Week 06

DSA Practice sheet

Name – Suraj Vishwakarma

Course – MCA 2nd year(2025-2026)

Q-ID – 24510031

(1) Write a program to perform Linear Search on an array.

```
public class LinearSearch {
   public static int linearSearch(int[] arr, int key) {
      for (int i = 0; i < arr.length; i++) {
         if (arr[i] == key)
            return i;
      }
      return -1;
   }

   public static void main(String[] args) {
      int[] arr = {10, 20, 30, 40, 50};
      int key = 30;
      int result = linearSearch(arr, key);
      if (result != -1)
         System.out.println("Element found at index: " + result);
      else
         System.out.println("Element not found");   }
}
```

2 Write a program to perform Binary Search on a sorted array.

```
public class BinarySearch {
    public static int binarySearch(int[] arr, int key) {
        int low = 0, high = arr.length - 1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (arr[mid] == key) return mid;
            else if (arr[mid] < key) low = mid + 1;
            else high = mid - 1;
        }
        return -1;
    }

    public static void main(String[] args) {
        int[] arr = {10, 20, 30, 40, 50};
        int key = 40;
        int result = binarySearch(arr, key);
        if (result != -1)
            System.out.println("Element found at index: " + result);
        else
            System.out.println("Element not found");
    }
}
```

3 Modify Linear Search to return index of all occurrences of a key.

```
import java.util.*;


public class LinearSearchAll {
   public static List<Integer> linearSearchAll(int[] arr, int key) {
      List<Integer> indices = new ArrayList<>();
      for (int i = 0; i < arr.length; i++) {
         if (arr[i] == key)
            indices.add(i);
      }
      return indices;
   }


   public static void main(String[] args) {
      int[] arr = {10, 20, 30, 20, 40, 20};
      int key = 20;
      List<Integer> result = linearSearchAll(arr, key);
      if (!result.isEmpty())
         System.out.println("Element found at indices: " + result);
      else
         System.out.println("Element not found");
   }
}
```

(4) Write a program to find the number of comparisons made in Linear Search.

```
public class LinearSearchComparisons {
   public static int linearSearch(int[] arr, int key) {
      int comparisons = 0;
```

```
        for (int value : arr) {
            comparisons++;
            if (value == key) {
                System.out.println("Comparisons made: " + comparisons);
                return comparisons;
            }
        }
        System.out.println("Comparisons made: " + comparisons);
        return comparisons;
    }

    public static void main(String[] args) {
        int[] arr = {10, 20, 30, 40, 50};
        int key = 40;
        linearSearch(arr, key);
    }
}
```

(5) Write a program to find the number of comparisons made in Binary Search.

```
public class BinarySearchComparisons {
    public static int binarySearch(int[] arr, int key) {
        int comparisons = 0;
        int low = 0, high = arr.length - 1;
        while (low <= high) {
            comparisons++;
            int mid = (low + high) / 2;
            if (arr[mid] == key) {
```

```
            System.out.println("Comparisons made: " + comparisons);
            return comparisons;
        } else if (arr[mid] < key) low = mid + 1;
        else high = mid - 1;
    }
    System.out.println("Comparisons made: " + comparisons);
    return comparisons;
  }

  public static void main(String[] args) {
    int[] arr = {10, 20, 30, 40, 50};
    int key = 30;
    binarySearch(arr, key);
  }
}
```

6. Write a program to sort an array using Bubble Sort.

```
public class BubbleSort {
    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }

    public static void main(String[] args) {
        int[] arr = {64, 25, 12, 22, 11};
        bubbleSort(arr);
        System.out.println("Sorted array: ");
        for (int num : arr) System.out.print(num + " ");
    }
}
```

---

7. Write a program to sort an array using Selection Sort.

```
public class SelectionSort {
    public static void selectionSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            int min = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[min]) min = j;
            }
            int temp = arr[min];
            arr[min] = arr[i];
              arr[i] = temp;
        }
    }

    public static void main(String[] args) {
        int[] arr = {64, 25, 12, 22, 11};
        selectionSort(arr);
        System.out.println("Sorted array: ");
        for (int num : arr) System.out.print(num + " ");
    }
}
```

8 Write a program to sort an array using Insertion Sort.

```
public class InsertionSort {
    public static void insertionSort(int[] arr) {
        for (int i = 1; i < arr.length; i++) {
            int key = arr[i];
            int j = i - 1;
            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j--;
            }
            arr[j + 1] = key;
        }
    }

    public static void main(String[] args) {
        int[] arr = {64, 25, 12, 22, 11};
        insertionSort(arr);
        System.out.println("Sorted array: ");
        for (int num : arr) System.out.print(num + " ");
    }
}
```

