Assignment 1

CSE 373

Design and Analysis of Algorithms

Section 9

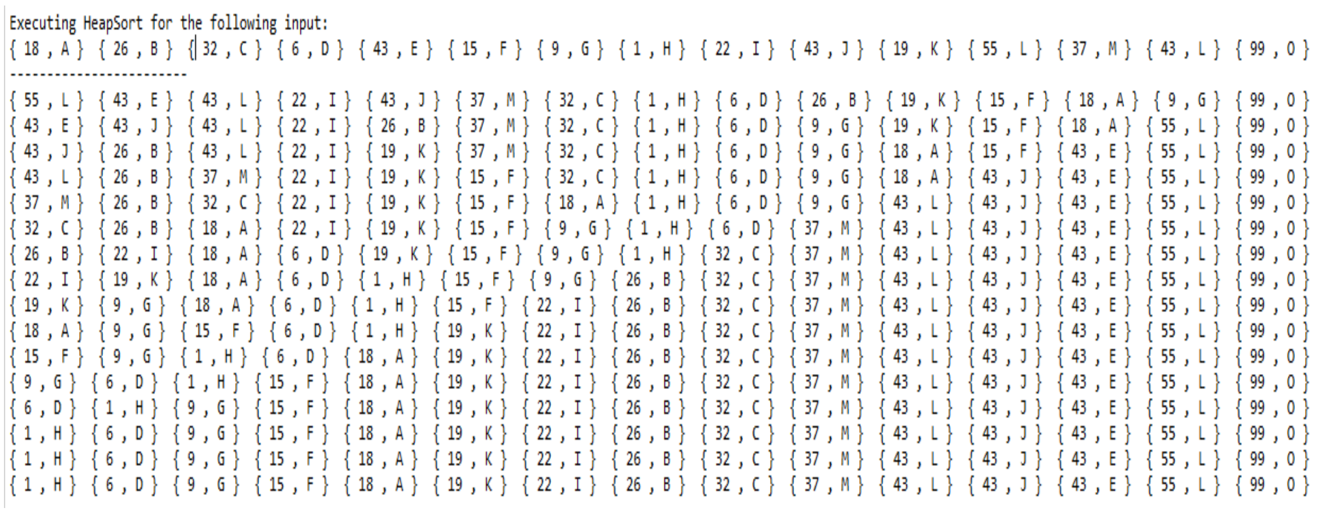
Spring 2020

North South University

Submitted To: Shaikh Shawon Arefin Shimon (SAS3)

Name	:	Md. Fahad Mojumder
Student ID	:	1712145642
Email Address	:	fahad.mojumder@northsouth.edu
Submission Date	:	15-02-2020

