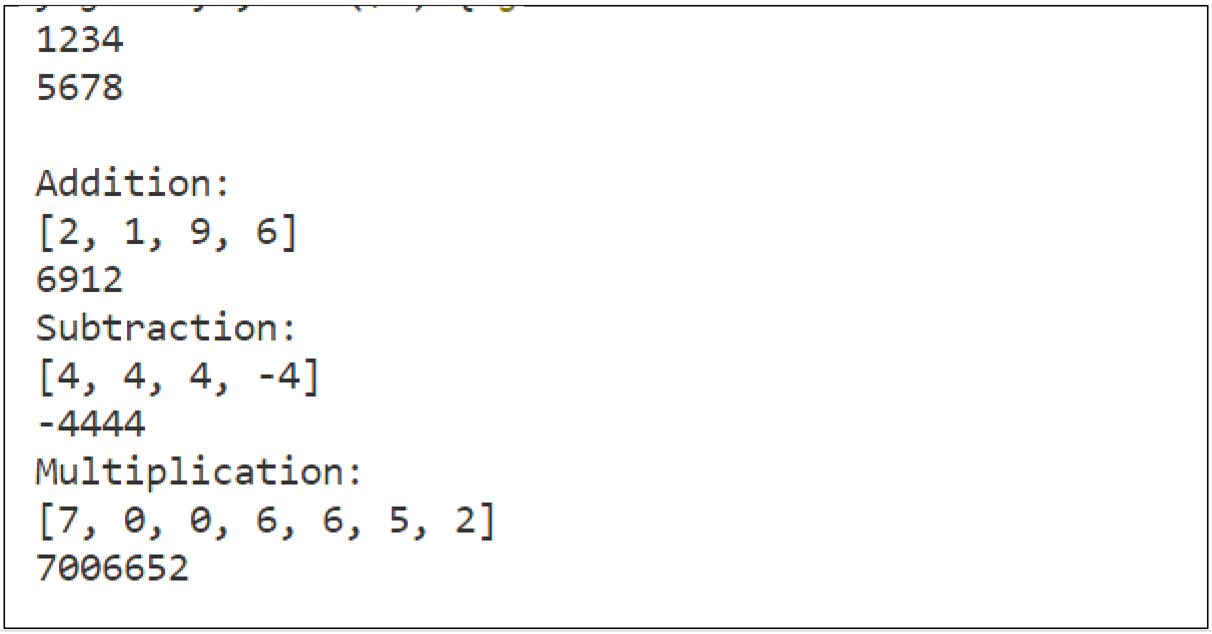
1. Define a function Multiply(LinkedList<Integer> ll1, LinkedList<Integer> ll2) that takes two linked lists of integers as inputs and returns a new linked list representing their product.
2. Create an empty linked list ll to hold the result.
3. Clone ll1 and ll2 into l1 and l2 using the clonell() function defined earlier to create deep copies of the input linked lists.
4. Initialize carry to zero.
5. For each digit d1 in l1, starting from the rightmost digit:
   a. Initialize a new linked list row to hold the current row of digits in the multiplication.
   b. Initialize carry to zero.
   c. For each digit d2 in l2, starting from the rightmost digit:
   i. Multiply d1 and d2 and add carry to the result.
   ii. Add the least significant digit of the result to the beginning of the row linked list and update carry to be the most significant digit.
   iii. Remove the least significant digit from the result.
   d. If carry is not zero, add it to the beginning of the row linked list.
   e. Add enough zeroes to the end of the row linked list to align it with the current digit position in l1.
   f. Add the row linked list to the ll linked list using the Add() function defined earlier.
6. If the first digit in l1 or l2 is negative, set the first digit in the result linked list ll to negative.
7. Remove any leading zeroes from the result linked list ll.
8. Return the result linked list ll.

Code: -

```java
import java.util.*;

public class CalculatorHighCapacity {

    public static LinkedList<Integer> clonell(LinkedList<Integer> list){
        LinkedList<Integer> newList = new LinkedList<>();
        Iterator<Integer> iterator = list.iterator();

        while (iterator.hasNext()) {
            Integer element = iterator.next();
            // Create a new object or copy the object here if it is not a
primitive type
            Integer newElement = Integer.valueOf(element.intValue());
            newList.add(newElement);
        }

        // newList now contains a deep copy of the originalList
        return newList;
    }
    public static LinkedList<Integer> Add(LinkedList<Integer> ll1,
LinkedList<Integer> ll2) {
        LinkedList<Integer> ll = new LinkedList<>();
        LinkedList<Integer> l1 = new LinkedList<Integer>();
        LinkedList<Integer> l2 = new LinkedList<Integer>();
        l1 = clonell(ll1);
        l2 = clonell(ll2);
        // (-l1)+(-l2) = -(l1 + l2)
        // if(l1.getFirst() < 0 && l2.getFirst() < 0){
        //      l1.set(0, -l1.getFirst());
        //      l2.set(0, -l2.getFirst());
        //      ll = Add(l1,l2);
        //      ll.set(ll.size() - 1, -ll.getLast());
        //      return ll;
        // }

        // (-l1)+(+l2) = (+l2)-(+l1)
        if(l1.getFirst() < 0 && l2.getFirst() >= 0){
            l1.set(0, -l1.getFirst());
            ll = Subtract(l2, l1);
            return ll;
        }
        // (+l1)+(-l2) = (+l1)-(+l2)
        else if(l1.getFirst() >= 0 && l2.getFirst() < 0){
            l2.set(0, -l2.getFirst());
            ll = Subtract(l1, l2);
            return ll;
        }
        int i = l1.size() - 1, j = l2.size() - 1, carry = 0;
        while (i >= 0 && j >= 0) {
            int sum = Math.abs(l1.get(i)) + Math.abs(l2.get(j)) + carry;
            carry = sum / 10;
            sum %= 10;
            ll.add(sum);
            i--;
            j--;
        }
        while (i >= 0) {
            int sum =  Math.abs(l1.get(i)) + carry;
            carry = sum / 10;
            sum = sum % 10;
            ll.add(sum);
            i--;
        }
        while (j >= 0) {
            int sum =  Math.abs(l2.get(j)) + carry;
            carry = sum / 10;
            sum = sum % 10;
            ll.add(sum);
            j--;
        }

        if (carry != 0) {
            int sum = carry;
            carry = sum / 10;
            sum = sum % 10;
            ll.add(sum);
        }
        if(l1.getFirst() < 0 && l2.getFirst() < 0){
            // String s = "-"+sum;
            // ll.add(Integer.parseInt(s));
            ll.set(ll.size() - 1, -ll.getLast());
        }

        return ll;
    }

    public static LinkedList<Integer> Subtract(LinkedList<Integer> ll1,
LinkedList<Integer> ll2) {
        LinkedList<Integer> ll = new LinkedList<>();
        LinkedList<Integer> l1 = new LinkedList<Integer>();
        LinkedList<Integer> l2 = new LinkedList<Integer>();
        l1 = clonell(ll1);
        l2 = clonell(ll2);
        // (+l1)-(-l2) = (+l1)+(+l2)
        if(l1.getFirst() >= 0 && l2.getFirst() < 0){
            l2.set(0, -l2.getFirst());
            ll = Add(l1, l2);
            return ll;
        }
        // (-l1)-(+l2) = (-l1)+(-l2)
        else if(l1.getFirst() < 0 && l2.getFirst() >= 0){
            l2.set(0, -l2.getFirst());
            ll = Add(l1, l2);
            return ll;
        }
        if(l1.size() < l2.size() || (l1.size()==l2.size() &&
Math.abs(l1.getFirst())<Math.abs(l2.getFirst()))){
            ll = Subtract(l2, l1);
            ll.set(ll.size()-1, -ll.getLast());
            return ll;
        }
        int i = l1.size() - 1, j = l2.size() - 1, borrow = 0;
        while (i >= 0 && j >= 0) {
            // int diff = Math.abs(Math.abs(l1.get(i)) - Math.abs(l2.get(j)))
- borrow;
            int diff = Math.abs(l1.get(i)) - Math.abs(l2.get(j)) - borrow;
            borrow = 0;
            if (diff < 0) {
                diff += 10;
                borrow = 1;
            }
            ll.add(diff);
            i--;
            j--;
        }

        while (i >= 0) {
            int diff = Math.abs(l1.get(i)) - borrow;
            borrow = 0;
            if (diff < 0) {
                diff += 10;
                borrow = 1;
            }
            ll.add(diff);
            i--;
        }

        while (j >= 0) {
            int diff = Math.abs(l2.get(j)) - borrow;
            borrow = 0;
            if (diff < 0) {
                diff += 10;
                borrow = 1;
            }
            ll.add(diff);
            j--;
        }
        //Remove the Zeros in start of the number after subtraction
        while (!ll.isEmpty() && ll.getLast() == 0) {
            ll.removeLast();
        }
        //if 0 is ans
        if (ll.isEmpty()) {
            ll.add(0);
        }

        if(l1.getFirst() < 0 && l2.getFirst() < 0){
            // String s = "-"+sum;
            // ll.add(Integer.parseInt(s));
            ll.set(ll.size() - 1, -ll.getLast());
        }
        // if(l1.size() == l2.size() && l1.getFirst()<l2.getFirst()){
        //     ll.set(ll.size()-1, -ll.getLast());
        // }
        // if(flag){
```

```
        //      ll.set(ll.size()-1, -ll.getLast());
        // }
        return ll;
    }

    public static LinkedList<Integer> Multiply(LinkedList<Integer> ll1,
LinkedList<Integer> ll2) {
        LinkedList<Integer> ll = new LinkedList<>();
        LinkedList<Integer> l1 = new LinkedList<Integer>();
        LinkedList<Integer> l2 = new LinkedList<Integer>();
        l1 = clonell(ll1);
        l2 = clonell(ll2);
        if(l1.getFirst() < 0 && l2.getFirst() < 0){
            l1.set(0, -l1.getFirst());
            l2.set(0, -l2.getFirst());
            ll = Multiply(l1, l2);
            return ll;
        }
        else if(l1.getFirst() < 0 && l2.getFirst() >= 0){
            l1.set(0, -l1.getFirst());
            ll = Multiply(l1, l2);
            ll.set(0, -ll.getFirst());
            return ll;
        }
        else if(l1.getFirst() >= 0 && l2.getFirst() < 0){
            l2.set(0, -l2.getFirst());
            ll = Multiply(l1, l2);
            ll.set(0, -ll.getFirst());
            return ll;
        }
        for (int i = 0; i < ll1.size() + ll2.size(); i++) {
            ll.add(0);
        }
        for (int i = ll1.size() - 1; i >= 0; i--) {
            int carry = 0;
            for (int j = ll2.size() - 1; j >= 0; j--) {
                int product = ll1.get(i) * ll2.get(j) + carry + ll.get(i + j
+ 1);

                carry = product / 10;
                ll.set(i + j + 1, product % 10);
            }
            ll.set(i, carry);
        }
        while (ll.size() > 1 && ll.getFirst() == 0) {
            ll.removeFirst();
        }
        return ll;
    }
    public static void iterateListReverse(LinkedList<Integer> linkedList) {
        for (int i = linkedList.size() - 1; i >= 0; i--) {
            System.out.print(linkedList.get(i));
        }
        System.out.println();
    }
    public static void iterateList(LinkedList<Integer> linkedList) {
        for (int i = 0; i < linkedList.size(); i++) {
            System.out.print(linkedList.get(i));
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        LinkedList<Integer> ll1 = new LinkedList<>();
        LinkedList<Integer> ll2 = new LinkedList<>();
        String num1 = sc.next();
        String num2 = sc.next();
        String s1;
        if(num1.charAt(0) == '-'){
            s1= ""+num1.charAt(0)+num1.charAt(1);
            ll1.add(Integer.parseInt(s1));
            for(int i=2; i<num1.length(); i++){
                ll1.add(Integer.valueOf(Character.getNumericValue(num1.charAt
(i))));
            }
        }
        else{
            s1= ""+num1.charAt(0);
            ll1.add(Integer.parseInt(s1));
            for(int i=1; i<num1.length(); i++){
                ll1.add(Integer.valueOf(Character.getNumericValue(num1.charAt
(i))));
            }
        }

        String s2;
        if(num2.charAt(0) == '-'){
            s2= ""+num2.charAt(0)+num2.charAt(1);
            ll2.add(Integer.parseInt(s2));
            for(int i=2; i<num2.length(); i++){
                ll2.add(Integer.valueOf(Character.getNumericValue(num2.charAt
(i))));
            }
        }
        else{
            s2= ""+num2.charAt(0);
            ll2.add(Integer.parseInt(s2));
            for(int i=1; i<num2.length(); i++){
                ll2.add(Integer.valueOf(Character.getNumericValue(num2.charAt
(i))));
            }
        }

        System.out.println();

        // iterateList(ll1);
        // iterateList(ll2);
        System.out.println("Addition: ");
        LinkedList<Integer> add = new LinkedList<>();
        add = Add(ll1, ll2);
        System.out.println(add);
        iterateListReverse(add);

        System.out.println("Subtraction: ");
        LinkedList<Integer> sub = new LinkedList<>();
        sub = Subtract(ll1, ll2);
        System.out.println(sub);
        iterateListReverse(sub);

        System.out.println("Multiplication: ");
        LinkedList<Integer> mul = new LinkedList<>();
        mul = Multiply(ll1, ll2);
        System.out.println(mul);
        iterateList(mul);
    }
}
```

Output: -



```
1234
5678

Addition:
[2, 1, 9, 6]
6912
Subtraction:
[4, 4, 4, -4]
-4444
Multiplication:
[7, 0, 0, 6, 6, 5, 2]
7006652
```

Applications: -

The application of the above aim can be found in various areas of computer science, such as cryptography, scientific computing, and data processing.

