Design And Analysis Of Algorithms Practical

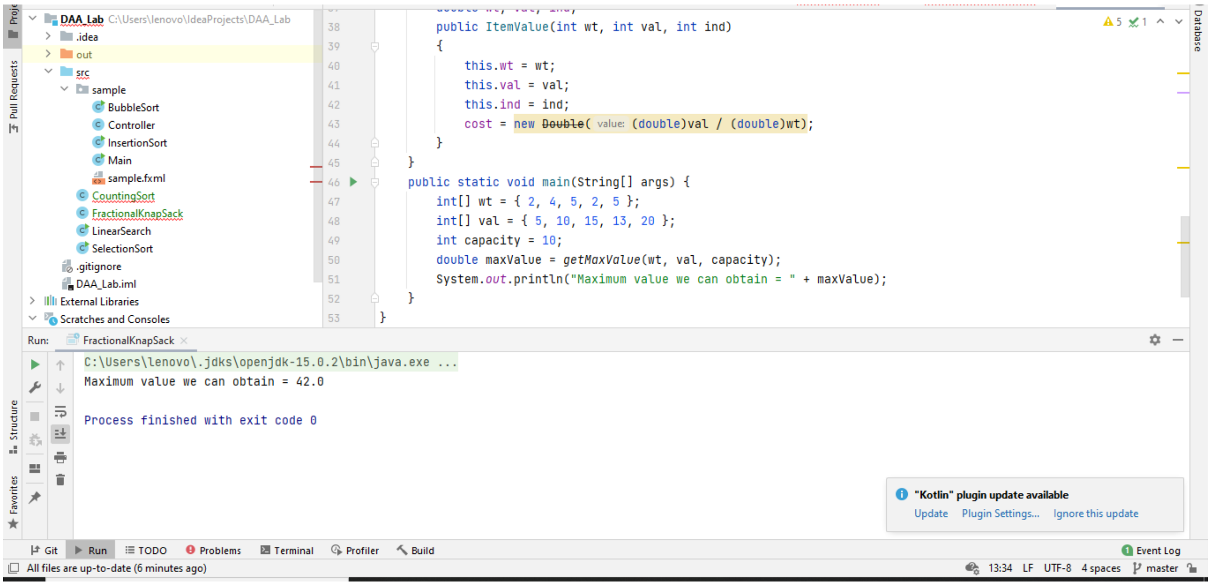
Objective:- Implement and analyze the complexity of Fractional KnapSack Greedy Approach.

Code:-

```
import java.util.Arrays;
import java.util.Comparator;
class FractionalKnapSack {
   private static double getMaxValue(int[] wt, int[] val, int capacity) {
       ItemValue[] iVal = new ItemValue[wt.length];
       for (int i = 0; i < wt.length; i++) {
           iVal[i] = new ItemValue(wt[i], val[i], i);
       }
       Arrays.sort(iVal, new Comparator<ItemValue>() {
           @Override
           public int compare(ItemValue o1, ItemValue o2)
           {
               return o2.cost.compareTo(o1.cost);
           }
       });
       double totalValue = 0d;
       for (ItemValue i : iVal) {
           int curWt = (int)i.wt;
           int curVal = (int)i.val;
           if (capacity - curWt >= 0) {
               capacity = capacity - curWt;
               totalValue += curVal;
           }
           else {
               double fraction
                       = ((double)capacity / (double)curWt);
               totalValue += (curVal * fraction);
               capacity
                       = (int)(capacity - (curWt * fraction));
               break;
           }
       }
       return totalValue;
   }
   static class ItemValue {
       Double cost;
       double wt, val, ind;
       public ItemValue(int wt, int val, int ind)
       {
           this.wt = wt;
           this.val = val;
           this.ind = ind;
           cost = new Double((double)val / (double)wt);
       }
   }
   public static void main(String[] args) {
       int[] wt = { 2, 4, 5, 2, 5 };
```

```
        int[] val = { 5, 10, 15, 13, 20 };
        int capacity = 10;
        double maxValue = getMaxValue(wt, val, capacity);
        System.out.println("Maximum value we can obtain = " + maxValue);
    }
}
```

Output:-



```
        public ItemValue(int wt, int val, int ind)
        {
            this.wt = wt;
            this.val = val;
            this.ind = ind;
            cost = new ~~Double~~( value: (double)val / (double)wt);
        }
    }
    public static void main(String[] args) {
        int[] wt = { 2, 4, 5, 2, 5 };
        int[] val = { 5, 10, 15, 13, 20 };
        int capacity = 10;
        double maxValue = getMaxValue(wt, val, capacity);
        System.out.println("Maximum value we can obtain = " + maxValue);
    }
}
```

```
C:\Users\lenovo\.jdks\openjdk-15.0.2\bin\java.exe ...
Maximum value we can obtain = 42.0

Process finished with exit code 0
```

Analysis:-

the whole problem can be solved in O(n log n) only.