1. <u>Section 1:</u> Implementation of Heap Sort

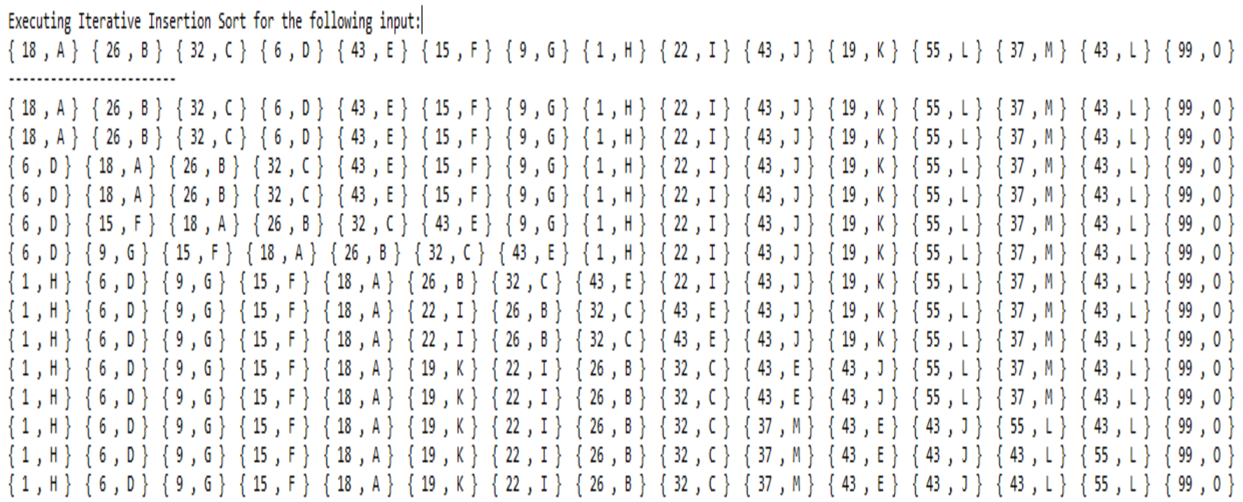


```
Executing HeapSort for the following input:
{ 18 , A } { 26 , B } { 32 , C } { 6 , D } { 43 , E } { 15 , F } { 9 , G } { 1 , H } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
------------------------
{ 55 , L } { 43 , E } { 43 , L } { 22 , I } { 43 , J } { 37 , M } { 32 , C } { 1 , H } { 6 , D } { 26 , B } { 19 , K } { 15 , F } { 18 , A } { 9 , G } { 99 , O }
{ 43 , E } { 43 , J } { 43 , L } { 22 , I } { 26 , B } { 37 , M } { 32 , C } { 1 , H } { 6 , D } { 9 , G } { 19 , K } { 15 , F } { 18 , A } { 55 , L } { 99 , O }
{ 43 , J } { 26 , B } { 43 , L } { 22 , I } { 19 , K } { 37 , M } { 32 , C } { 1 , H } { 6 , D } { 9 , G } { 18 , A } { 15 , F } { 43 , E } { 55 , L } { 99 , O }
{ 43 , L } { 26 , B } { 37 , M } { 22 , I } { 19 , K } { 15 , F } { 32 , C } { 1 , H } { 6 , D } { 9 , G } { 18 , A } { 43 , J } { 43 , E } { 55 , L } { 99 , O }
{ 37 , M } { 26 , B } { 32 , C } { 22 , I } { 19 , K } { 15 , F } { 18 , A } { 1 , H } { 6 , D } { 9 , G } { 43 , L } { 43 , J } { 43 , E } { 55 , L } { 99 , O }
{ 32 , C } { 26 , B } { 18 , A } { 22 , I } { 19 , K } { 15 , F } { 9 , G } { 1 , H } { 6 , D } { 37 , M } { 43 , L } { 43 , J } { 43 , E } { 55 , L } { 99 , O }
{ 26 , B } { 22 , I } { 18 , A } { 6 , D } { 19 , K } { 15 , F } { 9 , G } { 1 , H } { 32 , C } { 37 , M } { 43 , L } { 43 , J } { 43 , E } { 55 , L } { 99 , O }
{ 22 , I } { 19 , K } { 18 , A } { 6 , D } { 1 , H } { 15 , F } { 9 , G } { 26 , B } { 32 , C } { 37 , M } { 43 , L } { 43 , J } { 43 , E } { 55 , L } { 99 , O }
{ 19 , K } { 9 , G } { 18 , A } { 6 , D } { 1 , H } { 15 , F } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , L } { 43 , J } { 43 , E } { 55 , L } { 99 , O }
{ 18 , A } { 9 , G } { 15 , F } { 6 , D } { 1 , H } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , L } { 43 , J } { 43 , E } { 55 , L } { 99 , O }
{ 15 , F } { 9 , G } { 1 , H } { 6 , D } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , L } { 43 , J } { 43 , E } { 55 , L } { 99 , O }
{ 9 , G } { 6 , D } { 1 , H } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , L } { 43 , J } { 43 , E } { 55 , L } { 99 , O }
{ 6 , D } { 1 , H } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , L } { 43 , J } { 43 , E } { 55 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , L } { 43 , J } { 43 , E } { 55 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , L } { 43 , J } { 43 , E } { 55 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , L } { 43 , J } { 43 , E } { 55 , L } { 99 , O }
```