In cryptography, large prime numbers are used to generate secure keys for encryption and decryption. These prime numbers can be represented using linked lists, and the arithmetic operations implemented in this project can be used to manipulate these large numbers.

In scientific computing, researchers often deal with very large datasets that require working with large numbers. Using linked lists to represent these numbers can help improve the efficiency of operations such as multiplication and exponentiation.

Furthermore, the ability to work with arbitrarily large numbers can also be useful in data processing applications, such as those involving financial calculations or statistical analysis.

Overall, the application of this project can be found in many areas of computer science, where large numbers are required, and the ability to work with them efficiently is essential.

Analysis: -

Addition: To add two integers represented as linked lists, we start from the leastsignificant digit of both numbers and add them along with any carry from the previous digit. We continue this process until we reach the end of both lists. If one list is longer than the other, we assume that the missing digits are 0s. **The asymptotic running time of addition is O(n), where n is the length of the longer of the two lists.**

Subtraction: Subtraction is similar to addition, but we subtract the digits insteadof adding them. We also need to handle borrowing from the next digit when a digit in the minuend is smaller than the corresponding digit in the subtrahend. **The asymptotic running time of subtraction is also O(n), where n is the length of the longer of the two lists.**

Multiplication: To multiply two integers represented as linked lists, we use the standard multiplication algorithm taught in elementary school, which involves multiplying each digit in one number with every digit in the other number and adding the results. **The asymptotic running time of multiplication is O(n^2), where n is the length of the longer of the two lists.**

References: -

https://www.javatpoint.com/application-of-linked-list

https://chat.openai.com/chat

Practical No. 4

Aim: -

Implement a city database using unordered lists. Each database record contains the name of the city (a string of arbitrary length) and the coordinates of the city expressed as integer x and y coordinates. Your program should allow following functionalities:

a) Insert a record,
b) Delete a record by name or coordinate,
c) Search a record by name or coordinate.
d) Pint all records within a given distance of a specified point.

Implement the database using an array-based list implementation, and then a linked list implementation.

Introduction: -

Array-based List Implementation:

An array-based list implementation stores data in a contiguous block of memory, allowingfast access to any element based on its index. For example, if we have an array of 100 elements and want to access the 50th element, we can do so in constant time.

Advantages:

Fast access to elements based on index Efficient memory usage since we can pre-allocate a block of memory for the entire list Goodfor small lists with a fixed size

Disadvantages:

Inserting or deleting elements requires shifting all subsequent elements, which can be slowfor large lists Limited flexibility in terms of resizing the list

Linked List Implementation:

A linked list implementation stores data in nodes that are connected to each other through pointers. Each node contains a value and a pointer to the next node, allowing us to traversethe list from one node to the next.

Advantages:

Efficient insertion and deletion since we only need to update the pointers of neighboringnode

Flexibility in terms of resizing the list since we can dynamically allocate new nodes asneeded Can handle large lists without worrying about memory allocation

Disadvantages:

Accessing elements based on index is slow since we need to traverse the list from thebeginning to reach a specific element requires extra memory for the pointers, which can be a concern for large lists

Algorithm:

Create a data structure to hold city records. Each record will contain the followingfields:

city name (string)

x-coordinate (integer)

y-coordinate (integer)

Create an empty list to store the city records.

Create a function to insert a new city record into the list. This function should take asinput the city name, x-coordinate, and y-coordinate, and should create a new record with those values and append it to the list.

Create a function to delete a city record from the list. This function should take as input either the name or coordinates of the city to be deleted, and should search thelist for a record with matching values. If a match is found, the record should be removed from the list.

Create a function to search for a city record in the list. This function should take asinput either the name or coordinates of the city to be searched for, and should search the list for a record with matching values. If a match is found, the record should be returned.

Create a function to print all records within a given distance of a specified point. Thisfunction should take as input the x-coordinate, y-coordinate, and radius of the point,and should iterate through the list of city records. For each record, calculate the distance between its coordinates and the specified point using the distance formula:

distance = sqrt((x2 - x1)^2 + (y2 - y1)^2) If the distance is less than or equal to the specified radius, print the record.

Test the program by calling the insert, delete, search, and print functions with variousinputs

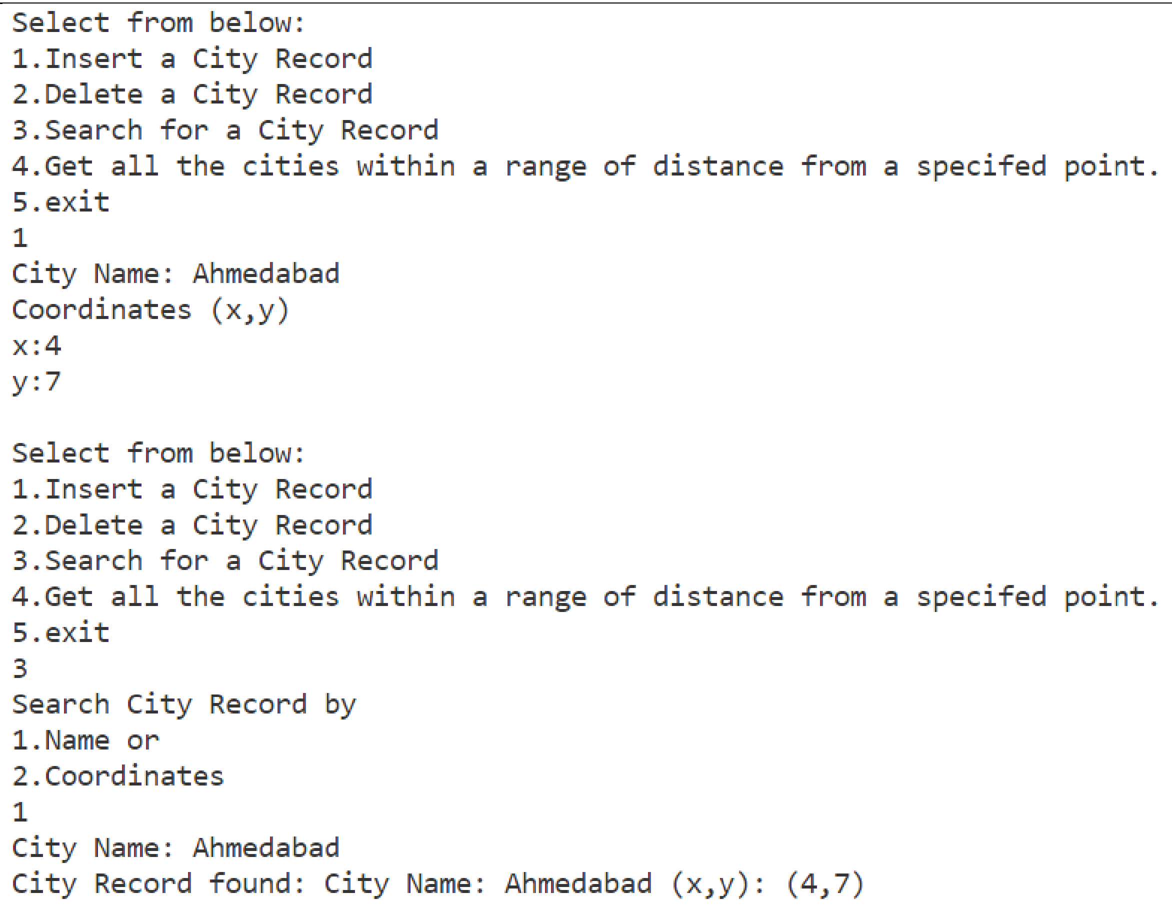
Array Based

Code: -

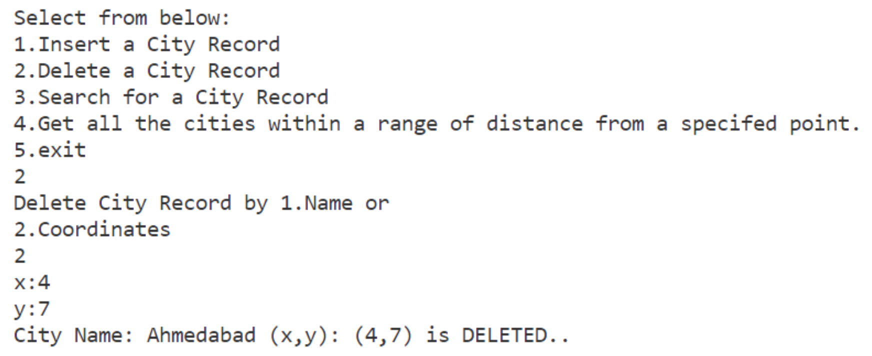
```java
import java.util.*;

class CityRecord {
    public String cityname;
    public int x;
    public int y;
    CityRecord(String city, int X, int Y) {
        cityname = city;
        x = X;
        y = Y;
    }
    @Override
    public String toString() {
        return "City Name: "+cityname+" (x,y): ("+x+","+y+")";
    }
}
class CityDatabase {

    public ArrayList<CityRecord> records;

    public CityDatabase() {
        records = new ArrayList<>();
    }

    public void insertRecord(String city, int x, int y) {
        records.add(new CityRecord(city, x, y));
    }

    public void deleteRecord(String city) {
        for (CityRecord cityRecord : records){
            if(cityRecord.cityname.equalsIgnoreCase(city)) {
                System.out.println(cityRecord+" is DELETED..\n");
                records.remove(cityRecord);
                return;
            }
        }
        // for (int i = 0; i < records.size(); i++) {
        //     if (records.get(i).cityname.equals(city)) {
        //         System.out.println(records.get(i)+" is DELETED..\n");
        //         records.remove(i);
        //         return;
        //     }
        // }
    }
    public void deleteRecord(int x, int y) {
        for (CityRecord cityRecord : records){
            if(cityRecord.x == x && cityRecord.y == y) {
                System.out.println(cityRecord+" is DELETED..\n");
                records.remove(cityRecord);
                return;
            }
        }
        // for (int i = 0; i < records.size(); i++) {
        //     if (records.get(i).x == x && records.get(i).y == y) {
        //         System.out.println(records.get(i)+" is DELETED..\n");
        //         records.remove(i);
        //         return;
        //     }
        // }
    }

    public CityRecord searchRecord(String cityName) {
        for (CityRecord record : records) {
            if (record.cityname.equals(cityName)) {
                return record;
            }
        }
        return null;
    }

    public CityRecord searchRecord(int x, int y) {
        for (CityRecord record : records) {
            if (record.x == x && record.y == y) {
                return record;
            }
        }
        return null;
    }

    public ArrayList<CityRecord> getRecordsWithinDistance(int x, int y, int
distance) {
        ArrayList<CityRecord> result = new ArrayList<>();
        for (CityRecord record : records) {
            if (Math.sqrt(Math.pow(record.x - x, 2) + Math.pow(record.y - y,
2)) <= distance) {
                result.add(record);
            }
        }
        return result;
    }
}

public class Lab5DistanceArrayList {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        CityDatabase citydb = new CityDatabase();
        String city;
        int X;
        int Y;
        while(true){
            System.out.println("Select from below:");
            System.out.println("1.Insert a City Record\n2.Delete a City
Record\n3.Search for a City Record\n4.Get all the cities within a range of
distance from a specifed point.");
            System.out.println("5.exit");
            int choice = sc.nextInt();
            switch (choice) {
                case 1:
                    sc.nextLine();
                    System.out.print("City Name: ");
                    city = sc.nextLine();
                    System.out.println("Coordinates (x,y)");
                    System.out.print("x:");
                    X = sc.nextInt();
                    System.out.print("y:");
                    Y = sc.nextInt();
                    citydb.insertRecord(city, X, Y);
                    break;
                case 2:
                    System.out.println("Delete City Record by 1.Name or
\n2.Coordinates");
                    int innerChoice = sc.nextInt();
                    switch (innerChoice) {
                        case 1:
                            sc.nextLine();
                            System.out.print("City Name: ");
                            city = sc.nextLine();
                            citydb.deleteRecord(city);
                            break;
                        case 2:
                            System.out.print("x:");
                            X = sc.nextInt();
                            System.out.print("y:");
                            Y = sc.nextInt();
                            citydb.deleteRecord(X, Y);
                            break;
                        default:
                            System.out.println("Invalid input");
                            break;
                    }
                    break;
                case 3:
                    System.out.println("Search City Record by \n1.Name or
\n2.Coordinates");
                    int innerChoice2 = sc.nextInt();
                    CityRecord searchedCity;
                    switch (innerChoice2) {
                        case 1:
                            sc.nextLine();
                            System.out.print("City Name: ");
                            city = sc.nextLine();
                            searchedCity = citydb.searchRecord(city);
                            if(searchedCity == null){
                                System.out.println("City Record not Found");
                            }
                            else{
                                System.out.println("City Record found:
"+searchedCity);
                            }
                            break;
                        case 2:
                            System.out.println("Coordinates (x,y): ");
                            System.out.print("x:");
                            X = sc.nextInt();
                            System.out.print("y:");
                            Y = sc.nextInt();
                            searchedCity = citydb.searchRecord(X,Y);
                            if(searchedCity == null){
                                System.out.println("City Record not Found");
                            }
                            else{
                                System.out.println("City Recorde found:
"+searchedCity);
                            }
                            break;
                        default:
                            System.out.println("Invalid input");
                            break;
                    }
```

```
                        break;
                    case 4:
                        System.out.println("Coordinates (x,y)");
                        System.out.print("x:");
                        X = sc.nextInt();
                        System.out.print("y:");
                        Y = sc.nextInt();
                        System.out.print("Distance: ");
                        int distance = sc.nextInt();
                        System.out.println("Cities found: ");
                        ArrayList<CityRecord> result =
citydb.getRecordsWithinDistance(X, Y, distance);
                        for(CityRecord c : result) {
                            System.out.println(c);
                        }
                        System.out.println();
                        break;
                    case 5:
                        return;
                    default:
                        System.out.println("Invalid Input...exiting");
                        return;
                }
                System.out.println();
            }
        }
}
```