---

9 Write a program to sort an array using Merge Sort.

```
public class MergeSort {
    public static void mergeSort(int[] arr, int l, int r) {
        if (l < r) {
            int m = (l + r) / 2;
            mergeSort(arr, l, m);
            mergeSort(arr, m + 1, r);
            merge(arr, l, m, r);
        }
    }

    public static void merge(int[] arr, int l, int m, int r) {
```

```
        int n1 = m - l + 1, n2 = r - m;
        int[] L = new int[n1];
        int[] R = new int[n2];
        for (int i = 0; i < n1; i++) L[i] = arr[l + i];
        for (int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];

        int i = 0, j = 0, k = l;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) arr[k++] = L[i++];
            else arr[k++] = R[j++];
        }
        while (i < n1) arr[k++] = L[i++];
        while (j < n2) arr[k++] = R[j++];
    }

    public static void main(String[] args) {
        int[] arr = {64, 25, 12, 22, 11};
        mergeSort(arr, 0, arr.length - 1);
        System.out.println("Sorted array: ");
        for (int num : arr) System.out.print(num + " ");
    }
}
```

---

10 Write a program to sort an array using Quick Sort.

```
public class QuickSort {
    public static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
    }

    public static int partition(int[] arr, int low, int high) {
        int pivot = arr[high], i = (low - 1);
        for (int j = low; j < high; j++) {
            if (arr[j] <= pivot) {
                i++;
                int temp = arr[i]; arr[i] = arr[j]; arr[j] = temp;
            }
        }
        int temp = arr[i + 1]; arr[i + 1] = arr[high]; arr[high] = temp;
        return i + 1;
    }

    public static void main(String[] args) {
        int[] arr = {64, 25, 12, 22, 11};
        quickSort(arr, 0, arr.length - 1);
        System.out.println("Sorted array: ");
        for (int num : arr) System.out.print(num + " ");
    }
}
```

---

11. Write a program to sort an array using Heap Sort.

```
public class HeapSort {
    public static void heapify(int[] arr, int n, int i) {
        int largest = i, left = 2 * i + 1, right = 2 * i + 2;
```

```
        if (left < n && arr[left] > arr[largest]) largest = left;
        if (right < n && arr[right] > arr[largest]) largest = right;

        if (largest != i) {
            int swap = arr[i]; arr[i] = arr[largest]; arr[largest] = swap;
            heapify(arr, n, largest);
        }
    }

    public static void heapSort(int[] arr) {
        int n = arr.length;
        for (int i = n / 2 - 1; i >= 0; i--) heapify(arr, n, i);
        for (int i = n - 1; i > 0; i--) {
            int temp = arr[0]; arr[0] = arr[i]; arr[i] = temp;
            heapify(arr, i, 0);
        }
    }

    public static void main(String[] args) {
        int[] arr = {64, 25, 12, 22, 11};
        heapSort(arr);
        System.out.println("Sorted array: ");
        for (int num : arr) System.out.print(num + " ");
    }
}
```

---

12. Write a program to search an element after sorting the array using Binary Search.

```
import java.util.Arrays;

public class SearchAfterSort {
    public static int binarySearch(int[] arr, int key) {
        int low = 0, high = arr.length - 1;
        while (low <= high) {
            int mid = (low + high) / 2;
            if (arr[mid] == key) return mid;
            else if (arr[mid] < key) low = mid + 1;
            else high = mid - 1;
        }
        return -1;
    }

    public static void main(String[] args) {
        int[] arr = {64, 25, 12, 22, 11};
        Arrays.sort(arr);
        System.out.println("Sorted array: " + Arrays.toString(arr));
        int key = 22;
        int result = binarySearch(arr, key);
        if (result != -1) System.out.println("Element found at index: " +
result);
        else System.out.println("Element not found");
    }
}
```

---

13. Write a program to compare time complexity of Bubble Sort and Quick Sort (with large input).

```
import java.util.*;

public class CompareSorts {
    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j]; arr[j] = arr[j + 1]; arr[j + 1] =
temp;
                }
            }
        }
    }

    public static void quickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
    }

    public static int partition(int[] arr, int low, int high) {
        int pivot = arr[high], i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j] <= pivot) {
                i++;
                int temp = arr[i]; arr[i] = arr[j]; arr[j] = temp;
            }
        }
        int temp = arr[i + 1]; arr[i + 1] = arr[high]; arr[high] = temp;
        return i + 1;
    }

    public static void main(String[] args) {
        Random rand = new Random();
        int size = 10000;
        int[] arr1 = new int[size];
        int[] arr2 = new int[size];

        for (int i = 0; i < size; i++) {
            arr1[i] = arr2[i] = rand.nextInt(100000);
        }

        long start = System.nanoTime();
        bubbleSort(arr1);
        long end = System.nanoTime();
        System.out.println("Bubble Sort Time: " + (end - start) + " ns");

        start = System.nanoTime();
        quickSort(arr2, 0, arr2.length - 1);
        end = System.nanoTime();
        System.out.println("Quick Sort Time: " + (end - start) + " ns");
    }
}
```

