Design and Analysis of Algorithms
Assignment #2

Sri Sai Anusha Gandu **16230560**

Insertion Sort

```
package com.java.insertion;

import java.util.Scanner;

public class InsertionSort {

    /* Insertion Sort function */
    public static void sort(int arr[]) {
        int N = arr.length;
        int i, j, temp;
        for (i = 1; i < N; i++) {
            j = i;
            temp = arr[i];
            while (j > 0 && temp < arr[j - 1]) {
                System.out.println("Comparison");
                arr[j] = arr[j - 1];
                j = j - 1;
            }
            arr[j] = temp;
        }
    }

    /* Main method */
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Insertion Sort Test\n");
        int n, i;
        /* Accept number of elements */
        System.out.println("Enter number of integer elements");
        n = scan.nextInt();
        /* Create integer array on n elements */
        int arr[] = new int[n];
        /* Accept elements */
        System.out.println("\nEnter " + n + " integer elements");
        for (i = 0; i < n; i++)
```

```
			arr[i] = scan.nextInt();
		/* Call method sort */
		sort(arr);
		/* Print sorted Array */
		System.out.println("\nElements after sorting ");
		for (i = 0; i < n; i++)
			System.out.print(arr[i] + " ");
		System.out.println();
	}

}
```

Merge Sort

```
package com.java.merge;

import java.util.Scanner;

public class MergeSort {

	public static void sort(int[] a, int low, int high) {
		int N = high - low;
		if (N <= 1)
			return;
		int mid = low + N / 2;
		// recursively sort
		sort(a, low, mid);
		sort(a, mid, high);
		// merge two sorted subarrays
		int[] temp = new int[N];
		int i = low, j = mid;
		for (int k = 0; k < N; k++) {
			if (i == mid) {
				System.out.println("Comparison");
				temp[k] = a[j++];
			} else if (j == high) {
				System.out.println("Comparison");
				temp[k] = a[i++];
			} else if (a[j] < a[i]) {
				System.out.println("Comparison");
				temp[k] = a[j++];
			} else {
```

```
                System.out.println("Comparison");
                temp[k] = a[i++];
            }
        }
        for (int k = 0; k < N; k++)
            a[low + k] = temp[k];
    }

    /* Main method */
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Merge Sort Test\n");
        int n, i;
        /* Accept number of elements */
        System.out.println("Enter number of integer elements");
        n = scan.nextInt();
        /* Create array of n elements */
        int arr[] = new int[n];
        /* Accept elements */
        System.out.println("\nEnter " + n + " integer elements");
        for (i = 0; i < n; i++)
            arr[i] = scan.nextInt();
        /* Call method sort */
        sort(arr, 0, n);
        /* Print sorted Array */
        System.out.println("\nElements after sorting ");
        for (i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

}
```

Heap Sort

```
package com.java.heap;

import java.util.Scanner;

public class HeapSort {

    private static int N;
```

```
/* Sort Function */
public static void sort(int arr[]) {
        heapify(arr);
        for (int i = N; i > 0; i--) {
                swap(arr, 0, i);
                N = N - 1;
                maxheap(arr, 0);
        }
}

/* Function to build a heap */
public static void heapify(int arr[]) {
        N = arr.length - 1;
        for (int i = N / 2; i >= 0; i--)
                maxheap(arr, i);
}

/* Function to swap largest element in heap */
public static void maxheap(int arr[], int i) {
        int left = 2 * i;
        int right = 2 * i + 1;
        int max = i;
        if (left <= N && arr[left] > arr[i]) {
                System.out.println("Comparison");
                max = left;
        }
        if (right <= N && arr[right] > arr[max]) {
                System.out.println("Comparison");
                max = right;
        }

        if (max != i) {
                swap(arr, i, max);
                maxheap(arr, max);
        }
}

/* Function to swap two numbers in an array */
public static void swap(int arr[], int i, int j) {
        System.out.println("Comparison");
        int tmp = arr[i];
        arr[i] = arr[j];
        arr[j] = tmp;
    }

    /* Main method */
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Heap Sort Test\n");
        int n, i;
        /* Accept number of elements */
        System.out.println("Enter number of integer elements");
        n = scan.nextInt();
        /* Make array of n elements */
        int arr[] = new int[n];
        /* Accept elements */
        System.out.println("\nEnter " + n + " integer elements");
        for (i = 0; i < n; i++)
            arr[i] = scan.nextInt();
        /* Call method sort */
        sort(arr);
        /* Print sorted Array */
        System.out.println("\nElements after sorting ");
        for (i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

}
```

Quick Sort

```
package com.java.quick;

import java.util.Scanner;

public class QuickSort {

    /** Quick Sort function **/
    public static void sort(int[] arr) {
        quickSort(arr, 0, arr.length - 1);
    }
```

```
/** Quick sort function **/
public static void quickSort(int arr[], int low, int high) {
        int i = low, j = high;
        int temp;
        int pivot = arr[(low + high) / 2];

        /** partition **/
        while (i <= j) {
                while (arr[i] < pivot) {
                        System.out.println("Comparison");
                        i++;
                }
                while (arr[j] > pivot) {
                        System.out.println("Comparison");
                        j--;
                }
                if (i <= j) {
                        /** swap **/
                        System.out.println("Comparison");
                        temp = arr[i];
                        arr[i] = arr[j];
                        arr[j] = temp;

                        i++;
                        j--;
                }
        }

        /** recursively sort lower half **/
        if (low < j)
                quickSort(arr, low, j);
        /** recursively sort upper half **/
        if (i < high)
                quickSort(arr, i, high);
}

/** Main method **/
public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Quick Sort Test\n");
        int n, i;
        /** Accept number of elements **/
        System.out.println("Enter number of integer elements");
        n = scan.nextInt();
        /** Create array of n elements **/
        int arr[] = new int[n];
        /** Accept elements **/
        System.out.println("\nEnter " + n + " integer elements");
        for (i = 0; i < n; i++)
            arr[i] = scan.nextInt();
        /** Call method sort **/
        sort(arr);
        /** Print sorted Array **/
        System.out.println("\nElements after sorting ");
        for (i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

}
```