Figure 1: Output of Heapsort

```java
private static <E extends Comparable<E>> void heapSortInternal(E[] inputArray) {
    int n = inputArray.length;
    for (int i = n / 2 - 1; i >= 0; i--)
    Heapify(inputArray, n, i);
    for (int i = n - 1; i >= 0; i--) {
        // Move current root to end
        Data temp = (Data) inputArray[0];
        inputArray[0] = inputArray[i];
        inputArray[i] = (E) temp;
        Heapify(inputArray, i, 0);
        SortHelper.print(inputArray, inputArray.length);
    }
}
private static <E extends Comparable<E>> void Heapify(E[] inputArray, int n, int i) {
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
if (l < n && inputArray[l].compareTo(inputArray[largest]) == 1) largest = l;
    if (r < n && inputArray[r].compareTo(inputArray[largest]) == 1) largest = r;
    if (largest != i) {
        Data swap = (Data) inputArray[i];
        inputArray[i] = inputArray[largest];
        inputArray[largest] = (E) swap;
        Heapify(inputArray, n, largest);
    }
}
```

2. <u>Section 2:</u> Implementation of Iterative Insertion Sort

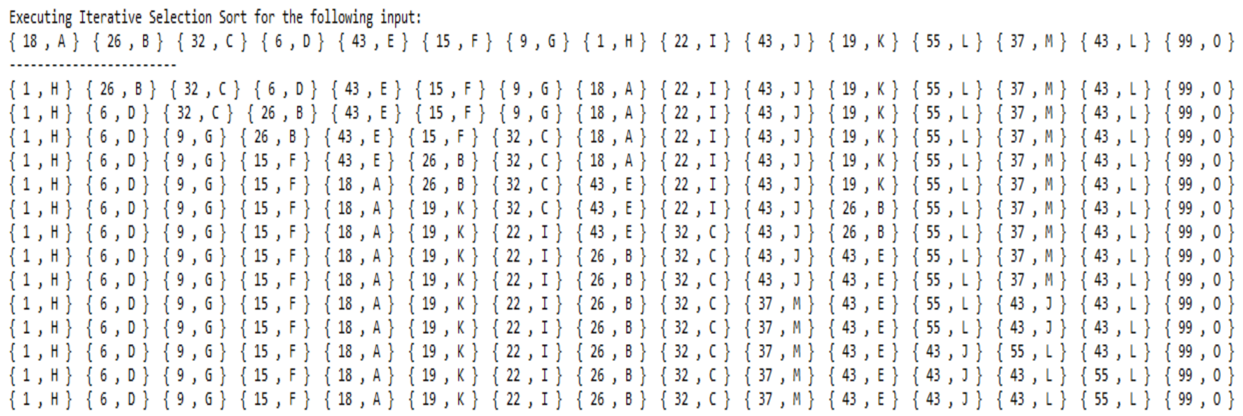


```
Executing Iterative Insertion Sort for the following input:
{ 18 , A } { 26 , B } { 32 , C } { 6 , D } { 43 , E } { 15 , F } { 9 , G } { 1 , H } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
------------------------
{ 18 , A } { 26 , B } { 32 , C } { 6 , D } { 43 , E } { 15 , F } { 9 , G } { 1 , H } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 18 , A } { 26 , B } { 32 , C } { 6 , D } { 43 , E } { 15 , F } { 9 , G } { 1 , H } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 6 , D } { 18 , A } { 26 , B } { 32 , C } { 43 , E } { 15 , F } { 9 , G } { 1 , H } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 6 , D } { 18 , A } { 26 , B } { 32 , C } { 43 , E } { 15 , F } { 9 , G } { 1 , H } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 6 , D } { 15 , F } { 18 , A } { 26 , B } { 32 , C } { 43 , E } { 9 , G } { 1 , H } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 6 , D } { 9 , G } { 15 , F } { 18 , A } { 26 , B } { 32 , C } { 43 , E } { 1 , H } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 26 , B } { 32 , C } { 43 , E } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 22 , I } { 26 , B } { 32 , C } { 43 , E } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 22 , I } { 26 , B } { 32 , C } { 43 , E } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 43 , E } { 43 , J } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 43 , E } { 43 , J } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , E } { 43 , J } { 55 , L } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , E } { 43 , J } { 43 , L } { 55 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , E } { 43 , J } { 43 , L } { 55 , L } { 99 , O }
```

