

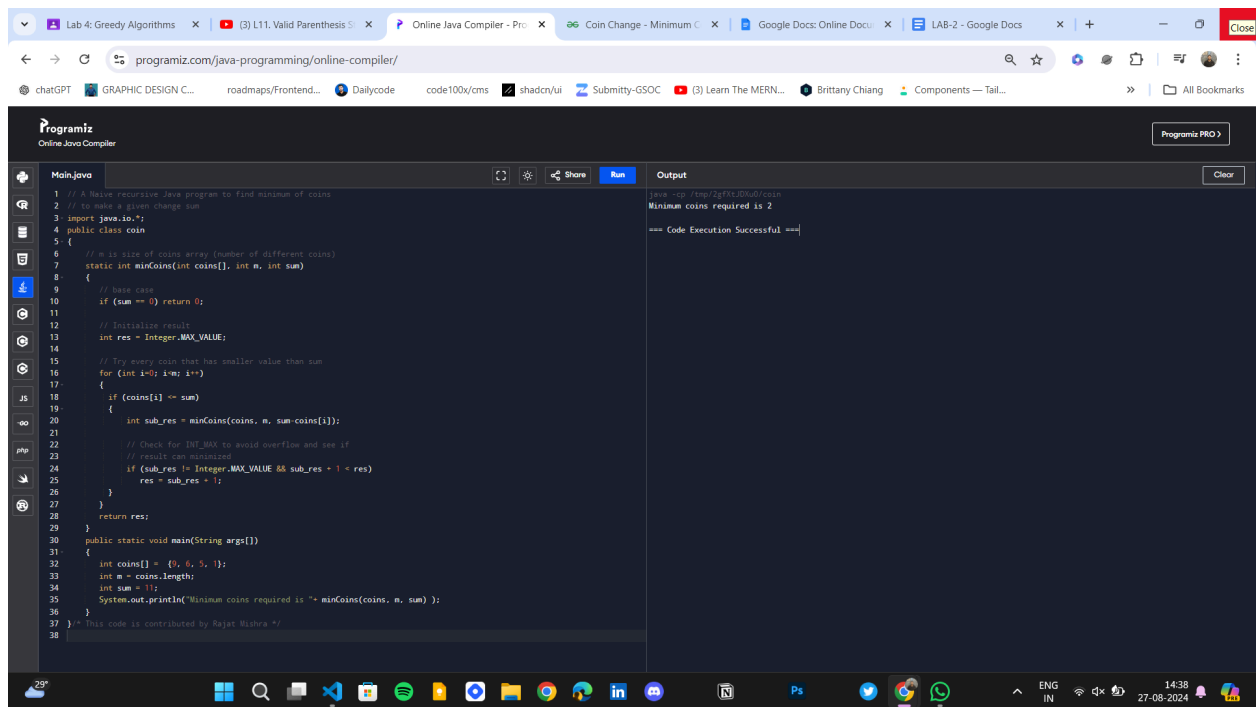
ADVANCED ALGORITHMS

Assignment

NAME : G.Sai Bharath Chandra Reddy

Roll.No : 22cs2013

1. Coin Change Problem (Minimum Coins)



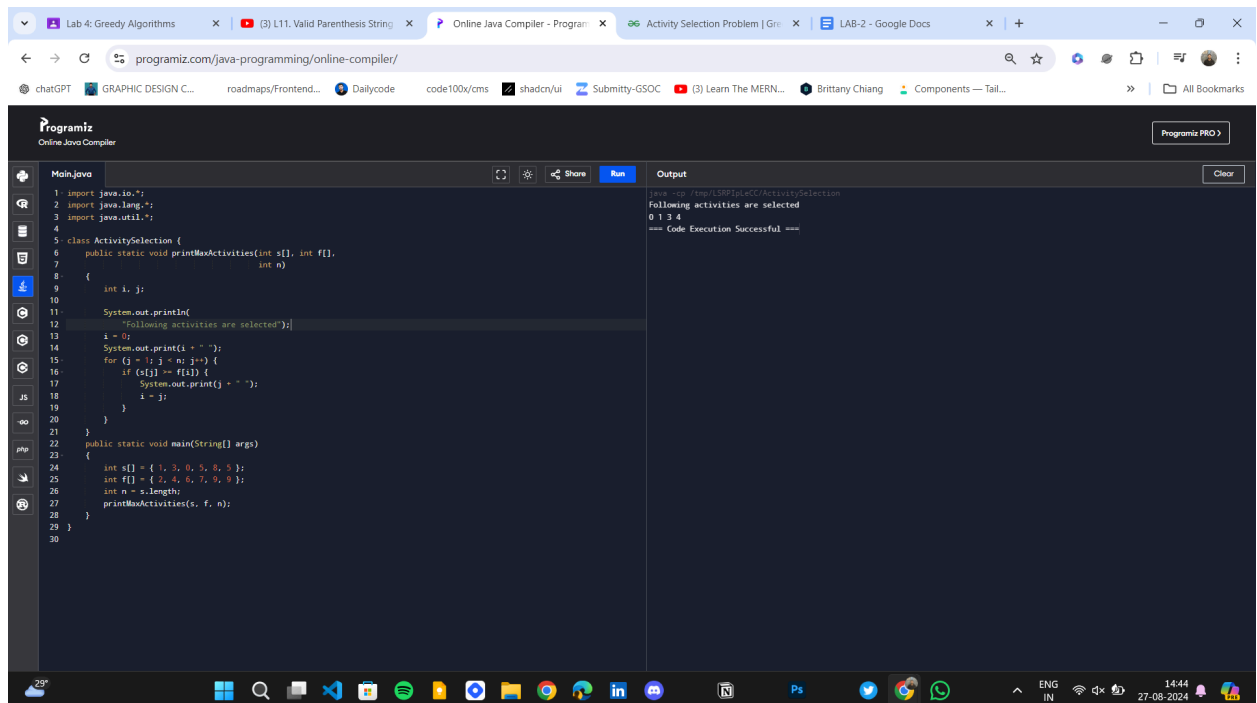
The screenshot shows a web browser with multiple tabs, including 'Lab 4: Greedy Algorithms', 'L11: Valid Parenthesis String', 'Online Java Compiler - Programiz', 'Coin Change - Minimum Coins', 'Google Docs: Online Document', and 'LAB-2 - Google Docs'. The active tab is 'Online Java Compiler - Programiz'. The browser address bar shows 'programiz.com/java-programming/online-compiler/'. The Programiz Online Java Compiler interface is displayed, showing a Java program for the Coin Change Problem. The code is as follows:

```
1 // A naive recursive Java program to find minimum of coins
2 // to make a given change sum
3 import java.io.*;
4 public class coin
5 {
6     // m is size of coins array (number of different coins)
7     static int minCoins(int coins[], int m, int sum)
8     {
9         // base case
10        if (sum == 0) return 0;
11
12        // Initialize result
13        int res = Integer.MAX_VALUE;
14
15        // Try every coin that has smaller value than sum
16        for (int i=0; i<m; i++)
17        {
18            if (coins[i] <= sum)
19            {
20                int sub_res = minCoins(coins, m, sum-coins[i]);
21
22                // Check for INT_MAX to avoid overflow and see if
23                // result can minimized
24                if (sub_res != Integer.MAX_VALUE && sub_res + 1 < res)
25                    res = sub_res + 1;
26            }
27        }
28        return res;
29    }
30    public static void main(String args[])
31    {
32        int coins[] = {9, 6, 5, 1};
33        int m = coins.length;
34        int sum = 11;
35        System.out.println("Minimum coins required is " + minCoins(coins, m, sum) );
36    }
37 } // This code is contributed by Rajat Mishra */
38
```

The output of the program is:

```
java -cp /tmp/1g7f1cJ0u6/coin
Minimum coins required is 2
=== Code Execution Successful ===
```