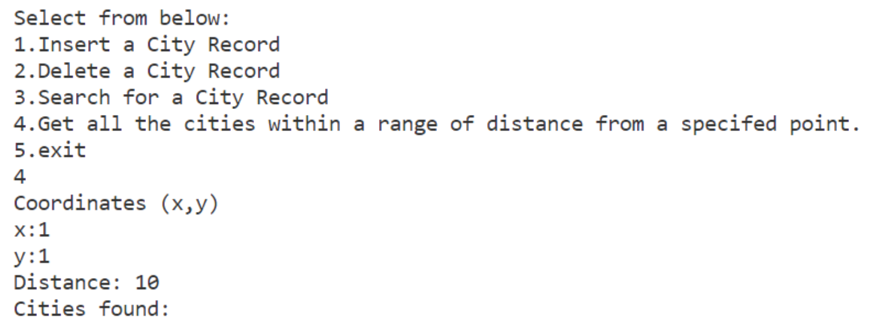
Output: -



```
Select from below:
1.Insert a City Record
2.Delete a City Record
3.Search for a City Record
4.Get all the cities within a range of distance from a specifed point.
5.exit
1
City Name: Ahmedabad
Coordinates (x,y)
x:4
y:7

Select from below:
1.Insert a City Record
2.Delete a City Record
3.Search for a City Record
4.Get all the cities within a range of distance from a specifed point.
5.exit
3
Search City Record by
1.Name or
2.Coordinates
1
City Name: Ahmedabad
City Record found: City Name: Ahmedabad (x,y): (4,7)
```



```
Select from below:
1.Insert a City Record
2.Delete a City Record
3.Search for a City Record
4.Get all the cities within a range of distance from a specifed point.
5.exit
2
Delete City Record by 1.Name or
2.Coordinates
2
x:4
y:7
City Name: Ahmedabad (x,y): (4,7) is DELETED..
```



```
Select from below:
1.Insert a City Record
2.Delete a City Record
3.Search for a City Record
4.Get all the cities within a range of distance from a specifed point.
5.exit
4
Coordinates (x,y)
x:1
y:1
Distance: 10
Cities found:
```

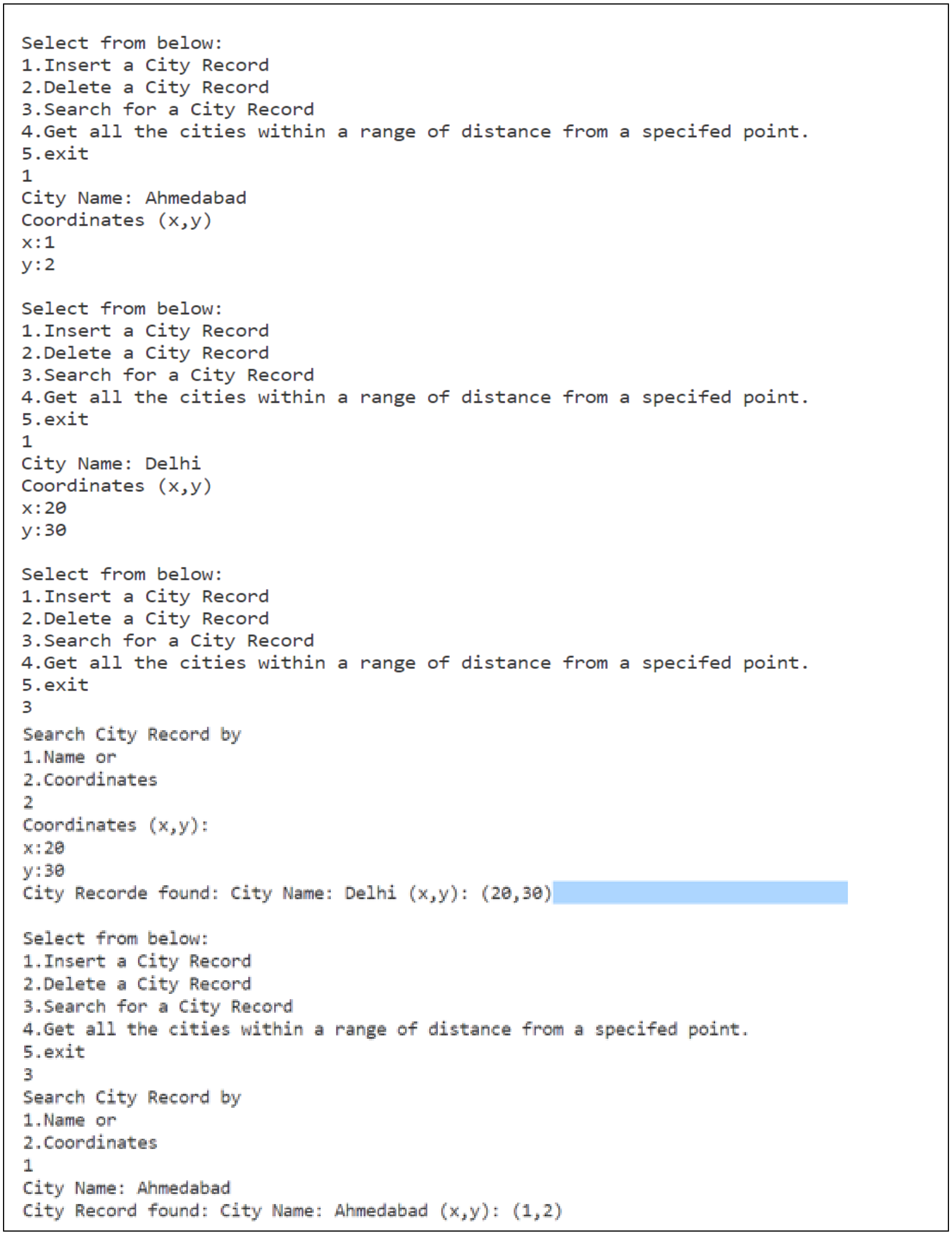
LinkedList Based

Code: -

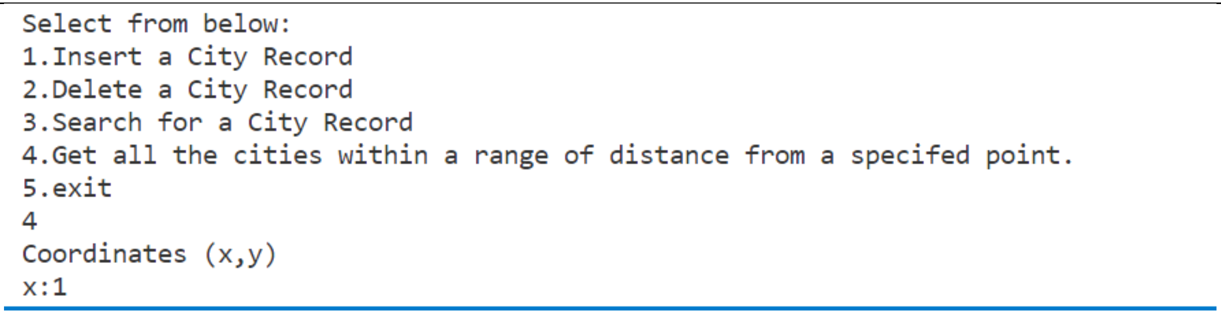
```java
import java.util.*;

class CityRecord {
    public String cityname;
    public int x;
    public int y;
    CityRecord(String city, int X, int Y) {
        cityname = city;
        x = X;
        y = Y;
    }
    @Override
    public String toString() {
        return "City Name: "+cityname+" (x,y): ("+x+","+y+")";
    }
}

class CityDatabase {

    public LinkedList<CityRecord> records;

    public CityDatabase() {
        records = new LinkedList<>();
    }

    public void insertRecord(String city, int x, int y) {
        records.add(new CityRecord(city, x, y));
    }

    public void deleteRecord(String city) {
        for (CityRecord cityRecord : records){
            if(cityRecord.cityname.equalsIgnoreCase(city)) {
                System.out.println(cityRecord+" is DELETED..\n");
                records.remove(cityRecord);
                return;
            }
        }
        // for (int i = 0; i < records.size(); i++) {
        //     if (records.get(i).cityname.equals(city)) {
```

```
        //          System.out.println(records.get(i)+" is DELETED..\n");
        //          records.remove(i);
        //          return;
        //      }
        // }
    }

    public void deleteRecord(int x, int y) {
        for (CityRecord cityRecord : records){
            if(cityRecord.x == x && cityRecord.y == y) {
                System.out.println(cityRecord+" is DELETED..\n");
                records.remove(cityRecord);
                return;
            }
        }
        // for (int i = 0; i < records.size(); i++) {
        //     if (records.get(i).x == x && records.get(i).y == y) {
        //         System.out.println(records.get(i)+" is DELETED..\n");
        //         records.remove(i);
        //         return;
        //     }
        // }
    }

    public CityRecord searchRecord(String cityName) {
        for (CityRecord record : records) {
            if (record.cityname.equals(cityName)) {
                return record;
            }
        }
        return null;
    }

    public CityRecord searchRecord(int x, int y) {
        for (CityRecord record : records) {
            if (record.x == x && record.y == y) {
                return record;
            }
        }
        return null;
    }

    public LinkedList<CityRecord> getRecordsWithinDistance(int x, int y, int
distance) {
        LinkedList<CityRecord> result = new LinkedList<>();
        for (CityRecord record : records) {
            if (Math.sqrt(Math.pow(record.x - x, 2) + Math.pow(record.y - y,
2)) <= distance) {
                result.add(record);
            }
        }
        return result;
    }
}

public class Lab5DistanceLinkedList {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        CityDatabase citydb = new CityDatabase();
        String city;
        int X;
        int Y;
        while(true){
            System.out.println("Select from below:");
            System.out.println("1.Insert a City Record\n2.Delete a City
Record\n3.Search for a City Record\n4.Get all the cities within a range of
distance from a specifed point.");
            System.out.println("5.exit");
            int choice = sc.nextInt();
            switch (choice) {
                case 1:
                    sc.nextLine();
                    System.out.print("City Name: ");
                    city = sc.nextLine();
                    System.out.println("Coordinates (x,y)");
                    System.out.print("x:");
                    X = sc.nextInt();
                    System.out.print("y:");
                    Y = sc.nextInt();
                    citydb.insertRecord(city, X, Y);
                    break;
                case 2:
                    System.out.println("Delete City Record by 1.Name or
\n2.Coordinates");
                    int innerChoice = sc.nextInt();
                    switch (innerChoice) {
                        case 1:
                            sc.nextLine();
                            System.out.print("City Name: ");
                            city = sc.nextLine();
                            citydb.deleteRecord(city);
                            break;
                        case 2:
                            System.out.print("x:");
                            X = sc.nextInt();
                            System.out.print("y:");
                            Y = sc.nextInt();
                            citydb.deleteRecord(X, Y);
                            break;
                        default:
                            System.out.println("Invalid input");
                            break;
                    }
                    break;
                case 3:
                    System.out.println("Search City Record by \n1.Name or 
\n2.Coordinates");
                    int innerChoice2 = sc.nextInt();
                    CityRecord searchedCity;
                    switch (innerChoice2) {
                        case 1:
                            sc.nextLine();
                            System.out.print("City Name: ");
                            city = sc.nextLine();
                            searchedCity = citydb.searchRecord(city);
                            if(searchedCity == null){
                                System.out.println("City Record not Found");
                            }
                            else{
                                System.out.println("City Record found: 
"+searchedCity);
                            }
                            break;
                        case 2:
                            System.out.println("Coordinates (x,y): ");
                            System.out.print("x:");
                            X = sc.nextInt();
                            System.out.print("y:");
                            Y = sc.nextInt();
                            searchedCity = citydb.searchRecord(X,Y);
                            if(searchedCity == null){
                                System.out.println("City Record not Found");
                            }
                            else{
                                System.out.println("City Recorde found: 
"+searchedCity);
                            }
                            break;
                        default:
                            System.out.println("Invalid input");
                            break;
                    }
                    break;
                case 4:
                    System.out.println("Coordinates (x,y)");
                    System.out.print("x:");
                    X = sc.nextInt();
                    System.out.print("y:");
                    Y = sc.nextInt();
                    System.out.print("Distance: ");
                    int distance = sc.nextInt();
                    System.out.println("Cities found: ");
                    LinkedList<CityRecord> result = 
citydb.getRecordsWithinDistance(X, Y, distance);
                    for(CityRecord c : result) {
                        System.out.println(c);
                    }
                    System.out.println();
                    break;
                case 5:
                    return;
                default:
                    System.out.println("Invalid Input...exiting");
                    return;
            }
            System.out.println();
        }
    }
}
```