Figure 2: Output of Iterative Insertion Sort

```java
private static <E extends Comparable<E>> void sortInternal(E[] inputArray, int size){
for(int i=1;i<size;++i)
  {
        Data key = (Data) inputArray[i];
        int j = i-1;
           while(j>=0 && inputArray[j].compareTo((E) key)==1)
        {
            inputArray[j + 1] = inputArray[j];
            j = j - 1;
        }
        inputArray[j+1] = (E)key;
        SortHelper.print(inputArray,inputArray.length);
   }
 }
```

3. <u>Section 3:</u> Implementation of Iterative Selection Sort

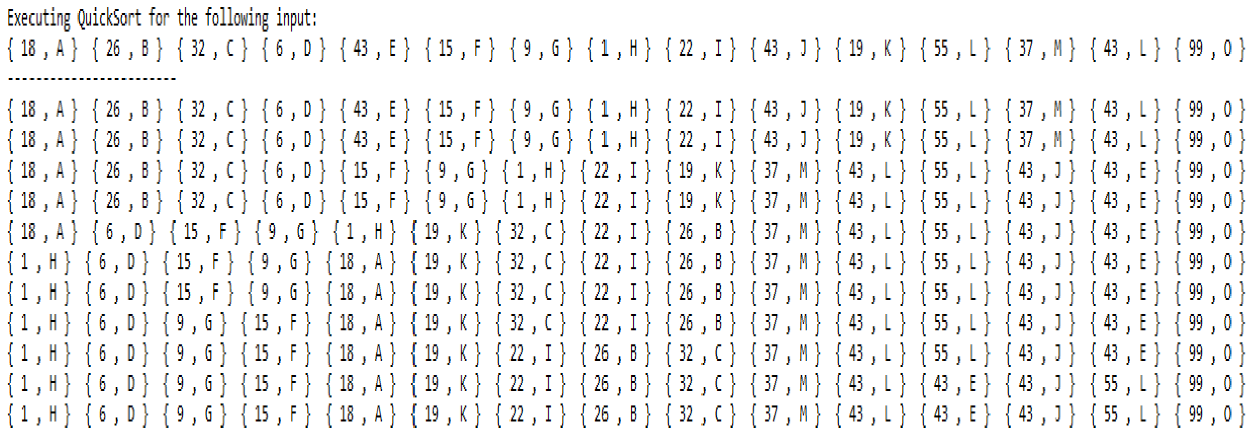


```
Executing Iterative Selection Sort for the following input:
{ 18 , A } { 26 , B } { 32 , C } { 6 , D } { 43 , E } { 15 , F } { 9 , G } { 1 , H } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
-----------------------
{ 1 , H } { 26 , B } { 32 , C } { 6 , D } { 43 , E } { 15 , F } { 9 , G } { 18 , A } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 32 , C } { 26 , B } { 43 , E } { 15 , F } { 9 , G } { 18 , A } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 26 , B } { 43 , E } { 15 , F } { 32 , C } { 18 , A } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 43 , E } { 26 , B } { 32 , C } { 18 , A } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 26 , B } { 32 , C } { 43 , E } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 32 , C } { 43 , E } { 22 , I } { 43 , J } { 26 , B } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 43 , E } { 32 , C } { 43 , J } { 26 , B } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 43 , J } { 43 , E } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 43 , J } { 43 , E } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , E } { 55 , L } { 43 , J } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , E } { 55 , L } { 43 , J } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , E } { 43 , J } { 55 , L } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , E } { 43 , J } { 43 , L } { 55 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , E } { 43 , J } { 43 , L } { 55 , L } { 99 , O }
```

