Name : Jagtap Nachiket Nitin
En.No.: 21221079

Assignment 5

Amortized Analysis

g) Sorting algorithm

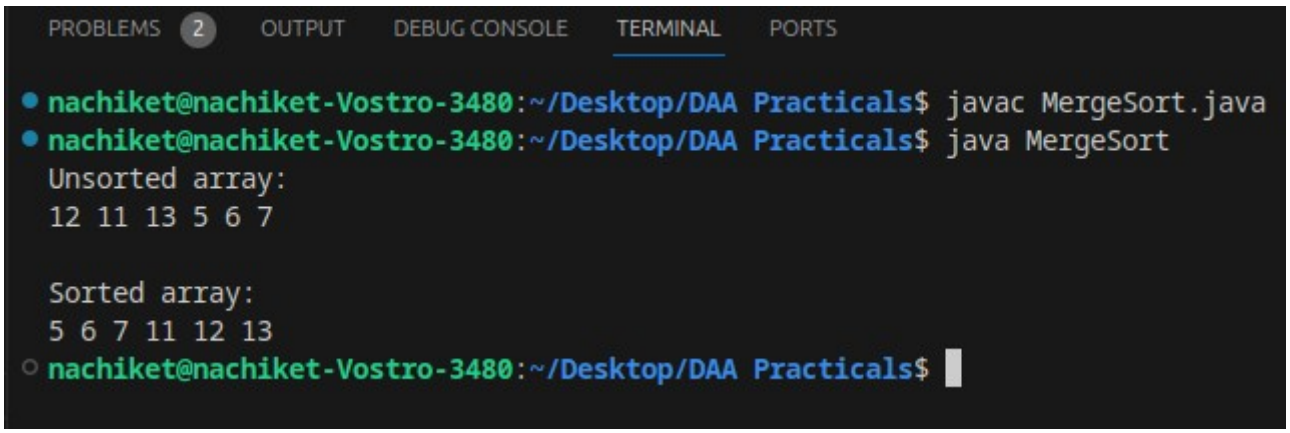
1) Merge Sort :

Java Code :

```
public class MergeSort {
public static void main(String[] args) {
int[] arr = {12, 11, 13, 5, 6, 7};

System.out.println("Unsorted array:");
printArray(arr);

mergeSort(arr, 0, arr.length - 1);

System.out.println("\nSorted array:");
printArray(arr);
}

public static void mergeSort(int[] arr, int left, int right) {
if (left < right) {
// Find the middle point of the array
int mid = (left + right) / 2;

// Recursively sort the first and second halves
mergeSort(arr, left, mid);
mergeSort(arr, mid + 1, right);

// Merge the sorted halves
merge(arr, left, mid, right);
}
}

public static void merge(int[] arr, int left, int mid, int right) {
int n1 = mid - left + 1;
int n2 = right - mid;

int[] leftArray = new int[n1];
int[] rightArray = new int[n2];

// Copy data to temp arrays leftArray[] and rightArray[]
for (int i = 0; i < n1; i++) {
leftArray[i] = arr[left + i];
}
for (int j = 0; j < n2; j++) {
```

```
rightArray[j] = arr[mid + 1 + j];
}

// Merge the temp arrays
int i = 0, j = 0, k = left;
while (i < n1 && j < n2) {
if (leftArray[i] <= rightArray[j]) {
arr[k] = leftArray[i];
i++;
} else {
arr[k] = rightArray[j];
j++;
}
k++;
}

// Copy remaining elements of leftArray[] if any
while (i < n1) {
arr[k] = leftArray[i];
i++;
k++;
}

// Copy remaining elements of rightArray[] if any
while (j < n2) {
arr[k] = rightArray[j];
j++;
k++;
}
}

public static void printArray(int[] arr) {
for (int value : arr) {
System.out.print(value + " ");
}
System.out.println();
}
}
```

Output :



2) Quick Sort :

Java Code:

```
public class QuickSort {
public static void main(String[] args) {
int[] arr = {12, 11, 13, 5, 6, 7};

System.out.println("Unsorted array:");
printArray(arr);

quickSort(arr, 0, arr.length - 1);

System.out.println("\nSorted array:");
printArray(arr);
}

public static void quickSort(int[] arr, int low, int high) {
if (low < high) {
// Partition the array into two sub-arrays
int pivotIndex = partition(arr, low, high);

// Recursively sort the sub-arrays
quickSort(arr, low, pivotIndex - 1);
quickSort(arr, pivotIndex + 1, high);
}
}

public static int partition(int[] arr, int low, int high) {
int pivot = arr[high];
int i = low - 1;

for (int j = low; j < high; j++) {
if (arr[j] < pivot) {
i++;
swap(arr, i, j);
}
}
```

```
swap(arr, i + 1, high);
return i + 1;
}

public static void swap(int[] arr, int i, int j) {
int temp = arr[i];
arr[i] = arr[j];
arr[j] = temp;
}

public static void printArray(int[] arr) {
for (int value : arr) {
System.out.print(value + " ");
}
System.out.println();
}
}
```

Output :