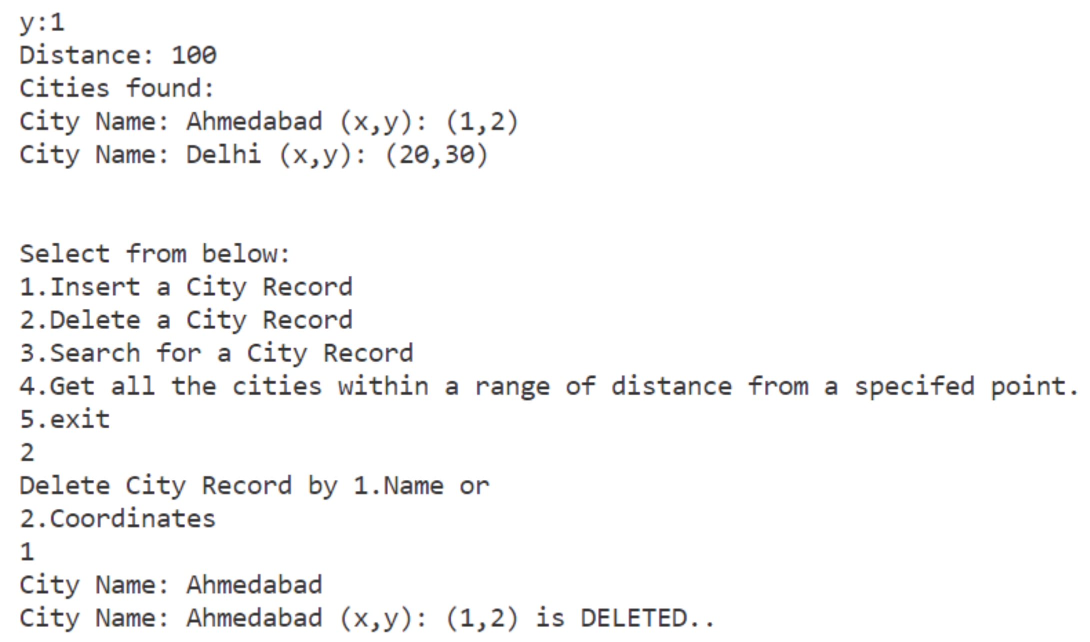
Output: -



```
Select from below:
1.Insert a City Record
2.Delete a City Record
3.Search for a City Record
4.Get all the cities within a range of distance from a specifed point.
5.exit
1
City Name: Ahmedabad
Coordinates (x,y)
x:1
y:2

Select from below:
1.Insert a City Record
2.Delete a City Record
3.Search for a City Record
4.Get all the cities within a range of distance from a specifed point.
5.exit
1
City Name: Delhi
Coordinates (x,y)
x:20
y:30

Select from below:
1.Insert a City Record
2.Delete a City Record
3.Search for a City Record
4.Get all the cities within a range of distance from a specifed point.
5.exit
3
Search City Record by
1.Name or
2.Coordinates
2
Coordinates (x,y):
x:20
y:30
City Recorde found: City Name: Delhi (x,y): (20,30)

Select from below:
1.Insert a City Record
2.Delete a City Record
3.Search for a City Record
4.Get all the cities within a range of distance from a specifed point.
5.exit
3
Search City Record by
1.Name or
2.Coordinates
1
City Name: Ahmedabad
City Record found: City Name: Ahmedabad (x,y): (1,2)
```



```
Select from below:
1.Insert a City Record
2.Delete a City Record
3.Search for a City Record
4.Get all the cities within a range of distance from a specifed point.
5.exit
4
Coordinates (x,y)
x:1
```



```
y:1
Distance: 100
Cities found:
City Name: Ahmedabad (x,y): (1,2)
City Name: Delhi (x,y): (20,30)


Select from below:
1.Insert a City Record
2.Delete a City Record
3.Search for a City Record
4.Get all the cities within a range of distance from a specifed point.
5.exit
2
Delete City Record by 1.Name or
2.Coordinates
1
City Name: Ahmedabad
City Name: Ahmedabad (x,y): (1,2) is DELETED..
```

Experiment No. 5 [Greedy Approach]

Aim: -

Implement interval scheduling algorithm. Given n events with their starting and ending times, find a schedule that includes as many events as possible. It is not possible to select an event partially. For example, consider the following example:

Event	Starting time	Ending time
A	1	3
B	2	5
C	3	9
D	6	8

Here, maximum number of events that can be scheduled is 2. We can schedule B and D together.

Introduction: -

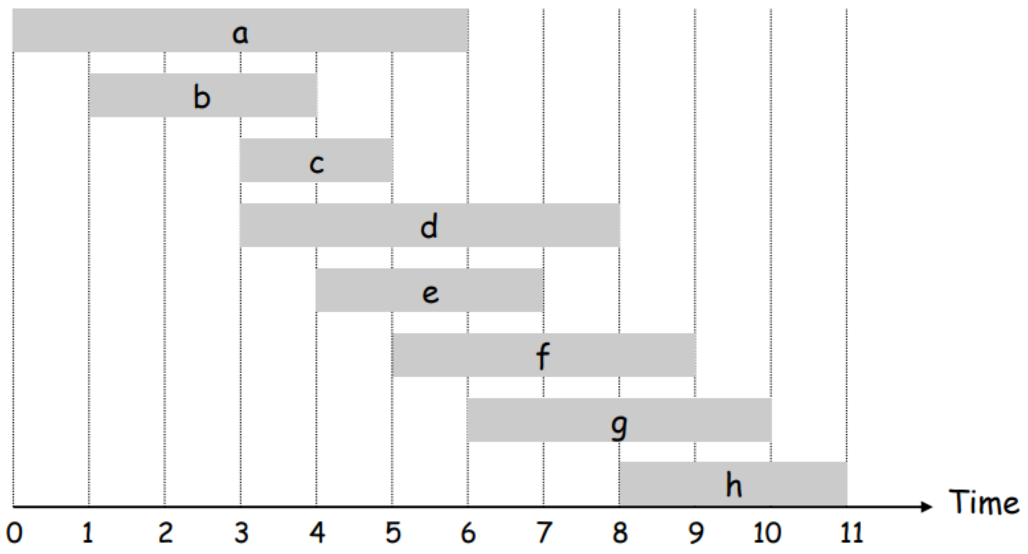
 Job j starts at sj and finishes at fj.

 Two jobs compatible if they don't overlap.

 Goal: find maximum subset of mutually compatible jobs.

Approach:

[Earliest finish time] Consider jobs in ascending order of finish time fj.



ALGORITHM:

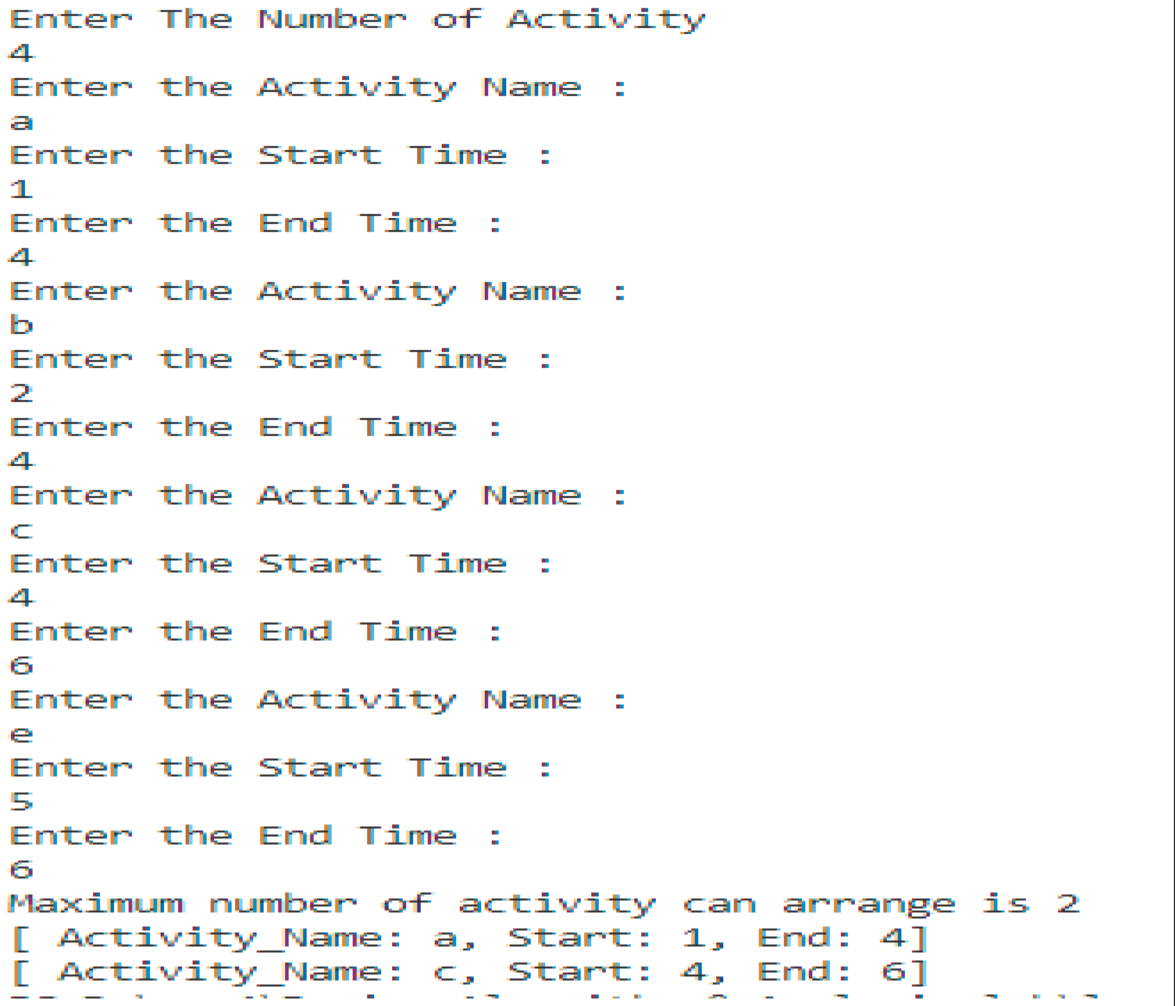
```
Algorithm Interval Partition {
Sort all intervals by finish time in ascending order
While there are intervals left {
    Let i be the next one
    If there is an existing classroom whose schedule is compatible with i {
        Add i to the compatible classroom that has the earliest
        finishing time
    }
    Else {
        Create a new classroom and add i to it
    }
}
```

Code: -

```
import java.util.Scanner;
import java.util.*;

class Activity {
    public int st;
    public int et;
    public String Activity_Name;

    public Activity() {
        this.Activity_Name = Activity_Name;
        this.st = st;
        this.et = et;
    }

    @Override
    public String toString() {
        return "[ Activity_Name: " + Activity_Name +", Start: " + st + ", End: 
" + et + "]";
    }

}

public class lab8_activitymanage {
```

```java
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int numberOfActivity = 0;
        System.out.println("Enter The Number of Activity");
        numberOfActivity = sc.nextInt();

        ArrayList<Activity> Schedule = new ArrayList<Activity>();

        for (int i = 0; i < numberOfActivity; i++) {
            Activity a = new Activity();

            System.out.println("Enter the Activity Name : ");
            sc.nextLine();
            a.Activity_Name = sc.nextLine();

            System.out.println("Enter the Start Time : ");
            a.st = sc.nextInt();

            System.out.println("Enter the End Time : ");
            a.et = sc.nextInt();
            if (a.et < a.st) {
                System.out.println("Enter correct number!!! ");
                return;
            }
            Schedule.add(a);
        }

        Schedule.sort(new Comparator<Activity>() {
            // compares Activity
            public int compare(Activity a1, Activity a2) {
                return a1.et - a2.et;
            }
        });

        ArrayList<Activity> TimeTable = new ArrayList<Activity>();
        int lastActivity = 0;
        for (Activity a : Schedule) {
            if (lastActivity <= a.st) {
                TimeTable.add(a);
                lastActivity = a.et;
            }
        }
        System.out.println("Maximum number of activity can arrange is "+TimeTable.size());
        for (Activity a : TimeTable) {
            System.out.println(a);
        }
    }
}
```

Output: -



```
Enter The Number of Activity
4
Enter the Activity Name :
a
Enter the Start Time :
1
Enter the End Time :
4
Enter the Activity Name :
b
Enter the Start Time :
2
Enter the End Time :
4
Enter the Activity Name :
c
Enter the Start Time :
4
Enter the End Time :
6
Enter the Activity Name :
e
Enter the Start Time :
5
Enter the End Time :
6
Maximum number of activity can arrange is 2
[ Activity_Name: a, Start: 1, End: 4]
[ Activity_Name: c, Start: 4, End: 6]
```

Applications:

Interval scheduling is a scheduling problem where a set of tasks need to be scheduled in such a way that no two tasks overlap. The goal is to find the maximum number of non-overlapping tasks that can be scheduled. Some applications of interval scheduling include:

1. Job scheduling: In a manufacturing plant, a set of jobs may need to be scheduled on different machines. The machines may be available for different intervals of time, and the goal is to find the maximum number of jobs that can be scheduled without overlapping.
2. Course scheduling: In a university, a set of courses may need to be scheduled in different time slots. The courses may have different durations, and the goal is to find the maximum number of courses that can be scheduled without overlapping.
3. Flight scheduling: In an airport, a set of flights may need to be scheduled on different runways. The runways may be available for different intervals of time, and the goal is to find the maximum number of flights that can be scheduled without overlapping.