Figure 3: Output of Iterative Selection Sort

```java
private static <E extends Comparable<E>> void sortInternal(E[] inputArray
        , int size) {
    for (int i = 0; i < size - 1; i++) {
        int min_index = i;
        for (int j = i + 1; j < size; j++) {
            if (inputArray[j].compareTo(inputArray[min_index]) == -1) {
                min_index = j;
            }
        }
        Data temp = (Data) inputArray[min_index];
        inputArray[min_index] = inputArray[i];
        inputArray[i] = (E) temp;
        SortHelper.print(inputArray, inputArray.length);
    }
}
```

4. Section 4: Implementation of Quick Sort

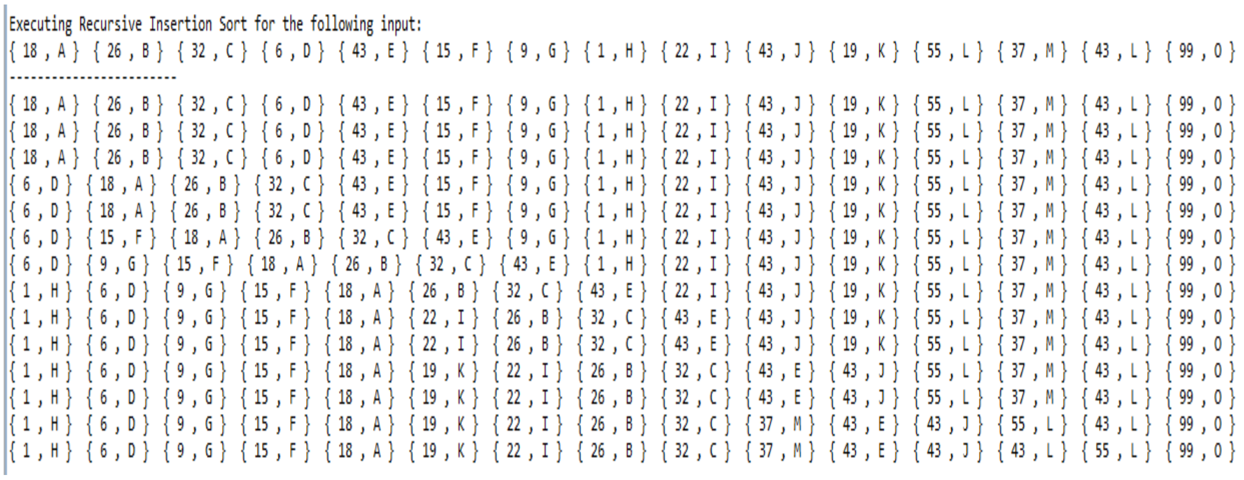


```
Executing QuickSort for the following input:
{ 18 , A } { 26 , B } { 32 , C } { 6 , D } { 43 , E } { 15 , F } { 9 , G } { 1 , H } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
-----------------------
{ 18 , A } { 26 , B } { 32 , C } { 6 , D } { 43 , E } { 15 , F } { 9 , G } { 1 , H } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 18 , A } { 26 , B } { 32 , C } { 6 , D } { 43 , E } { 15 , F } { 9 , G } { 1 , H } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 18 , A } { 26 , B } { 32 , C } { 6 , D } { 15 , F } { 9 , G } { 1 , H } { 22 , I } { 19 , K } { 37 , M } { 43 , L } { 55 , L } { 43 , J } { 43 , E } { 99 , O }
{ 18 , A } { 26 , B } { 32 , C } { 6 , D } { 15 , F } { 9 , G } { 1 , H } { 22 , I } { 19 , K } { 37 , M } { 43 , L } { 55 , L } { 43 , J } { 43 , E } { 99 , O }
{ 18 , A } { 6 , D } { 15 , F } { 9 , G } { 1 , H } { 19 , K } { 32 , C } { 22 , I } { 26 , B } { 37 , M } { 43 , L } { 55 , L } { 43 , J } { 43 , E } { 99 , O }
{ 1 , H } { 6 , D } { 15 , F } { 9 , G } { 18 , A } { 19 , K } { 32 , C } { 22 , I } { 26 , B } { 37 , M } { 43 , L } { 55 , L } { 43 , J } { 43 , E } { 99 , O }
{ 1 , H } { 6 , D } { 15 , F } { 9 , G } { 18 , A } { 19 , K } { 32 , C } { 22 , I } { 26 , B } { 37 , M } { 43 , L } { 55 , L } { 43 , J } { 43 , E } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 32 , C } { 22 , I } { 26 , B } { 37 , M } { 43 , L } { 55 , L } { 43 , J } { 43 , E } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , L } { 55 , L } { 43 , J } { 43 , E } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , L } { 43 , E } { 43 , J } { 55 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , L } { 43 , E } { 43 , J } { 55 , L } { 99 , O }
```