2. Activity Selection Problem



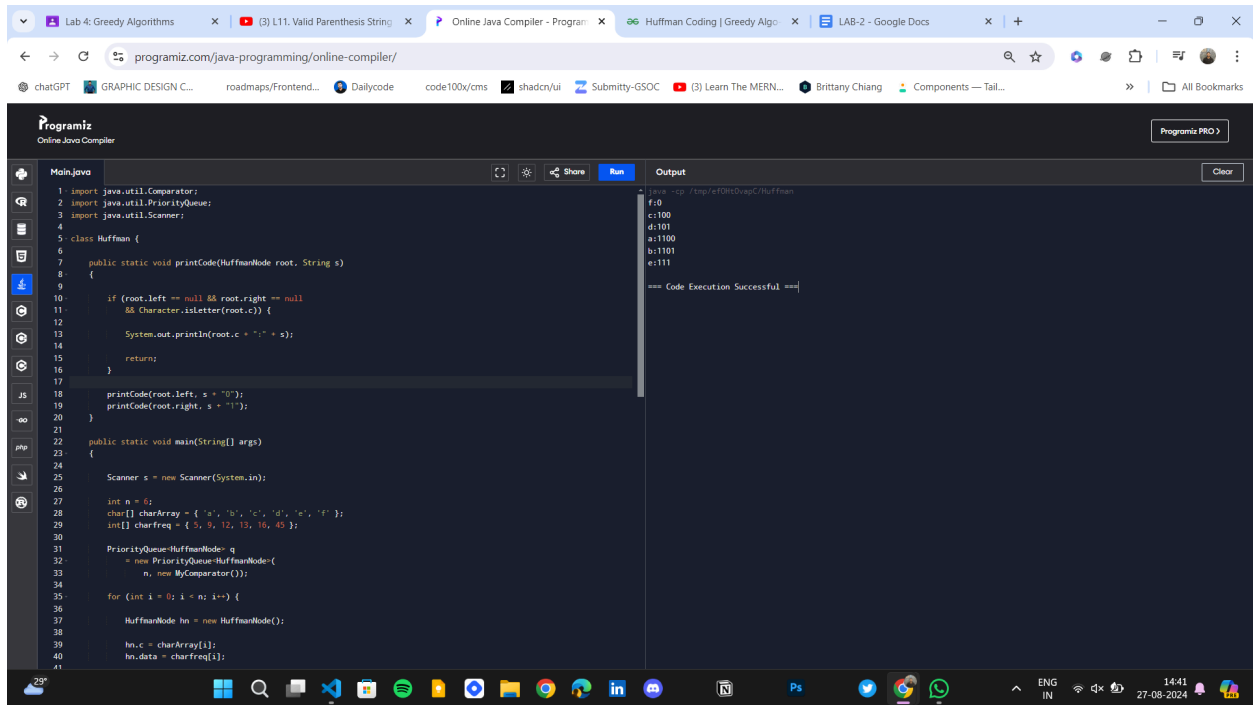
The screenshot shows a web browser with multiple tabs, including 'Lab 4: Greedy Algorithms', 'L11: Valid Parenthesis String', 'Online Java Compiler - Programiz', 'Activity Selection Problem | Greedy Algorithms', and 'LAB-2 - Google Docs'. The active tab is 'Online Java Compiler - Programiz'. The browser address bar shows 'programiz.com/java-programming/online-compiler/'. The Programiz Online Java Compiler interface is displayed, showing a Java program for the Activity Selection Problem. The code is as follows:

```
1 import java.io.*;
2 import java.lang.*;
3 import java.util.*;
4
5 class ActivitySelection {
6     public static void printMaxActivities(int s[], int f[],
7     int n)
8     {
9         int i, j;
10
11        System.out.println(
12            "Following activities are selected");
13        i = 0;
14        System.out.print(i + " ");
15        for (j = 1; j < n; j++) {
16            if (s[j] >= f[i]) {
17                System.out.print(j + " ");
18                i = j;
19            }
20        }
21    }
22    public static void main(String[] args)
23    {
24        int s[] = { 1, 3, 6, 5, 8, 5 };
25        int f[] = { 2, 4, 6, 7, 9, 9 };
26        int n = s.length;
27        printMaxActivities(s, f, n);
28    }
29 }
30
```

The output of the program is:

```
java -cp /tmp/1G0Pp1u4CC/ActivitySelection
Following activities are selected
0 1 3 4
=== Code Execution Successful ===
```