References:

https://chat.openai.com/chat

https://www.geeksforgeeks.org/activity-selection-problem-greedy-algo-1/

Practical No. 6 [Divide and Conquer]

Aim: -

Implement both a standard $O(n^3)$matrix multiplication algorithm and Strassen's matrixmultiplication algorithm. Using empirical testing, try and estimate the constant factors for the runtimeequations of the two algorithms. How big must $\boldsymbol{n}$ be before Strassen's algorithm becomes more efficient than the standard algorithm?

Introduction: -

the aim of this task is to implement both a standard $O(n3)$ matrix multiplication algorithm and Strassen's matrix multiplication algorithm, and then compare their efficiency by estimating the constant factors for their runtime equations. Through empirical testing, we have determined how large n must be before Strassen's algorithm becomes more efficient than the standard algorithm.

Strassen's Matrix multiplication can be performed only on **square matrices** where **n** is a **power of 2**. Order of both of the matrices are **n × n**.

Divide **X**, **Y** and **Z** into four (n/2)×(n/2) matrices as represented below −

$$Z = \begin{bmatrix} I & J \\ K & L \end{bmatrix} \qquad X = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \text{ and } \qquad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

Using Strassen's Algorithm compute the following −

$$M_1 := (A+C) \times (E+F)$$

$$M_2 := (B+D) \times (G+H)$$

$$M_3 := (A-D) \times (E+H)$$

$$M_4 := A \times (F-H)$$

$$M_5 := (C+D) \times (E)$$

$$M_6 := (A+B) \times (H)$$

$$M_7 := D \times (G-E)$$

Then,

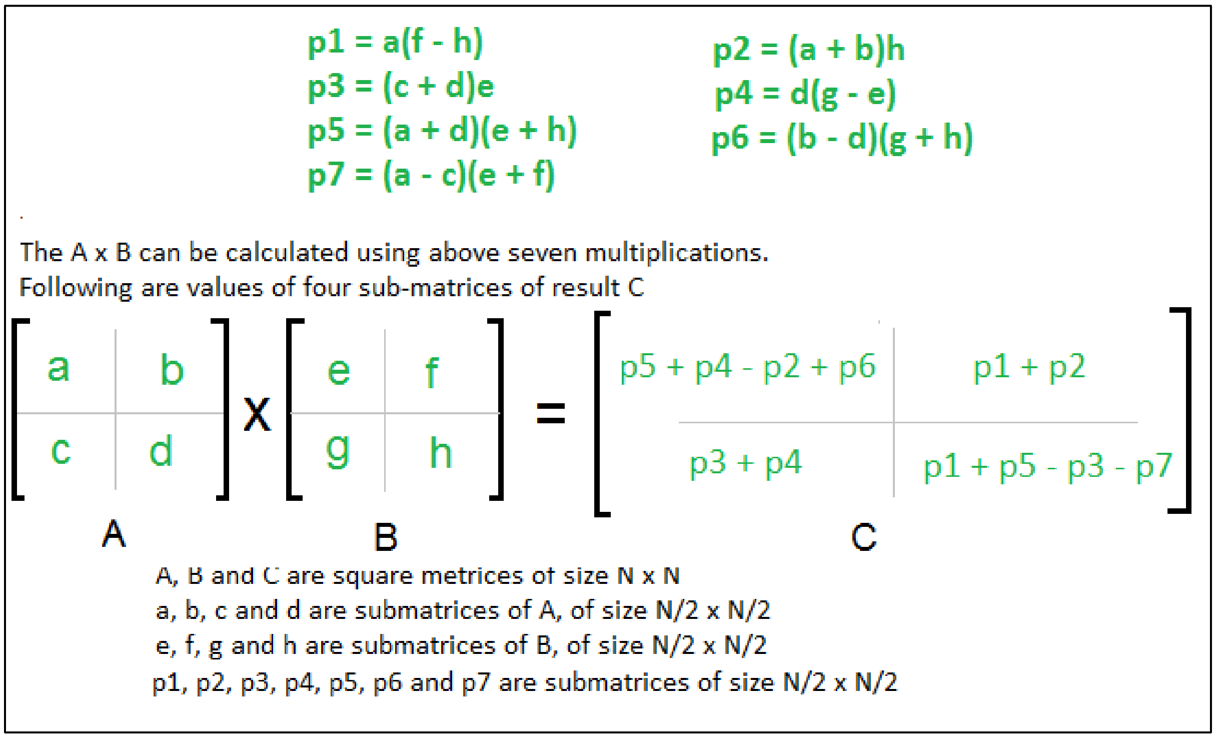
$$I := M_2 + M_3 - M_6 - M_7$$

$$J := M_4 + M_6$$

$$K := M_5 + M_7$$

$$L := M_1 - M_3 - M_4 - M_5$$

Strassen’s method is similar to divide and conquer method in the sense that this method also divide matrices to sub-matrices of size N/2 x N/2 as shown in the above diagram, but in Strassen’s method, the four sub-matrices of result are calculated using following formulae.



p1 = a(f - h)
p2 = (a + b)h
p3 = (c + d)e
p4 = d(g - e)
p5 = (a + d)(e + h)
p6 = (b - d)(g + h)
p7 = (a - c)(e + f)

The A x B can be calculated using above seven multiplications.
Following are values of four sub-matrices of result C

$$\underbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}_{A} \times \underbrace{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}_{B} = \underbrace{\begin{bmatrix} p5 + p4 - p2 + p6 & p1 + p2 \\ p3 + p4 & p1 + p5 - p3 - p7 \end{bmatrix}}_{C}$$

A, B and C are square metrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2
p1, p2, p3, p4, p5, p6 and p7 are submatrices of size N/2 x N/2

Standard Matrix Multiplication

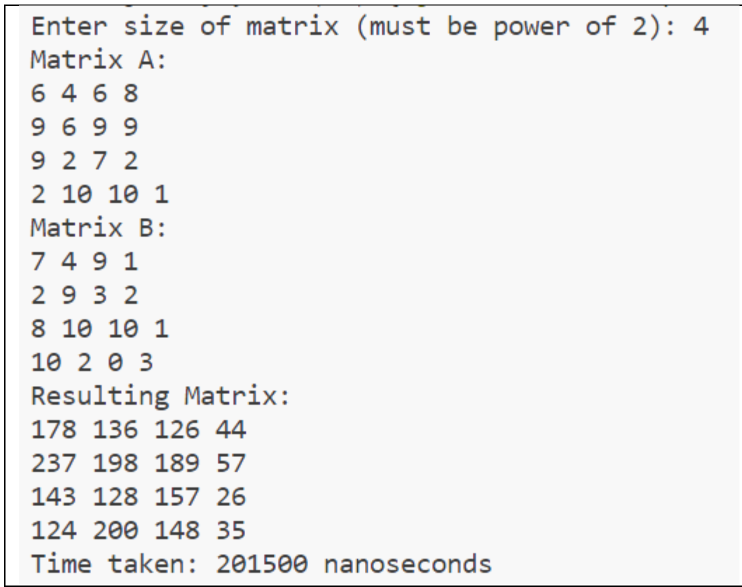
Code: -

```
import java.util.Random;
import java.util.Scanner;
```

```
public class MatrixMultiplication_Recurrence {

    // Generate a random matrix with elements between 0 and 10
    public static int[][] generateRandomMatrix(int n) {
        int[][] matrix = new int[n][n];
        Random rand = new Random();

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                matrix[i][j] = rand.nextInt(11);
            }
        }

        return matrix;
    }

    // Print out a matrix
    public static void printMatrix(int[][] matrix) {
        int n = matrix.length;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println();
        }
    }

    public static int[][] multiply(int[][] a, int[][] b) {
        int n = a.length;
        int[][] c = new int[n][n];

        // Base case
        if (n == 1) {
            c[0][0] = a[0][0] * b[0][0];
        } else {
            // Split matrices into submatrices
            int[][] a11 = new int[n/2][n/2];
            int[][] a12 = new int[n/2][n/2];
            int[][] a21 = new int[n/2][n/2];
            int[][] a22 = new int[n/2][n/2];

            int[][] b11 = new int[n/2][n/2];
            int[][] b12 = new int[n/2][n/2];
            int[][] b21 = new int[n/2][n/2];
            int[][] b22 = new int[n/2][n/2];
            splitMatrix(a, a11, a12, a21, a22);
            splitMatrix(b, b11, b12, b21, b22);

            // Compute the products of the submatrices recursively
            int[][] c11 = add(multiply(a11, b11), multiply(a12, b21));
            int[][] c12 = add(multiply(a11, b12), multiply(a12, b22));
            int[][] c21 = add(multiply(a21, b11), multiply(a22, b21));
            int[][] c22 = add(multiply(a21, b12), multiply(a22, b22));

            // Combine the submatrices into the result matrix
            c = new int[n][n];
            joinMatrices(c, c11, c12, c21, c22);

        }
        return c;
    }

    public static void splitMatrix(int[][] P, int[][] P11, int[][] P12,
int[][] P21, int[][] P22) {
        int n = P.length;
        for (int i = 0; i < n/2; i++) {
            for (int j = 0; j < n/2; j++) {
                P11[i][j] = P[i][j];
                P12[i][j] = P[i][j + n/2];
                P21[i][j] = P[i + n/2][j];
                P22[i][j] = P[i + n/2][j + n/2];
            }
        }
    }

    public static void joinMatrices(int[][] P, int[][] P11, int[][] P12,
int[][] P21, int[][] P22) {
        int n = P.length;
        for (int i = 0; i < n/2; i++) {
            for (int j = 0; j < n/2; j++) {
                P[i][j] = P11[i][j];
                P[i][j + n/2] = P12[i][j];
                P[i + n/2][j] = P21[i][j];
                P[i + n/2][j + n/2] = P22[i][j];
            }
        }
    }

    // Add two matrices
```

```java
    public static int[][] add(int[][] a, int[][] b) {
        int n = a.length;
        int[][] c = new int[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                c[i][j] = a[i][j] + b[i][j];
            }
        }
        return c;
    }

    // Subtract two matrices
    public static int[][] subtract(int[][] a, int[][] b) {
        int n = a.length;
        int[][] c = new int[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                c[i][j] = a[i][j] - b[i][j];
            }
        }
        return c;
    }


    public static void main(String[] args) {
        // User sc for the size of the matrix
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter size of matrix (must be power of 2): ");
        int n = sc.nextInt();

        // Randomly generate matrices with elements between 0 and 10
        int[][] a = generateRandomMatrix(n);
        int[][] b = generateRandomMatrix(n);

        // Print out the matrices before multiplication
        System.out.println("Matrix A:");
        printMatrix(a);
        System.out.println("Matrix B:");
        printMatrix(b);

        // Multiply matrices using Strassen's algorithm and time it
        long startTime = System.nanoTime();
        int[][] c = multiply(a, b);
        long endTime = System.nanoTime();
        // Print out the resulting matrix and the time taken
        System.out.println("Resulting Matrix:");
        printMatrix(c);
        System.out.println("Time taken: " + (endTime - startTime) + "
nanoseconds");
    }

}
```

Output: -



```
Enter size of matrix (must be power of 2): 4
Matrix A:
6 4 6 8
9 6 9 9
9 2 7 2
2 10 10 1
Matrix B:
7 4 9 1
2 9 3 2
8 10 10 1
10 2 0 3
Resulting Matrix:
178 136 126 44
237 198 189 57
143 128 157 26
124 200 148 35
Time taken: 201500 nanoseconds
```