Figure 4: Output of Quick Sort

```
         public static<E extends Comparable<E>> void quickSort(E[] inputArray, int low, int
high){
             if (low < high)
             {
                 SortHelper.print(inputArray,inputArray.length);
                 int part_index = (int) part(inputArray, low, high);
                 quickSort(inputArray, low, part_index-1);
                 quickSort(inputArray, part_index+1, high);
             }
         }
                 public static<E extends Comparable<E>>int part(E[]  inputArray,int low, int
high)
         {
             Data pivot = (Data) inputArray[high];
             int i = low-1;
                    for(int j=low;j<high;j++)
             {
                 if(inputArray[j].compareTo((E) pivot)==-1)
                 {
                     i++;
                     Data temp = (Data) inputArray[i];
                     inputArray[i] = inputArray[j];
                     inputArray[j]  = (E) temp;
                 }
             }
                    Data temp = (Data) inputArray[i+1];
             inputArray[i+1] = inputArray[high];
             inputArray[high] = (E) temp;
             return i+1;
         }
```

5. <u>Section 5:</u> Implementation of Recursive Insertion Sort

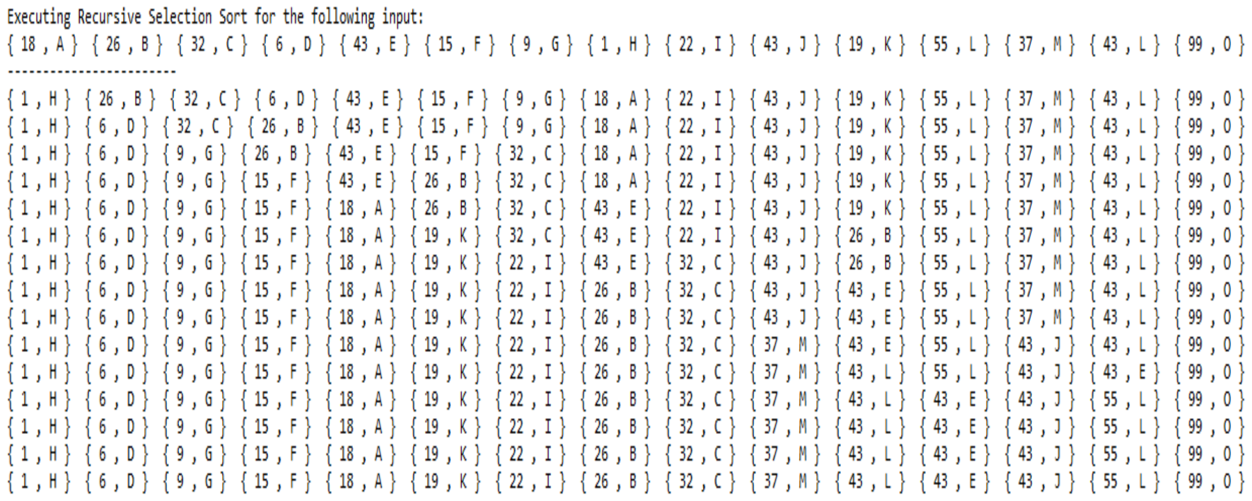


```
Executing Recursive Insertion Sort for the following input:
{ 18 , A } { 26 , B } { 32 , C } { 6 , D } { 43 , E } { 15 , F } { 9 , G } { 1 , H } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
------------------------
{ 18 , A } { 26 , B } { 32 , C } { 6 , D } { 43 , E } { 15 , F } { 9 , G } { 1 , H } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 18 , A } { 26 , B } { 32 , C } { 6 , D } { 43 , E } { 15 , F } { 9 , G } { 1 , H } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 18 , A } { 26 , B } { 32 , C } { 6 , D } { 43 , E } { 15 , F } { 9 , G } { 1 , H } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 6 , D } { 18 , A } { 26 , B } { 32 , C } { 43 , E } { 15 , F } { 9 , G } { 1 , H } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 6 , D } { 18 , A } { 26 , B } { 32 , C } { 43 , E } { 15 , F } { 9 , G } { 1 , H } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 6 , D } { 15 , F } { 18 , A } { 26 , B } { 32 , C } { 43 , E } { 9 , G } { 1 , H } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 6 , D } { 9 , G } { 15 , F } { 18 , A } { 26 , B } { 32 , C } { 43 , E } { 1 , H } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 26 , B } { 32 , C } { 43 , E } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 22 , I } { 26 , B } { 32 , C } { 43 , E } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 22 , I } { 26 , B } { 32 , C } { 43 , E } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 43 , E } { 43 , J } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 43 , E } { 43 , J } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , E } { 43 , J } { 55 , L } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , E } { 43 , J } { 43 , L } { 55 , L } { 99 , O }
```