---

14. Write a program to find the kth smallest element using Quick Sort (partition method)

```
public class KthSmallest {
    public static int partition(int[] arr, int low, int high) {
        int pivot = arr[high], i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j] <= pivot) {
                i++;
                int temp = arr[i]; arr[i] = arr[j]; arr[j] = temp;
            }
        }
        int temp = arr[i + 1]; arr[i + 1] = arr[high]; arr[high] = temp;
        return i + 1;
    }

    public static int kthSmallest(int[] arr, int low, int high, int k) {
        if (k > 0 && k <= high - low + 1) {
            int pos = partition(arr, low, high);
            if (pos - low == k - 1) return arr[pos];
            if (pos - low > k - 1) return kthSmallest(arr, low, pos - 1, k);
            return kthSmallest(arr, pos + 1, high, k - pos + low - 1);
        }
        return Integer.MAX_VALUE;
    }

    public static void main(String[] args) {
        int[] arr = {12, 3, 5, 7, 4, 19, 26};
        int k = 3;
        System.out.println(k + "rd smallest element is " + kthSmallest(arr,
0, arr.length - 1, k));
    }
}
```

---

15. Write a program to find the largest and smallest element in an array using a single.

```
public class MinMaxSingleScan {
    public static void main(String[] args) {
        int[] arr = {64, 25, 12, 22, 11};
        int min = arr[0], max = arr[0];
        for (int num : arr) {
            if (num < min) min = num;
            if (num > max) max = num;
        }
        System.out.println("Smallest element: " + min);
        System.out.println("Largest element: " + max);
    }
}
```

SECTION B: Objective Questions (30)

- **Linear Search has time complexity:**

(b) **O(n)**

- **Binary Search works only on:**

(a) **Sorted arrays**

- **Worst-case time complexity of Binary Search:**

(b) **O(log n)**

Which sorting algorithm is stable?
(b) **Merge Sort**

Which sorting algorithm is NOT in-place?

(a) **Merge Sort**

Bubble Sort has best case time complexity:
(a) **O(n)**

Worst case complexity of Selection Sort:
(a) **$O(n^2)$**

In Insertion Sort, elements are inserted into:
(a) **Sorted part of array**

Quick Sort uses which technique?
(a) **Divide and Conquer**

Merge Sort is based on:
(a) **Divide and Conquer**

Heap Sort uses which data structure?
(b) **Heap (complete binary tree)**

Which algorithm performs the least comparisons on average for large data?
(b) **Quick Sort**

Best case time complexity of Binary Search:
(a) **O(1)**

Merge Sort requires extra space of:
(c) **O(n)**

In Quick Sort, worst case occurs when:
(a) **Pivot is always smallest/largest element**

Which sorting algorithm is best for nearly sorted data?
(b) **Insertion Sort**

Which is NOT comparison-based sorting?
(b) **Counting Sort**

Average case time complexity of Quick Sort:
(b) **O(n log n)**

Which algorithm guarantees O(n log n) in all cases?
(d) **Both b & c** (Merge Sort & Heap Sort)

What is time complexity of Heapify operation?
(b) **O(log n)**

Which sorting algorithm is called "exchange sort"?
(a) **Bubble Sort**

🎬 **Worst case complexity of Insertion Sort:**
(b) **$O(n^2)$**

🎬 **Which searching algorithm is faster for large data sets?**
(b) **Binary Search**

