DS-Assignment - 10

Name: Shravani kaware 202152316

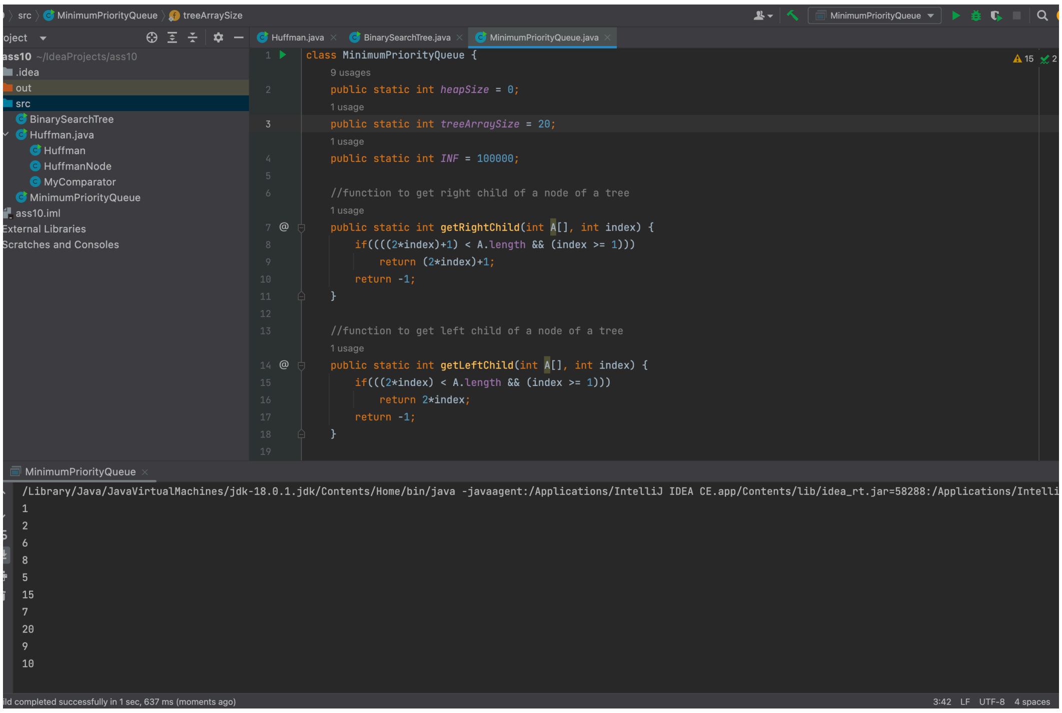
Q1).

```
class MinimumPriorityQueue {
  public static int heapSize = 0;
  public static int treeArraySize = 20;
  public static int INF = 100000;

  //function to get right child of a node of a tree
  public static int getRightChild(int A[], int index)
{
    if((((2*index)+1) < A.length && (index >= 1)))
      return (2*index)+1;
    return -1;
  }

  //function to get left child of a node of a tree
  public static int getLeftChild(int A[], int index)
{
      if(((2*index) < A.length && (index >= 1)))
          return 2*index;
      return -1;
  }

  //function to get the parent of a node of a tree
  public static int getParent(int A[], int index) {
    if ((index > 1) && (index < A.length)) {
      return index/2;
    }
    return -1;
  }

  public static void minHeapify(int A[], int index) {
    int leftChildIndex = getLeftChild(A, index);
    int rightChildIndex = getRightChild(A, index);
```

```
    // finding smallest among index, left child and right child
    int smallest = index;

    if ((leftChildIndex <= heapSize) && (leftChildIndex>0)) {
      if (A[leftChildIndex] < A[smallest]) {
        smallest = leftChildIndex;
      }
    }

    if ((rightChildIndex <= heapSize && (rightChildIndex>0))) {
      if (A[rightChildIndex] < A[smallest]) {
        smallest = rightChildIndex;
      }
    }

    // smallest is not the node, node is not a heap
    if (smallest != index) {
      int temp;
      //swapping
      temp = A[smallest];
      A[smallest] = A[index];
      A[index] = temp;
      minHeapify(A, smallest);
    }
  }

  public static void buildMinHeap(int A[]) {
    for(int i=heapSize/2; i>=1; i--) {
      minHeapify(A, i);
    }
  }

  public static int minimum(int A[]) {
    return A[1];
  }

  public static int extractMin(int A[]) {
```

```java
        int minm = A[1];
        A[1] = A[heapSize];
        heapSize--;
        minHeapify(A, 1);
        return minm;
    }

    public static void decreaseKey(int A[], int index,
int key) {
        A[index] = key;
        while((index>1) && (A[getParent(A, index)] >
A[index])) {
            int temp;
            temp = A[getParent(A, index)];
            A[getParent(A, index)] = A[index];
            A[index] = temp;
            index = getParent(A, index);
        }
    }

    public static void increaseKey(int A[], int index,
int key) {
        A[index] = key;
        minHeapify(A, index);
    }

    public static void insert(int A[], int key) {
        heapSize++;
        A[heapSize] = INF;
        decreaseKey(A, heapSize, key);
    }

    public static void printHeap(int A[]) {
        for(int i=1; i<=heapSize; i++) {
            System.out.println(A[i]);
        }
        System.out.println("");
    }

    public static void main(String[] args) {
```

```java
    int A[] = new int[treeArraySize];
    buildMinHeap(A);

    insert(A, 20);
    insert(A, 15);
    insert(A, 8);
    insert(A, 10);
    insert(A, 5);
    insert(A, 7);
    insert(A, 6);
    insert(A, 2);
    insert(A, 9);
    insert(A, 1);

    printHeap(A);

    increaseKey(A, 5, 22);
    printHeap(A);

    System.out.println(minimum(A));
    System.out.println("");
    System.out.println(extractMin(A));
    System.out.println("");

    printHeap(A);

    System.out.println(extractMin(A));
    System.out.println(extractMin(A));
    System.out.println(extractMin(A));
    System.out.println(extractMin(A));
    System.out.println(extractMin(A));
    System.out.println(extractMin(A));
    System.out.println(extractMin(A));
    System.out.println(extractMin(A));
    System.out.println(extractMin(A));
  }
}
```