Figure 5: Output of Recursive Insertion Sort

```
    private static <E extends Comparable<E>> void sortInternal(E[] inputArray
            , int size) {
        if (size <= 1) return;
        sortInternal(inputArray, size - 1);
        SortHelper.print(inputArray, inputArray.length);
        Data end = (Data) inputArray[size - 1];
        int i = size - 2;
        while (i >= 0 && inputArray[i].compareTo((E) end) == 1) {
            inputArray[i + 1] = inputArray[i];
            i--;
        }
        inputArray[i + 1] = (E) end;
      }
   }
```

6. Section 5: Implementation of Recursive Selection Sort



```
Executing Recursive Selection Sort for the following input:
{ 18 , A } { 26 , B } { 32 , C } { 6 , D } { 43 , E } { 15 , F } { 9 , G } { 1 , H } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
------------------------
{ 1 , H } { 26 , B } { 32 , C } { 6 , D } { 43 , E } { 15 , F } { 9 , G } { 18 , A } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 32 , C } { 26 , B } { 43 , E } { 15 , F } { 9 , G } { 18 , A } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 26 , B } { 43 , E } { 15 , F } { 32 , C } { 18 , A } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 43 , E } { 26 , B } { 32 , C } { 18 , A } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 26 , B } { 32 , C } { 43 , E } { 22 , I } { 43 , J } { 19 , K } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 32 , C } { 43 , E } { 22 , I } { 43 , J } { 26 , B } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 43 , E } { 32 , C } { 43 , J } { 26 , B } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 43 , J } { 43 , E } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 43 , J } { 43 , E } { 55 , L } { 37 , M } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , E } { 55 , L } { 43 , J } { 43 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , L } { 55 , L } { 43 , J } { 43 , E } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , L } { 43 , E } { 43 , J } { 55 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , L } { 43 , E } { 43 , J } { 55 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , L } { 43 , E } { 43 , J } { 55 , L } { 99 , O }
{ 1 , H } { 6 , D } { 9 , G } { 15 , F } { 18 , A } { 19 , K } { 22 , I } { 26 , B } { 32 , C } { 37 , M } { 43 , L } { 43 , E } { 43 , J } { 55 , L } { 99 , O }
```

Figure 6: Output of Recursive Selection Sort

```
    private static <E extends Comparable<E>> void sortInternal(E[] inputArray
            , int size, int index) {
        if (index == size) return;
        int k = minIndex((Data[]) inputArray, index, size - 1);
        if (k != index) {
            Data temp = (Data) inputArray[k];
            inputArray[k] = inputArray[index];
            inputArray[index] = (E) temp;
        }
        SortHelper.print(inputArray, inputArray.length);
        sortInternal(inputArray, size, index + 1);
    }
    //for comparing index
    static int minIndex(Data a[], int i, int j) {
        if (i == j) return i;
        int k = minIndex(a, i + 1, j);
        return (a[i].compareTo(a[k]) == -1) ? i : k;
    }
```