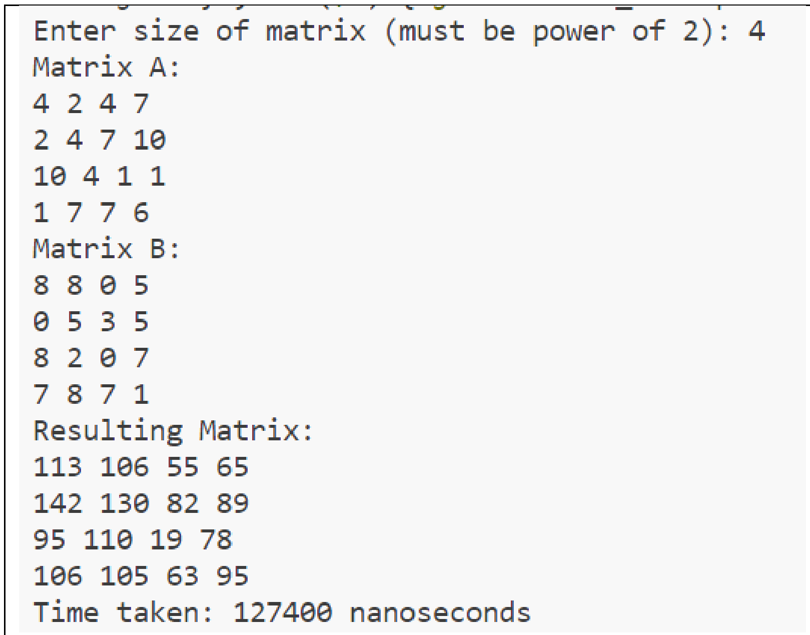
Strassen Matrix Multiplication

Code: -

```
import java.util.Random;
import java.util.Scanner;

public class Matrix_Multiplication_Strassen {

    // Generate a random matrix with elements between 0 and 10
    public static int[][] generateRandomMatrix(int n) {
        int[][] matrix = new int[n][n];
        Random rand = new Random();

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                matrix[i][j] = rand.nextInt(11);
            }
```

```java
        }

        return matrix;
    }

    // Print out a matrix
    public static void printMatrix(int[][] matrix) {
        int n = matrix.length;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println();
        }
    }

    // Multiply two matrices using Strassen's algorithm
    public static int[][] strassenMultiply(int[][] a, int[][] b) {
        int n = a.length;
        int[][] c = new int[n][n];

        // Base case
        if (n == 1) {
            c[0][0] = a[0][0] * b[0][0];
        } else {
            // Split matrices into submatrices
            int[][] a11 = new int[n/2][n/2];
            int[][] a12 = new int[n/2][n/2];
            int[][] a21 = new int[n/2][n/2];
            int[][] a22 = new int[n/2][n/2];

            int[][] b11 = new int[n/2][n/2];
            int[][] b12 = new int[n/2][n/2];
            int[][] b21 = new int[n/2][n/2];
            int[][] b22 = new int[n/2][n/2];

            splitMatrix(a, a11, a12, a21, a22);
            splitMatrix(b, b11, b12, b21, b22);

            // Recursive step
            int[][] p1 = strassenMultiply(add(a11, a22), add(b11, b22));
            int[][] p2 = strassenMultiply(add(a21, a22), b11);
            int[][] p3 = strassenMultiply(a11, subtract(b12, b22));
            int[][] p4 = strassenMultiply(a22, subtract(b21, b11));
            int[][] p5 = strassenMultiply(add(a11, a12), b22);
            int[][] p6 = strassenMultiply(subtract(a21, a11), add(b11,
b12));
            int[][] p7 = strassenMultiply(subtract(a12, a22), add(b21,
b22));

            // Compute submatrices of the resulting matrix
            int[][] c11 = subtract(add(add(p1, p4), p7), p5);
            int[][] c12 = add(p3, p5);
            int[][] c21 = add(p2, p4);
            int[][] c22 = subtract(add(add(p1, p3), p6), p2);

            // Combine the submatrices into the result matrix
            c = new int[n][n];
            joinMatrices(c, c11, c12, c21, c22);
        }

        return c;
    }

    public static void splitMatrix(int[][] P, int[][] P11, int[][] P12,
int[][] P21, int[][] P22) {
        int n = P.length;
        for (int i = 0; i < n/2; i++) {
            for (int j = 0; j < n/2; j++) {
                P11[i][j] = P[i][j];
                P12[i][j] = P[i][j + n/2];
                P21[i][j] = P[i + n/2][j];
                P22[i][j] = P[i + n/2][j + n/2];
            }
        }
    }

    public static void joinMatrices(int[][] P, int[][] P11, int[][] P12,
int[][] P21, int[][] P22) {
        int n = P.length;
        for (int i = 0; i < n/2; i++) {
            for (int j = 0; j < n/2; j++) {
                P[i][j] = P11[i][j];
                P[i][j + n/2] = P12[i][j];
                P[i + n/2][j] = P21[i][j];
                P[i + n/2][j + n/2] = P22[i][j];
            }
        }
    }
    // Add two matrices
    public static int[][] add(int[][] a, int[][] b) {
        int n = a.length;
        int[][] c = new int[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                c[i][j] = a[i][j] + b[i][j];
            }
        }
        return c;
    }

    // Subtract two matrices
    public static int[][] subtract(int[][] a, int[][] b) {
        int n = a.length;
        int[][] c = new int[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                c[i][j] = a[i][j] - b[i][j];
            }
        }
        return c;
    }


    public static void main(String[] args) {
        // User input for the size of the matrix
        Scanner input = new Scanner(System.in);
        System.out.print("Enter size of matrix (must be power of 2): ");
        int n = input.nextInt();

        // Check if matrix size is a power of 2
        if ((n & (n - 1)) != 0) {
            System.out.println("Error: matrix size must be a power of 2.");
            return;
        }

        // Randomly generate matrices with elements between 0 and 10
        int[][] a = generateRandomMatrix(n);
        int[][] b = generateRandomMatrix(n);

        // Print out the matrices before multiplication
        System.out.println("Matrix A:");
        printMatrix(a);
        System.out.println("Matrix B:");
        printMatrix(b);
        // Multiply matrices using Strassen's algorithm and time it
        long startTime = System.nanoTime();
        int[][] c = strassenMultiply(a, b);
        long endTime = System.nanoTime();

        // Print out the resulting matrix and the time taken
        System.out.println("Resulting Matrix:");
        printMatrix(c);
        System.out.println("Time taken: " + (endTime - startTime) + "
nanoseconds");
    }
}
```

Output: -



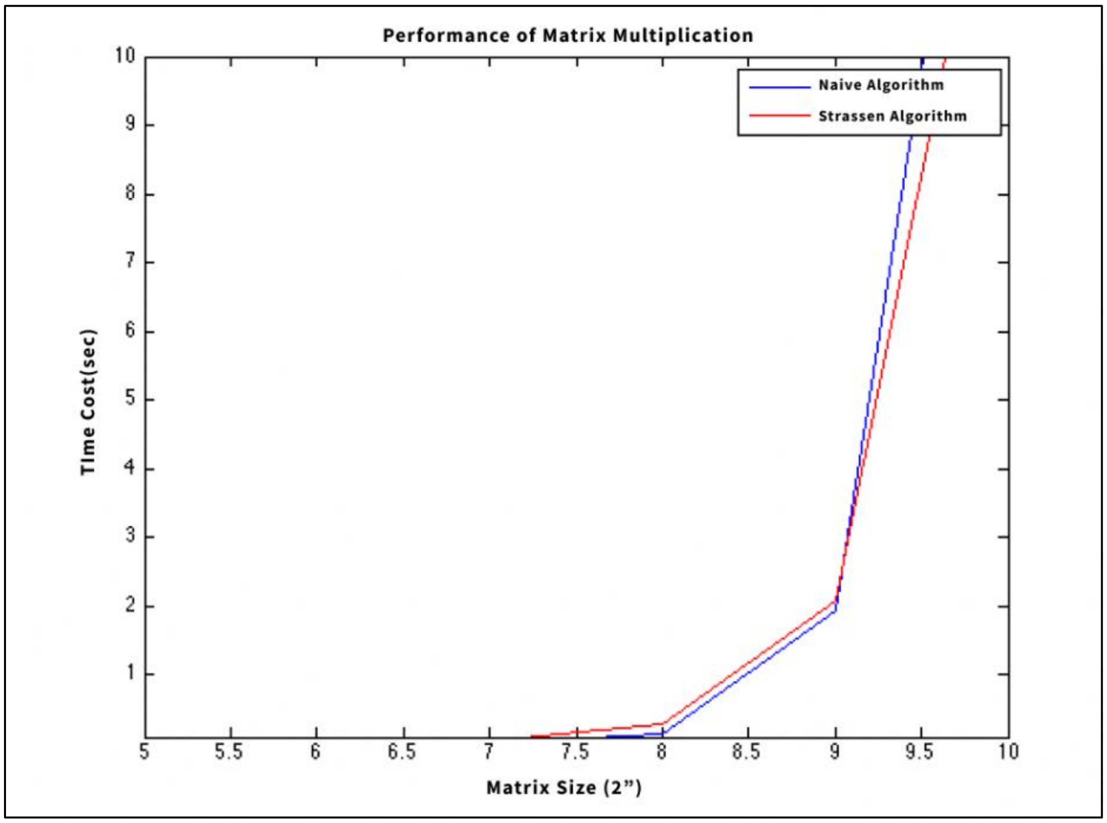
```
Enter size of matrix (must be power of 2): 4
Matrix A:
4 2 4 7
2 4 7 10
10 4 1 1
1 7 7 6
Matrix B:
8 8 0 5
0 5 3 5
8 2 0 7
7 8 7 1
Resulting Matrix:
113 106 55 65
142 130 82 89
95 110 19 78
106 105 63 95
Time taken: 127400 nanoseconds
```

Comparison:

Strassen's matrix multiplication algorithm is a divide-and-conquer approach that can reduce the number of multiplications required to calculate the product of two matrices. The algorithm divides the matrices recursively into smaller submatrices and then calculates the product using a set of mathematical equations. By doing so, the time complexity of matrix multiplication can be reduced from O(n^3) to O(n^log2(7)) ≈ O(n^2.81), which is particularly useful for large matrices where the standard algorithm can be prohibitively expensive. However, it is important to note that Strassen's algorithm may produce inaccuracies due to floating-point rounding errors, making it unstable. In conclusion, Strassen's matrix multiplication algorithm can be an efficient alternative to the standard algorithm for large matrices, with a faster time complexity of O(n^log2(7)) ≈ O(n^2.81).

Generally, Strassen's Method is not preferred for practical applications for following reasons.

1. The constants used in Strassen's method are high and for a typical application Naive method works better.
2. For Sparse matrices, there are better methods especially designed for them.
3. The submatrices in recursion take extra space.
4. Because of the limited precision of computer arithmetic on non-integer values, larger errors accumulate in Strassen's algorithm than in Naive Method.



References: -

https://www.geeksforgeeks.org/strassens-matrix-multiplication/

https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_strassens_matrix_multiplication.htm

https://www.interviewbit.com/blog/strassens-matrix-multiplication/

Experiment No. 7 [Dynamic Programming]

Aim: -

Implement the Floyd Warshall Algorithm for All Pair Shortest Path Problem.You are given a weighted diagraph $G = (V, E)$, with arbitrary edge weights or costs $\boldsymbol{c_{vw}}$ between any node $\boldsymbol{v}$ and node $\boldsymbol{w}$.Find the cheapest path from every node to every other node. Edges may have negative weights.Consider the following test case to check your algorithm:

$\boldsymbol{v}$	$\boldsymbol{w}$	$\boldsymbol{c_{vw}}$
0	1	-1
0	2	4
1	2	3
1	3	2
1	4	2
3	2	5
3	1	1
4	3	-3

The Floyd Warshall Algorithm is for solving all pairs of shortest-path problems. The problem is to find the shortest distances between every pair of vertices in a given edge-weighted directed Graph.

It is an algorithm for finding the shortest path between all the pairs of vertices in a weighted graph. This algorithm follows the dynamic programming approach to find the shortest path.

A C-function for a N x N graph is given below. The function stores the all-pair shortest path in the matrix cost [N][N]. The cost matrix of the given graph is available in cost Mat [N][N].