src
MinimumPriorityQueue
treeArraySize
MinimumPriorityQueue
Huffman.java
BinarySearchTree.java
MinimumPriorityQueue.java
ass10 ~/IdeaProjects/ass10
.idea
out
src
BinarySearchTree
Huffman.java
Huffman
HuffmanNode
MyComparator
MinimumPriorityQueue
ass10.iml
External Libraries
Scratches and Consoles
class MinimumPriorityQueue {
9 usages
public static int heapSize = 0;
1 usage
public static int treeArraySize = 20;
1 usage
public static int INF = 100000;
//function to get right child of a node of a tree
1 usage
public static int getRightChild(int A[], int index) {
if((((2*index)+1) < A.length && (index >= 1)))
return (2*index)+1;
return -1;
}
//function to get left child of a node of a tree
1 usage
public static int getLeftChild(int A[], int index) {
if(((2*index) < A.length && (index >= 1)))
return 2*index;
return -1;
}
MinimumPriorityQueue
/Library/Java/JavaVirtualMachines/jdk-18.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=58288:/Applications/Intelli
1
2
6
8
5
15
7
20
9
10
ld completed successfully in 1 sec, 637 ms (moments ago)
3:42 LF UTF-8 4 spaces

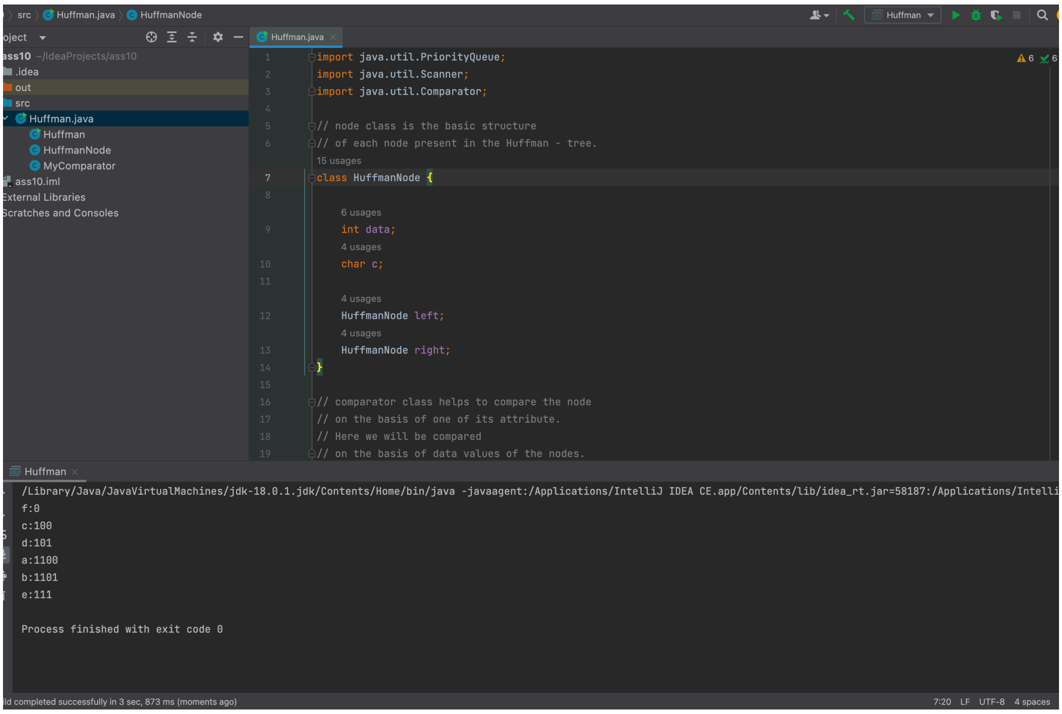
Caption

Q2).

```
import java.util.PriorityQueue;
import java.util.Scanner;
import java.util.Comparator;

// node class is the basic structure
// of each node present in the Huffman - tree.
class HuffmanNode {

	int data;
	char c;

	HuffmanNode left;
	HuffmanNode right;
}

// comparator class helps to compare the node
// on the basis of one of its attribute.
// Here we will be compared
// on the basis of data values of the nodes.
class MyComparator implements Comparator<HuffmanNode> {
	public int compare(HuffmanNode x, HuffmanNode y)
	{

		return x.data - y.data;
	}
}

public class Huffman {

	// recursive function to print the
	// huffman-code through the tree traversal.
	// Here s is the huffman - code generated.
	public static void printCode(HuffmanNode root, String s)
	{

		// base case; if the left and right are null
		// then its a leaf node and we print
		// the code s generated by traversing the tree.
		if (root.left
				== null
			&& root.right
				== null
			&& Character.isLetter(root.c)) {

			// c is the character in the node
			System.out.println(root.c + ":" + s);

			return;
		}

		// if we go to left then add "0" to the code.
		// if we go to the right add"1" to the code.

