Programmer: Michael Lankford

Course: CSCI 3352

Date: 2/20/2022

Programming Assignment: #1

Environment: Apache Netbeans IDE 12.5 running JDK 1.8 on Windows

Program #1: Quicksort.java

Objective: Implement the quicksort algorithm to sort an array

Scope: A non-distinct array with at least 30 elements

Limitations: None

Input: int[] arrayUnsorted1 and int[] arraySemisorted1

Preconditions: Array elements must be integers

Output: The sorted versions of int[] arrayUnsorted1 and int[] arraySemisorted1

Postconditions: The array will be outputted in ascending order

Algorithm:

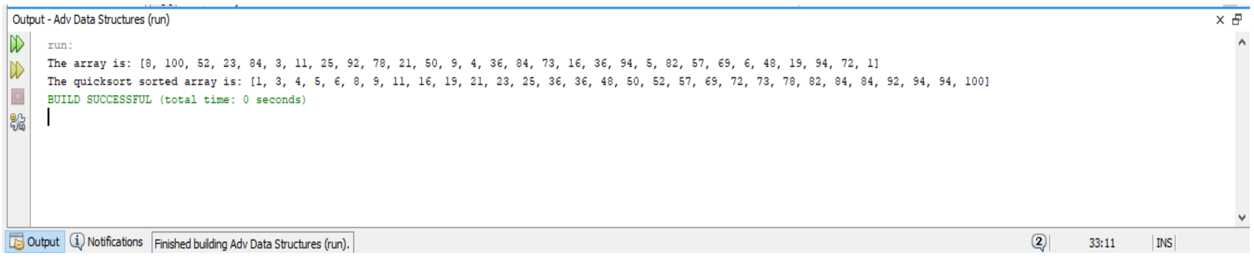
```
Quicksort(numArray, partition, pivot)
    q = call Partition(numArray, partition, pivot)
    call Quicksort(numArray, partition, q – 1)
    call Quicksort(numArray, q + 1, pivot)
```

Program Code:

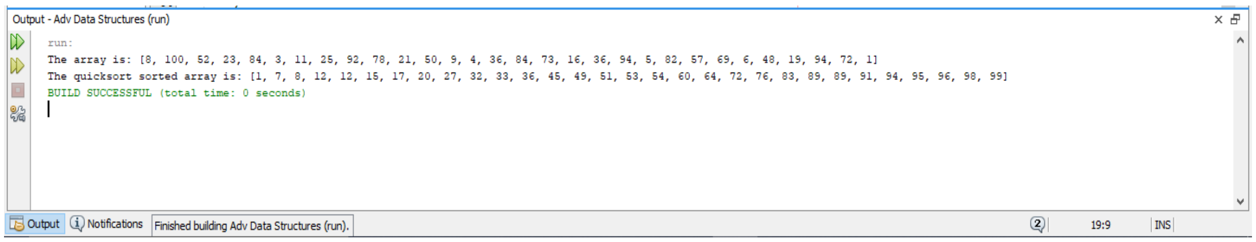
```
public class Quicksort
{
      public void Quicksort(int[] numArray, int partition, int pivot)
                                                       //Quicksort method
   {
       if (partition < pivot)
                                                       //If partition < pivot call
                                                Partition to rearrange the subarray
       {
           int q = Partition(numArray, partition, pivot);
           Quicksort(numArray, partition, q - 1);
                                                       //Pass lower subarray to
                                                Quicksort
           Quicksort(numArray, q + 1, pivot);
                                                       //Pass upper subarray to
                                                Quicksort
       }
```

```
    }

    public int Partition(int[] numArray, int partition, int pivot)
                                                        //Partition method
    {
        int x = numArray[pivot];                        //Select pivot element
        int i = partition - 1;

        for (int j = partition; j < pivot; j++)
                                                        //For subarray elements check to
                                                 see if pivot > current element
        {
            if (numArray[j] <= x)
                                                         //If current element is <= pivot
                                                 element then swap them and increment i
                                                 for next element
            {
                i++;
                int tempElement = numArray[j];
                numArray[j] = numArray[i];
                numArray[i] = tempElement;
            }
        }

        int tempElement = numArray[i + 1];
                                                        //Swap pivot element with
                                                 leftmost element greater than x

        numArray[i + 1] = numArray[pivot];
        numArray[pivot] = tempElement;

        return (i + 1);
    }
}
```