Counting Sort

```
package com.java.counting;

import java.util.Scanner;

public class CountingSort {

    private static final int MAX_RANGE = 1000000;

    /** Counting Sort function **/
    public static void sort(int[] arr) {
        int N = arr.length;
        if (N == 0)
            return;
        /** find max and min values **/
        int max = arr[0], min = arr[0];
        for (int i = 1; i < N; i++) {
            if (arr[i] > max) {
                System.out.println("Comparison");
                max = arr[i];
            }
            if (arr[i] < min) {
                System.out.println("Comparison");
                min = arr[i];
            }
        }
        int range = max - min + 1;

        /** check if range is small enough for count array **/
        /**
         * else it might give out of memory exception while allocating memory for
array
         **/
        if (range > MAX_RANGE) {
            System.out.println("\nError : Range too large for sort");
            return;
        }

        int[] count = new int[range];
        /** make count/frequency array for each element **/
        for (int i = 0; i < N; i++) {
            count[arr[i] - min]++;
        }
        /** modify count so that positions in final array is obtained **/
        for (int i = 1; i < range; i++) {
            count[i] += count[i - 1];
        }
        /** modify original array **/
        int j = 0;
        for (int i = 0; i < range; i++)
            while (j < count[i]) {
                System.out.println("Comparison");
                arr[j++] = i + min;
            }
    }

    /** Main method **/
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Counting Sort Test\n");
        int n, i;
        /** Accept number of elements **/
        System.out.println("Enter number of integer elements");
        n = scan.nextInt();
        /** Create integer array on n elements **/
        int arr[] = new int[n];
        /** Accept elements **/
        System.out.println("\nEnter " + n + " integer elements");
        for (i = 0; i < n; i++)
            arr[i] = scan.nextInt();
        /** Call method sort **/
        sort(arr);
        /** Print sorted Array **/
        System.out.println("\nElements after sorting ");
        for (i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

}
```

Radix Sort

```
package com.java.radix;

import java.util.Scanner;

public class RadixSort {

    /** Radix Sort function **/
    public static void sort(int[] a) {
        int i, m = a[0], exp = 1, n = a.length;
        int[] b = new int[1000];
        for (i = 1; i < n; i++)
            if (a[i] > m) {
                System.out.println("Comparison");
                m = a[i];
            }
        while (m / exp > 0) {
            int[] bucket = new int[1000];
            for (i = 0; i < n; i++)
                System.out.println("Comparison");
                bucket[(a[i] / exp) % 1000]++;
            for (i = 1; i < 1000; i++)
```

```
                System.out.println("Comparison");
                bucket[i] += bucket[i - 1];
            for (i = n - 1; i >= 0; i--)
                System.out.println("Comparison");
                b[--bucket[(a[i] / exp) % 1000]] = a[i];
            for (i = 0; i < n; i++)
                System.out.println("Comparison");
                a[i] = b[i];
            exp *= 1000;
        }
    }

    /** Main method **/
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Radix Sort Test\n");
        int n, i;
        /** Accept number of elements **/
        System.out.println("Enter number of integer elements");
        n = scan.nextInt();
        /** Create integer array on n elements **/
        int arr[] = new int[n];
        /** Accept elements **/
        System.out.println("\nEnter " + n + " integer elements");
        for (i = 0; i < n; i++)
            arr[i] = scan.nextInt();
        /** Call method sort **/
        sort(arr);
        /** Print sorted Array **/
        System.out.println("\nElements after sorting ");
        for (i = 0; i < n; i++)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

}
```

The tabulated form of the number of comparisons is given below.

Algorithm	Random Array	Sorted Array	Reverse Array
Insertion Sort	250743	999	500499
Merge Sort	9966	9966	9966
Heap Sort	9966	9966	9966
Quick Sort	9966	9966	500499
Counting Sort	~(1000+k)	~(1000+k)	~(1000+k)
Radix sort	~1000	1000	1

Insertion Sort

Sorted Array – (n-1)

Reverse Array – (n(n-1)/2)+(n-1)

Random Array – $(n(n+3)/4)-1-(\sum_2^n(1/i))$

Merge Sort

Sorted Array – nlogn

Reverse Array – nlogn

Random Array – nlogn

Heap Sort

Sorted Array – nlogn

Reverse Array – nlogn

Random Array – nlogn

Quick Sort

Sorted Array – nlogn

Reverse Array – $(n^2+n-2)/2$

Random Array – nlogn

Counting Sort

Sorted Array – ~(n+k)

Reverse Array – ~(n+k)

Random Array – ~(n+k)

Radix Sort

Sorted Array – n

Reverse Array – 1

Random Array – ~n

These results approximately match the Big O Notation of the respective sorting algorithms.

References: www.sanfoundry.com