		// recursive calls for left and
		// right sub-tree of the generated tree.
		printCode(root.left, s + "0");
		printCode(root.right, s + "1");
	}
```

```
// main function
public static void main(String[] args)
{

        Scanner s = new Scanner(System.in);

        // number of characters.
        int n = 6;
        char[] charArray = { 'a', 'b', 'c', 'd', 'e', 'f' };
        int[] charfreq = { 5, 9, 12, 13, 16, 45 };

        // creating a priority queue q.
        // makes a min-priority queue(min-heap).
        PriorityQueue<HuffmanNode> q
                = new PriorityQueue<HuffmanNode>(n, new MyComparator());

        for (int i = 0; i < n; i++) {

                // creating a Huffman node object
                // and add it to the priority queue.
                HuffmanNode hn = new HuffmanNode();

                hn.c = charArray[i];
                hn.data = charfreq[i];

                hn.left = null;
                hn.right = null;

                // add functions adds
                // the huffman node to the queue.
                q.add(hn);
        }

        // create a root node
        HuffmanNode root = null;

        // Here we will extract the two minimum value
        // from the heap each time until
        // its size reduces to 1, extract until
        // all the nodes are extracted.
        while (q.size() > 1) {

                // first min extract.
                HuffmanNode x = q.peek();
                q.poll();

                // second min extract.
                HuffmanNode y = q.peek();
                q.poll();

                // new node f which is equal
                HuffmanNode f = new HuffmanNode();

                // to the sum of the frequency of the two nodes
                // assigning values to the f node.
                f.data = x.data + y.data;
                f.c = '-';

                // first extracted node as left child.
                f.left = x;
```

```
            // second extracted node as the right child.
            f.right = y;

            // marking the f node as the root node.
            root = f;

            // add this node to the priority-queue.
            q.add(f);
        }

        // print the codes by traversing the tree
        printCode(root, "");
    }
}
```