Floyd-Warshall Algorithm

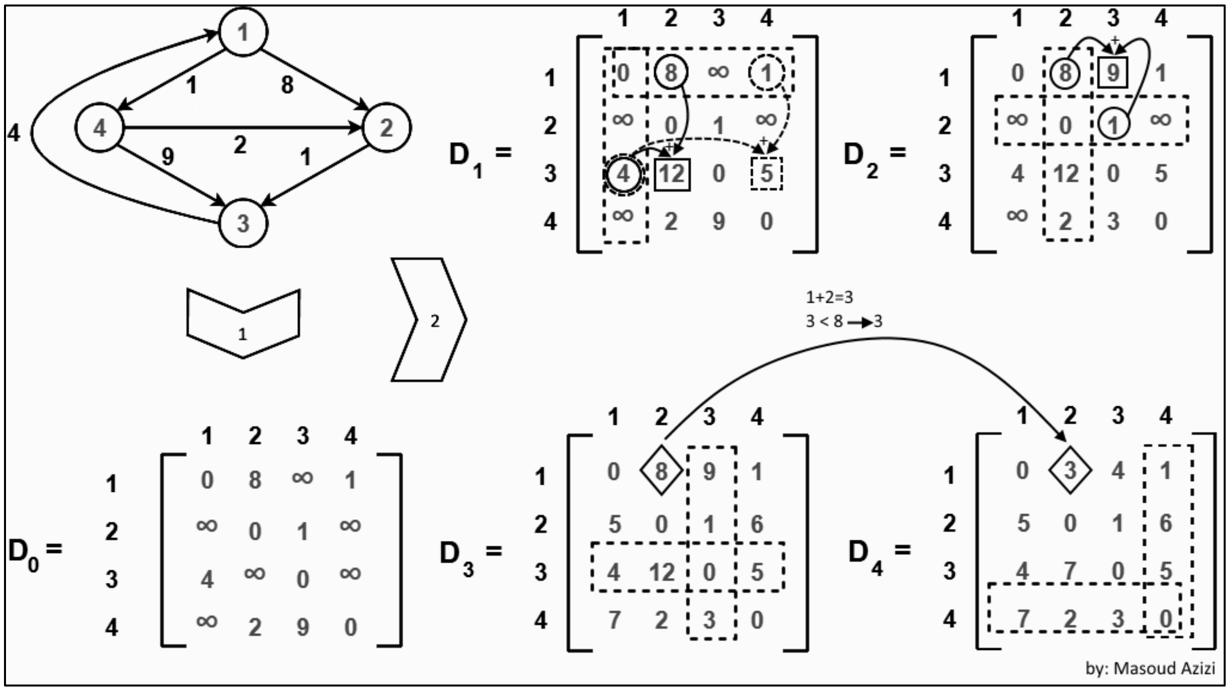
```
n = no of vertices
A = matrix of dimension n*n
for k = 1 to n
    for i = 1 to n
        for j = 1 to n
            $A^k[i, j] = \min(A^{k-1}[i, j], A^{k-1}[i, k] + A^{k-1}[k, j])$
return A
```

Time Complexity

There are three loops. Each loop has constant complexities. So, the time complexity of the Floyd-Warshall algorithm is O(n^3).

Space Complexity

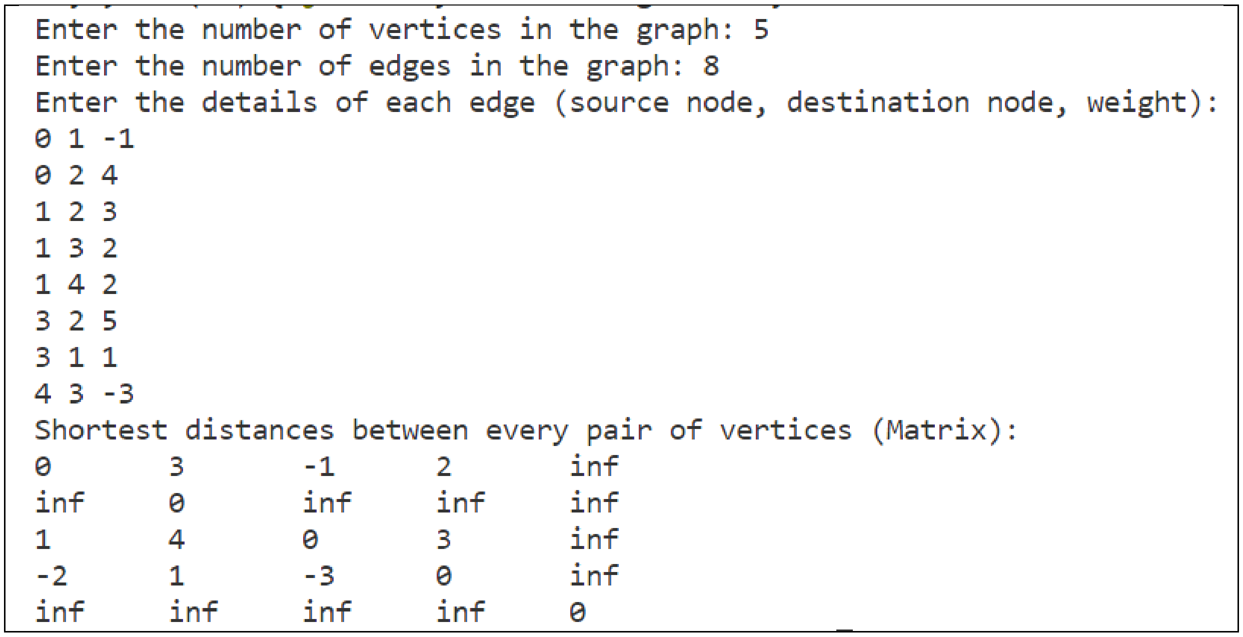
The space complexity of the Floyd-Warshall algorithm is O(n^2).



Code: -

```
import java.util.*;

public class FloydWarshallAlgorithm {
    private int dist[][];
    private int vertex;
    private int inf = Integer.MAX_VALUE;

    public FloydWarshallAlgorithm(int v) {
        vertex = v;
        dist = new int[vertex+1][vertex+1];

        for(int i = 1; i <= vertex; i++) {
            for(int j = 1; j <= vertex; j++) {
                if(i == j) {
                    dist[i][j] = 0;
                } else {
                    dist[i][j] = inf;
                }
            }
        }
    }




    public void addEdge(int u, int v, int w) {
        dist[u][v] = w;
    }

    public void floydWarshall() {
        for(int k = 1; k <= vertex; k++) {
            for(int i = 1; i <= vertex; i++) {
                for(int j = 1; j <= vertex; j++) {
                    if(dist[i][k] != inf && dist[k][j] != inf && dist[i][k]
+ dist[k][j] < dist[i][j]) {
                        dist[i][j] = dist[i][k] + dist[k][j];
                    }
                }
            }
        }
    }

    public void printDistances() {
        System.out.println("Shortest distances between every pair of
vertices (Matrix):");
        for(int i = 1; i <= vertex; i++) {
            for(int j = 1; j <= vertex; j++) {
                if(dist[i][j] == inf) {
                    System.out.print("inf\t");
                } else {
                    System.out.print(dist[i][j] + "\t");
                }
            }
            System.out.println();
        }
    }

    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of vertices in the graph: ");
        int vertices = sc.nextInt();
        FloydWarshallAlgorithm fw = new FloydWarshallAlgorithm(vertices);

        System.out.print("Enter the number of edges in the graph: ");
        int edges = sc.nextInt();

        System.out.println("Enter the details of each edge (source node,
destination node, weight):");
        for(int i = 0; i < edges; i++) {
            int u = sc.nextInt();
            int v = sc.nextInt();
            int w = sc.nextInt();
            fw.addEdge(u, v, w);
        }

        fw.floydWarshall();
        fw.printDistances();
    }
}
```

Output: -



```
Enter the number of vertices in the graph: 5
Enter the number of edges in the graph: 8
Enter the details of each edge (source node, destination node, weight):
0 1 -1
0 2 4
1 2 3
1 3 2
1 4 2
3 2 5
3 1 1
4 3 -3
Shortest distances between every pair of vertices (Matrix):
0       3       -1      2       inf
inf     0       inf     inf     inf
1       4       0       3       inf
-2      1       -3      0       inf
inf     inf     inf     inf     0
```

Applications: -

To find the shortest path is a directed graph.

To find the transitive closure of directed graphs.

To find the Inversion of real matrices.

For testing whether an undirected graph is bipartite.

References: -

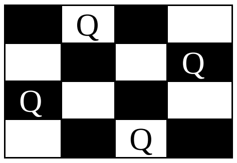
https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/

https://www.programiz.com/dsa/floyd-warshall-algorithm

https://www.javatpoint.com/floyd-warshall-algorithm

Practical No. 8 [Backtracking]

Solve the $\boldsymbol{n}$queens' problem using backtracking. Here, the task is to place $\boldsymbol{n}$ chess queens on an nxn board so that no two queens attack each other. For example, following is a solution for the 4 Queen' problem.



Introduction: -

Backtracking is an algorithmic technique that considers searching in every possible combination for solving a computational problem.
It is known for solving problems recursively one step at a time and removing those solutions that that do not satisfy the problem constraints at any point of time.
It is a **refined brute force** approach that tries out all the possible solutions and chooses the best possible ones out of them.
The backtracking approach is generally used in the cases where there are possibilities of multiple solutions.
There are three types of problems in backtracking –

1. Decision Problem – In this, we search for a feasible solution.
2. Optimization Problem – In this, we search for the best solution.
3. Enumeration Problem – In this, we find all feasible solutions.

Backtracking algorithms are classified into two types:

1) Algorithm for recursive backtracking
2) Non-recursive backtracking algorithm

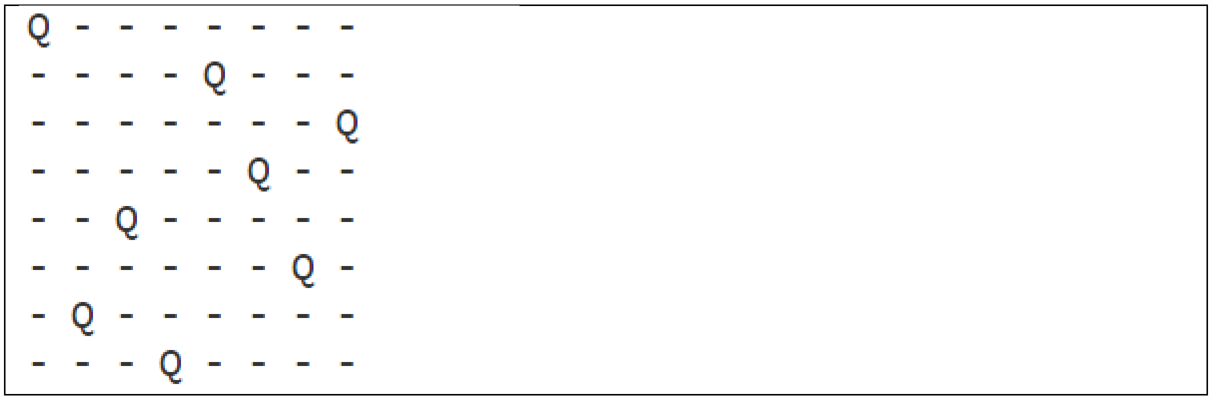
The key to a successful backtracking algorithm is to find a good representation of the problem that allows for efficient testing of partial solutions and pruning of search paths that are unlikely to lead to a solution. Additionally, heuristics such as ordering the variables or values to be tried can greatly speed up the algorithm.

Code: -

```java
public class NQueens {
    private int N;
    private int[] board;

    public NQueens(int N) {
        this.N = N;
        this.board = new int[N];
    }
    public void solve() {
        if (placeQueens(0)) {
            printQueens();
        } else {
            System.out.println("No solution found.");
        }
    }

    private boolean placeQueens(int row) {
        if (row == N) {
            return true;
        }

        for (int i = 0; i < N; i++) {
            board[row] = i;

            if (isValid(row)) {
                if (placeQueens(row + 1)) {
                    return true;
                }
            }
        }

        return false;
    }

    private boolean isValid(int row) {
        for (int i = 0; i < row; i++) {
            if (board[row] == board[i] || Math.abs(board[row] - board[i]) ==
row - i) {
                return false;
            }
        }

        return true;
    }

    private void printQueens() {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                if (board[i] == j) {
                    System.out.print("Q ");
                } else {
                    System.out.print("- ");
```

```java
                }
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        NQueens nQueens = new NQueens(8);
        nQueens.solve();
    }
}
```

Output: -



```
Q - - - - - - -
- - - - Q - - -
- - - - - - - Q
- - - - - Q - -
- - Q - - - - -
- - - - - - Q -
- Q - - - - - -
- - - Q - - - -
```

Applications: -

N-queen problem

Sum of subset problem

Graph coloring

Hamiliton cycle

Maze solving problem.

The Knight’s tour problem.

References: -

https://www.interviewbit.com/courses/programming/backtracking/

https://www.geeksforgeeks.org/introduction-to-backtracking-data-structure-and-algorithm-tutorials/

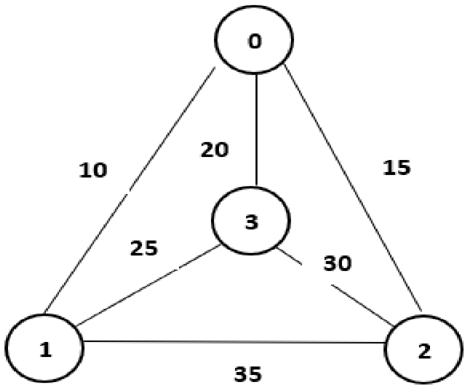
Practical No. 9 [Branch and Bound]

Aim: -

Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible tour that visits every city exactly once and returns to the starting point.