🎬 **Stability means:**
(a) **Preserves relative order of equal elements**

🎬 **Which is in-place and stable?**
(a) **Insertion Sort**

🎬 **Time complexity of building a heap is:**
(a) **O(n)**

🎬 **Binary Search on linked list takes:**
(c) **O(n)**

🎬 **In Selection Sort, total swaps in worst case are:**
(b) **O(n)**

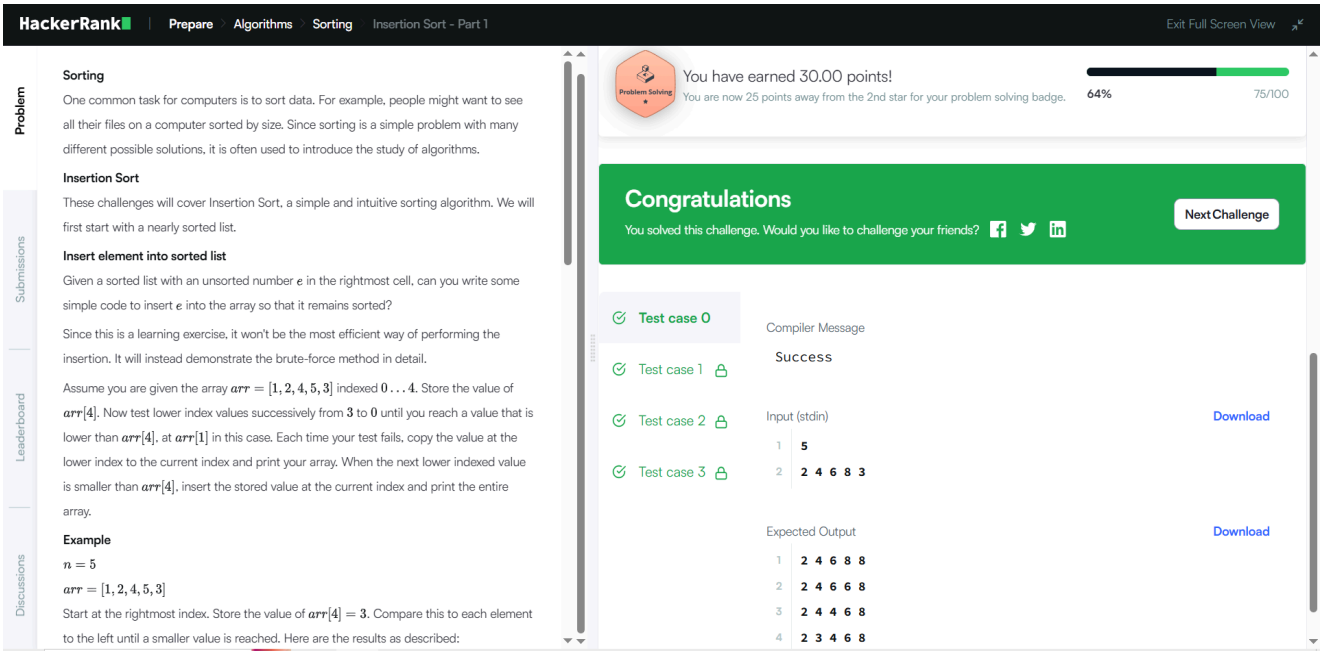
🎬 **Best sorting algorithm when memory is limited?**
(b) **Heap Sort**

🎬 **Which has best average case complexity among these?**
(d) **Quick Sort**

SECTION C:

🎬 Sorting Practice

(basic):https://www.hackerrank.com/challenges/insertionsort1/problem


HackerRank
Prepare
Algorithms
Sorting
Insertion Sort - Part 1
Exit Full Screen View
Problem
Submissions
Leaderboard
Discussions
Sorting
One common task for computers is to sort data. For example, people might want to see all their files on a computer sorted by size. Since sorting is a simple problem with many different possible solutions, it is often used to introduce the study of algorithms.
Insertion Sort
These challenges will cover Insertion Sort, a simple and intuitive sorting algorithm. We will first start with a nearly sorted list.
Insert element into sorted list
Given a sorted list with an unsorted number e in the rightmost cell, can you write some simple code to insert e into the array so that it remains sorted?
Since this is a learning exercise, it won't be the most efficient way of performing the insertion. It will instead demonstrate the brute-force method in detail.
Assume you are given the array arr = [1, 2, 4, 5, 3] indexed 0 . . . 4. Store the value of arr[4]. Now test lower index values successively from 3 to 0 until you reach a value that is lower than arr[4], at arr[1] in this case. Each time your test fails, copy the value at the lower index to the current index and print your array. When the next lower indexed value is smaller than arr[4], insert the stored value at the current index and print the entire array.
Example
n = 5
arr = [1, 2, 4, 5, 3]
Start at the rightmost index. Store the value of arr[4] = 3. Compare this to each element to the left until a smaller value is reached. Here are the results as described:
You have earned 30.00 points!
You are now 25 points away from the 2nd star for your problem solving badge.
64%
75/100
Congratulations
You solved this challenge. Would you like to challenge your friends?
Next Challenge
Test case 0
Test case 1
Test case 2
Test case 3
Compiler Message
Success
Input (stdin)
Download
1 5
2 2 4 6 8 3
Expected Output
Download
1 2 4 6 8 8
2 2 4 6 6 8
3 2 4 4 6 8
4 2 3 4 6 8