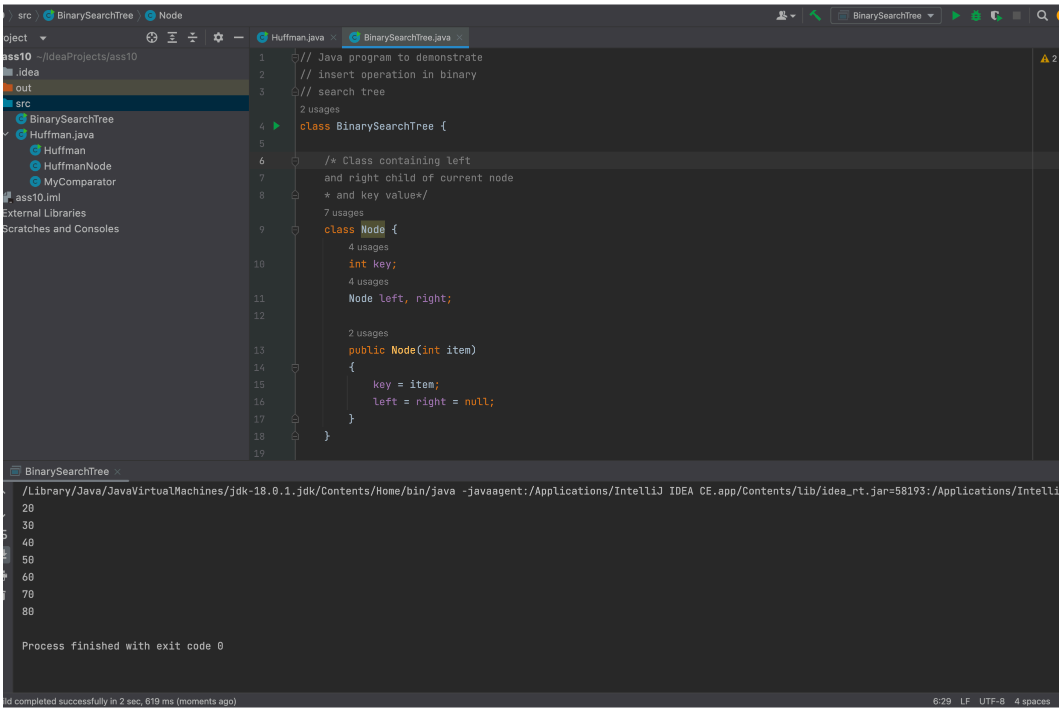

src › Huffman.java › HuffmanNode
Huffman
Huffman.java
ass10 ~/IdeaProjects/ass10
.idea
out
src
Huffman.java
Huffman
HuffmanNode
MyComparator
ass10.iml
External Libraries
Scratches and Consoles
import java.util.PriorityQueue;
import java.util.Scanner;
import java.util.Comparator;
// node class is the basic structure
// of each node present in the Huffman - tree.
15 usages
class HuffmanNode {
6 usages
int data;
4 usages
char c;
4 usages
HuffmanNode left;
4 usages
HuffmanNode right;
}
// comparator class helps to compare the node
// on the basis of one of its attribute.
// Here we will be compared
// on the basis of data values of the nodes.
Huffman
/Library/Java/JavaVirtualMachines/jdk-18.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=58187:/Applications/Intelli
f:0
c:100
d:101
a:1100
b:1101
e:111
Process finished with exit code 0
ld completed successfully in 3 sec, 873 ms (moments ago)
7:20 LF UTF-8 4 spaces

Caption

Q3). // Java program to demonstrate

```
// insert operation in binary
// search tree
class BinarySearchTree {

	/* Class containing left
	and right child of current node
	* and key value*/
	class Node {
		int key;
		Node left, right;

		public Node(int item)
		{
			key = item;
			left = right = null;
		}
	}

	// Root of BST
	Node root;

	// Constructor
	BinarySearchTree() { root = null; }

	BinarySearchTree(int value) { root = new Node(value); }

	// This method mainly calls insertRec()
	void insert(int key) { root = insertRec(root, key); }

	/* A recursive function to
	insert a new key in BST */
	Node insertRec(Node root, int key)
	{

		/* If the tree is empty,
		return a new node */
		if (root == null) {
			root = new Node(key);
			return root;
		}

		/* Otherwise, recur down the tree */
		if (key < root.key)
			root.left = insertRec(root.left, key);
		else if (key > root.key)
			root.right = insertRec(root.right, key);

		/* return the (unchanged) node pointer */
		return root;
	}

	// This method mainly calls InorderRec()
	void inorder() { inorderRec(root); }

	// A utility function to
	// do inorder traversal of BST
	void inorderRec(Node root)
	{
		if (root != null) {
```

```
			inorderRec(root.left);
			System.out.println(root.key);
			inorderRec(root.right);
		}
	}

	// Driver Code
	public static void main(String[] args)
	{
		BinarySearchTree tree = new BinarySearchTree();

		/* Let us create following BST
			50
		/	 \
		30	 70
		/ \ / \
	20 40 60 80 */
		tree.insert(50);
		tree.insert(30);
		tree.insert(20);
		tree.insert(40);
		tree.insert(70);
		tree.insert(60);
		tree.insert(80);

		// print inorder traversal of the BST
		tree.inorder();
	}
}
```


src
BinarySearchTree
Node
BinarySearchTree
Huffman.java
BinarySearchTree.java
ass10 ~/IdeaProjects/ass10
.idea
out
src
BinarySearchTree
Huffman.java
Huffman
HuffmanNode
MyComparator
ass10.iml
External Libraries
Scratches and Consoles
// Java program to demonstrate
// insert operation in binary
// search tree
2 usages
class BinarySearchTree {
/* Class containing left
and right child of current node
* and key value*/
7 usages
class Node {
4 usages
int key;
4 usages
Node left, right;
2 usages
public Node(int item)
{
key = item;
left = right = null;
}
}
BinarySearchTree
/Library/Java/JavaVirtualMachines/jdk-18.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=58193:/Applications/Intelli
20
30
40
50
60
70
80
Process finished with exit code 0
ld completed successfully in 2 sec, 619 ms (moments ago)
6:29
LF
UTF-8
4 spaces

Caption