Solve this problem using branch and bound technique.

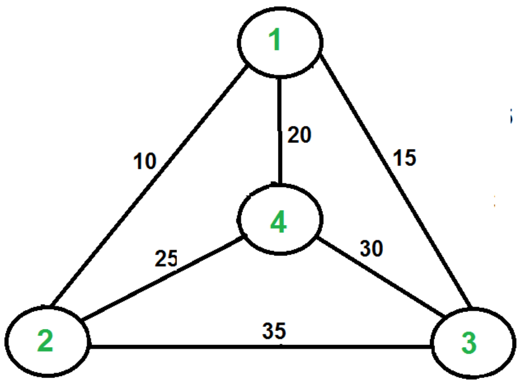
For example, consider the following graph:



A Travelling Salesman Problem (TSP) tour in the graph is $0-1-3-2-0$. The cost of the tour is $10 + 25 + 30 + 15 = 80$.

Introduction: -

Given a set of cities and the distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point. Note the difference between Hamiltonian Cycle and TSP. The Hamiltonian cycle problem is to find if there exists a tour that visits every city exactly once. Here we know that Hamiltonian Tour exists (because the graph is complete) and in fact, many such tours exist, the problem is to find a minimum weight Hamiltonian Cycle.



For example, consider the graph shown in the figure on the right side. A TSP tour in the graph is 1-2-4-3-1. The cost of the tour is 10+25+30+15 which is 80. The problem is a famous NP-hard problem. There is no polynomial-time know solution for this problem. The following are different solutions for the traveling salesman problem.

Algorithm: -

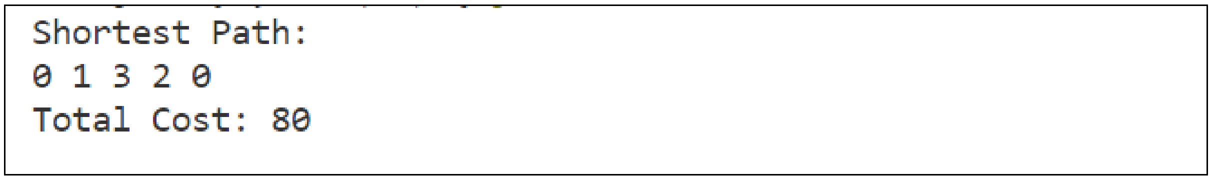
- Travelling salesman problem takes a graph G {V, E} as an input and declare another graph as the output (say G') which will record the path the salesman is going to take from one node to another.
- The algorithm begins by sorting all the edges in the input graph G from the least distance to the largest distance.
- The first edge selected is the edge with least distance, and one of the two vertices (say A and B) being the origin node (say A).
- Then among the adjacent edges of the node other than the origin node (B), find the least cost edge and add it onto the output graph.
- Continue the process with further nodes making sure there are no cycles in the output graph and the path reaches back to the origin node A.
- However, if the origin is mentioned in the given problem, then the solution must always start from that node only. Let us look at some example problems to understand this better.

Code: -

```
public class TSP {
    private int[][] graph;
    private int numCities;
    private int[] path;
    private boolean[] visited;
    private int[] bestPath;
    private int minCost;
    public TSP(int[][] graph) {
        this.graph = graph;
        this.numCities = graph.length;
        this.path = new int[numCities + 1];
        this.visited = new boolean[numCities];
        this.bestPath = new int[numCities + 1];
        this.minCost = Integer.MAX_VALUE;
    }
    public void solve() {
        path[0] = 0;
        visited[0] = true;
        branchAndBound(0, 0, 1);
        printBestPath();
    }
    private void branchAndBound(int currentNode, int currentCost, int level)
{
        if (level == numCities) {
            int cost = currentCost + graph[currentNode][0];
            if (cost < minCost) {
                System.arraycopy(path, 0, bestPath, 0, numCities + 1);
                minCost = cost;
            }
            return;
        }

        for (int i = 1; i < numCities; i++) {
            if (!visited[i]) {
                path[level] = i;
                visited[i] = true;

                int lowerBound = getLowerBound(currentNode, level,
currentCost);
                if (lowerBound < minCost) {
                    branchAndBound(i, currentCost + graph[currentNode][i],
level + 1);
                }
                visited[i] = false;
                path[level] = -1;
            }
        }
    }
    private int getLowerBound(int currentNode, int level, int currentCost) {
        int lb = 0;
        for (int i = 0; i < numCities; i++) {
            if (!visited[i]) {
                int minEdge = Integer.MAX_VALUE;
                for (int j = 0; j < numCities; j++) {
                    if (i != j && !visited[j]) {
                        minEdge = Math.min(minEdge, graph[i][j]);
                    }
                }
                lb += minEdge;
            }
        }
        return currentCost + lb;
    }
    private void printBestPath() {
        System.out.println("Shortest Path: ");
        for (int i = 0; i <= numCities; i++) {
```

```java
            System.out.print(bestPath[i] + " ");
        }
        System.out.println();
        System.out.println("Total Cost: " + minCost);
    }
    public static void main(String[] args) {
        int[][] graph = {
            {0, 10, 15, 20},
            {10, 0, 35, 25},
            {15, 35, 0, 30},
            {20, 25, 30, 0}
        };

        TSP tsp = new TSP(graph);
        tsp.solve();
    }
}
```

Output: -



```
Shortest Path:
0 1 3 2 0
Total Cost: 80
```

Applications: -

The TSP has several applications even in its purest formulation, such as planning, logistics, and the manufacture of microchips. Slightly modified, it appears as a sub-problem in many areas, such as DNA sequencing.

References: -

[DAA | Travelling Salesman Problem]

[Travelling salesman problem - Wikipedia]

Practical No. 10

To design and solve given problems using different algorithmic approaches and analyze their complexity.

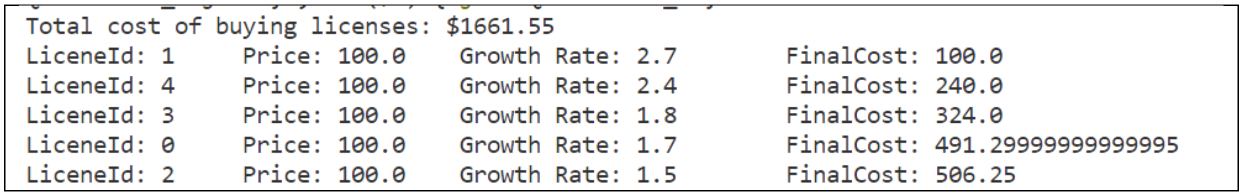
1. Your friends are starting a security company that needs to obtain licenses for n different pieces of cryptographic software. Due to regulations, they can onlyobtain these licenses at the rate of at most one per month.Each license is currently selling for a price of $100. However, they areall becoming more expensive according to exponential growth curves: inparticular, the cost of license j increases by a factor of $rj > 1$each month, whererj is a given parameter. This means that if license j is purchased t months fromnow, it will cost100 $r\ t\ j$. We will assume that all the price growth rates aredistinct; that is, $ri \neq rj$ for licenses $i \neq j$ (even though they start at the sameprice of $100).
   The question is: Given that the company can only buy at most one licensea month, in which order should it buy the licenses so that the total amount ofmoney it spends is as small as possible?
   Give an algorithm that takes the n rates of price growth $r_1, r_2, \ldots, r_n$, andcomputes an order in which to buy the licenses so that the total amount ofmoney spent is minimized. The running time of your algorithm should bepolynomial in n.

Code: -

```
import java.util.*;

class License implements Comparable<License>{
    int index;
    double price;
    double growthRate;
    double finalCost;

    License(int index, double price, double growthRate) {
        this.index = index;
        this.price = price;
        this.growthRate = growthRate;
    }
    @Override
    public int compareTo(License next) {
        return Double.compare(next.growthRate, this.growthRate);
    }

    @Override
    public String toString() {
        return "LiceneId: "+index+"\tPrice: "+price+"\tGrowth Rate: 
"+growthRate+"\tFinalCost: "+finalCost;
    }
```

```
}
public class Question10_1 {
    public static List<License> findBuyingList(double[] growthRates) {
        int n = growthRates.length;
        List<License> licenses = new ArrayList<>();
        List<License> boughtLicenses = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            licenses.add(new License(i, 100, growthRates[i]));
        }

        // Sort licenses in descending order based on growth rate
        Collections.sort(licenses); //nlog(n)

        double totalCost = 0.0;
        double previousTotalCost = 0.0;
        double currentMonth = 0; //month counter
        for (License license : licenses) {
            // Calculate the price of the license at the current month
            //double priceAtCurrentMonth =
license.price*license.growthRate*currentMonth;
            double priceAtCurrentMonth =
license.price*Math.pow(license.growthRate,currentMonth);

            // Update
            previousTotalCost = totalCost;
            totalCost += priceAtCurrentMonth;
            license.finalCost = totalCost - previousTotalCost;
            boughtLicenses.add(license);

            // Increment current month
            currentMonth += 1.0;
        }

        return boughtLicenses;
    }
    public static void main(String[] args) {
        // Example usage
        double[] growthRates = {1.7, 2.7, 1.5, 1.8, 2.4}; // Growth rates
for each license
        double totalCost = 0;
        List<License> buyingList = findBuyingList(growthRates);
        for(License l : buyingList) {
            totalCost += l.finalCost;
        }
        System.out.printf("Total cost of buying licenses: $%.2f",totalCost);
        System.out.println();
        for(License l: buyingList) {
            System.out.println(l);
        }
    }
}
```

Output: -



```
Total cost of buying licenses: $1661.55
LiceneId: 1     Price: 100.0    Growth Rate: 2.7        FinalCost: 100.0
LiceneId: 4     Price: 100.0    Growth Rate: 2.4        FinalCost: 240.0
LiceneId: 3     Price: 100.0    Growth Rate: 1.8        FinalCost: 324.0
LiceneId: 0     Price: 100.0    Growth Rate: 1.7        FinalCost: 491.29999999999995
LiceneId: 2     Price: 100.0    Growth Rate: 1.5        FinalCost: 506.25
```

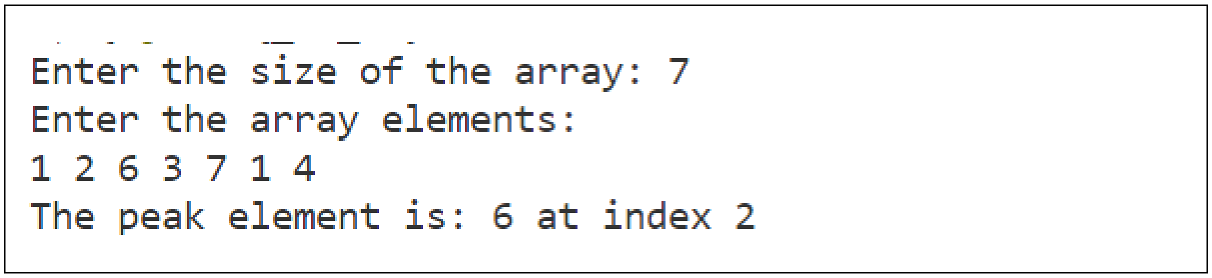
2. Suppose you are given an array A with n entries, with each entry holding adistinct number. You are told that the sequence of values $A[1], A[2], \dots, A[n]$is unimodal.That is, for some index p between 1 and n, the values in the array entriesincrease up to position p in A and then decrease the remainder of the wayuntil position n. (So if you were to draw a plot with the array position j on the$x$-axis and the value of the entry $A[j]$ on the y-axis, the plotted points wouldrise until x-value p, where they'd achieve their maximum value, and then fall fromthere on). You'd like to find the "peak entry" p without having to read the entirearray - in fact, by reading as few entries of A as possible. Show how to findthe entry p by reading at most $O(log\ n)$ entries of A.

Code: -

```
import java.util.*;

public class Q_10_2 {

    // binary search O(nlogn)
    public static int findPeak(int[] A, int n) {
        int l = 0, r = n - 1;
        while (l <= r) {
            int m = (l + r) / 2;
            if ((m == 0 || A[m] > A[m-1]) && (m == n-1 || A[m] > A[m+1])) {
                return m; // peak found
            } else if (m > 0 && A[m] < A[m-1]) {
                r = m - 1; // search left half
            } else {
                l = m + 1; // search right half
            }
        }
        return -1; // peak not found
```

```java
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int n = sc.nextInt();
        int[] A = new int[n];
        System.out.println("Enter the array elements:");
        for (int i = 0; i < n; i++) {
            A[i] = sc.nextInt();
        }
        int peakIndex = findPeak(A, n);
        System.out.println("The peak element is: " + A[peakIndex] + " at index "
+ peakIndex);
    }
}
```

Output: -



```
Enter the size of the array: 7
Enter the array elements:
1 2 6 3 7 1 4
The peak element is: 6 at index 2
```