Outputs:



```
run:
The array is: [8, 100, 52, 23, 84, 3, 11, 25, 92, 78, 21, 50, 9, 4, 36, 84, 73, 16, 36, 94, 5, 82, 57, 69, 6, 48, 19, 94, 72, 1]
The quicksort sorted array is: [1, 3, 4, 5, 6, 8, 9, 11, 16, 19, 21, 23, 25, 36, 36, 48, 50, 52, 57, 69, 72, 73, 78, 82, 84, 84, 92, 94, 94, 100]
BUILD SUCCESSFUL (total time: 0 seconds)
```



```
run:
The array is: [8, 100, 52, 23, 84, 3, 11, 25, 92, 78, 21, 50, 9, 4, 36, 84, 73, 16, 36, 94, 5, 82, 57, 69, 6, 48, 19, 94, 72, 1]
The quicksort sorted array is: [1, 7, 8, 12, 12, 15, 17, 20, 27, 32, 33, 36, 45, 49, 51, 53, 54, 60, 64, 72, 76, 83, 89, 89, 91, 94, 95, 96, 98, 99]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Program #2: Heapsort.java

Objective: Implement the heapsort algorithm to sort an array

Scope: A non-distinct array with at least 30 elements

Limitations: None

Input: int[] arrayUnsorted2 and int[] arraySemisorted2

Preconditions: Array elements must be integers

Output: The sorted versions of int[] arrayUnsorted1 and int[] arraySemisorted1

Postconditions: The array will be outputted in ascending order

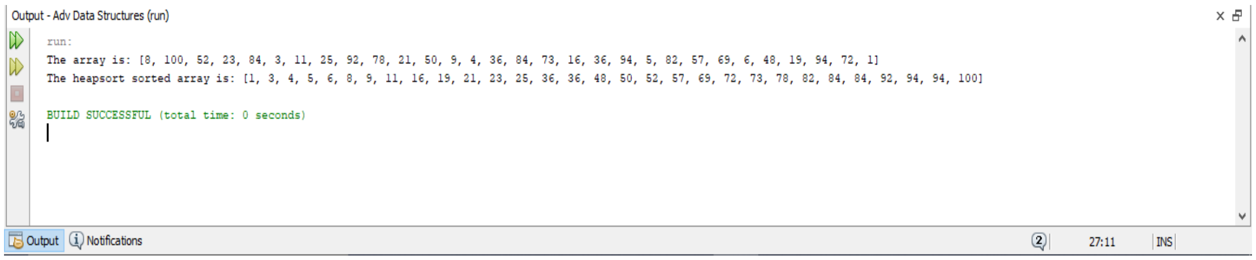
Algorithm:

```
Heapsort(numArray)
        call buildMaxHeap(numArray, arrayLength)
        for arrayLength – 1
            switch current array element with array 0
            call Heapify(numArray, i, 0)
```

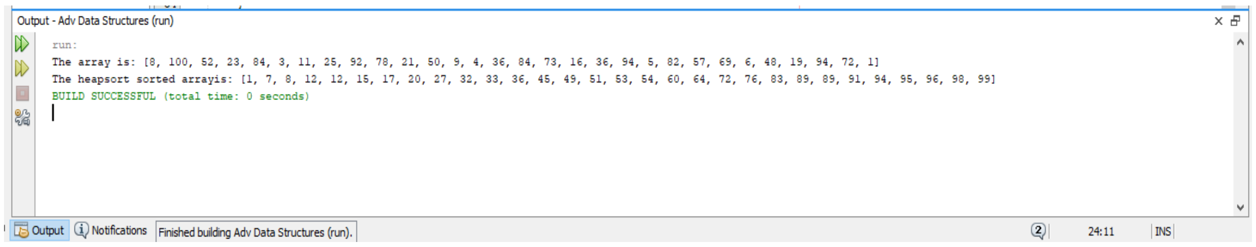
Program Code:

```
public class Heapsort
{
    public void Heapify(int[] numArray, int arrayLength, int root)
                                                 //Heapify method
    {
        int largest = root;
        int left = 2 * root + 1;
        int right = 2 * root + 2;

        if (left < arrayLength && numArray[left] > numArray[largest])
                                                 //If left child is larger than
                                          the root, swap them, else, leave them
        {
            largest = left;
        }

        if (right < arrayLength && numArray[right] > numArray[largest])
                                                 //If right child is larger than
                                          the root, swap them, else, leave them
        {
            largest = right;
        }

        if (largest != root)
                                                 //If the largest != root then
                                          swap them and call Heapify again
        {
            int temp = numArray[root];
            numArray[root] = numArray[largest];
            numArray[largest] = temp;

            Heapify(numArray, arrayLength, largest);
        }
    }

    public void buildMaxHeap(int[] numArray, int arrayLength)
                                                    //buildMaxHeap method
    {
        for (int i = arrayLength / 2 - 1; i >= 0; i--)
                                                    //For tree nodes run Heapify on
                                             each 1 element heap
        {
            Heapify(numArray, arrayLength, i);
        }
    }

    public void Heapsort(int[] numArray)            //Heapsort method
    {
        int arrayLength = numArray.length;

        buildMaxHeap(numArray, arrayLength);
                                                    //Call buildMaxHeap to build the
                                             heap

        for (int i = arrayLength - 1; i >= 0; i--)
                                                    //For the array, swap element 0
                                             with the current element and call
                                             Heapify
        {
            int temp = numArray[0];
            numArray[0] = numArray[i];
            numArray[i] = temp;

            Heapify(numArray, i, 0);
        }
    }
}
```

Outputs:



```
Output - Adv Data Structures (run)
run:
The array is: [8, 100, 52, 23, 84, 3, 11, 25, 92, 78, 21, 50, 9, 4, 36, 84, 73, 16, 36, 94, 5, 82, 57, 69, 6, 48, 19, 94, 72, 1]
The heapsort sorted array is: [1, 3, 4, 5, 6, 8, 9, 11, 16, 19, 21, 23, 25, 36, 36, 48, 50, 52, 57, 69, 72, 73, 78, 82, 84, 84, 92, 94, 94, 100]

BUILD SUCCESSFUL (total time: 0 seconds)
```



```
Output - Adv Data Structures (run)
run:
The array is: [8, 100, 52, 23, 84, 3, 11, 25, 92, 78, 21, 50, 9, 4, 36, 84, 73, 16, 36, 94, 5, 82, 57, 69, 6, 48, 19, 94, 72, 1]
The heapsort sorted arrayis: [1, 7, 8, 12, 12, 15, 17, 20, 27, 32, 33, 36, 45, 49, 51, 53, 54, 60, 64, 72, 76, 83, 89, 89, 91, 94, 95, 96, 98, 99]
BUILD SUCCESSFUL (total time: 0 seconds)
```