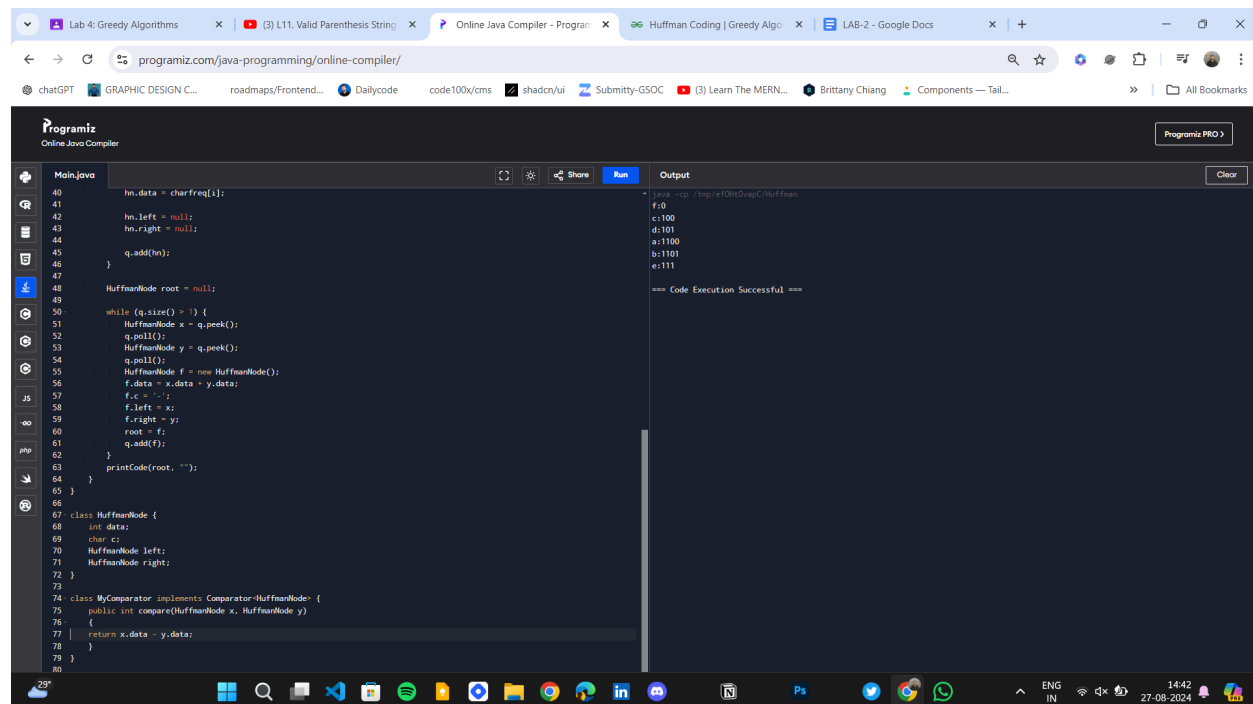
3. Huffman Coding



```
1 import java.util.Comparator;
2 import java.util.PriorityQueue;
3 import java.util.Scanner;
4
5 class Huffman {
6
7     public static void printCode(HuffmanNode root, String s)
8     {
9
10        if (root.left == null && root.right == null
11            && Character.isLetter(root.c)) {
12
13            System.out.println(root.c + ":" + s);
14
15            return;
16        }
17
18        printCode(root.left, s + "0");
19        printCode(root.right, s + "1");
20    }
21
22    public static void main(String[] args)
23    {
24
25        Scanner s = new Scanner(System.in);
26
27        int n = 6;
28        char[] charArray = { 'a', 'b', 'c', 'd', 'e', 'f' };
29        int[] charfreq = { 5, 9, 12, 13, 16, 45 };
30
31        PriorityQueue<HuffmanNode> q
32            = new PriorityQueue<HuffmanNode>(
33                n, new MyComparator());
34
35        for (int i = 0; i < n; i++) {
36
37            HuffmanNode hn = new HuffmanNode();
38
39            hn.c = charArray[i];
40            hn.data = charfreq[i];
```

```
f:0
c:100
d:101
a:1100
b:1101
e:111

=== Code Execution Successful ===
```

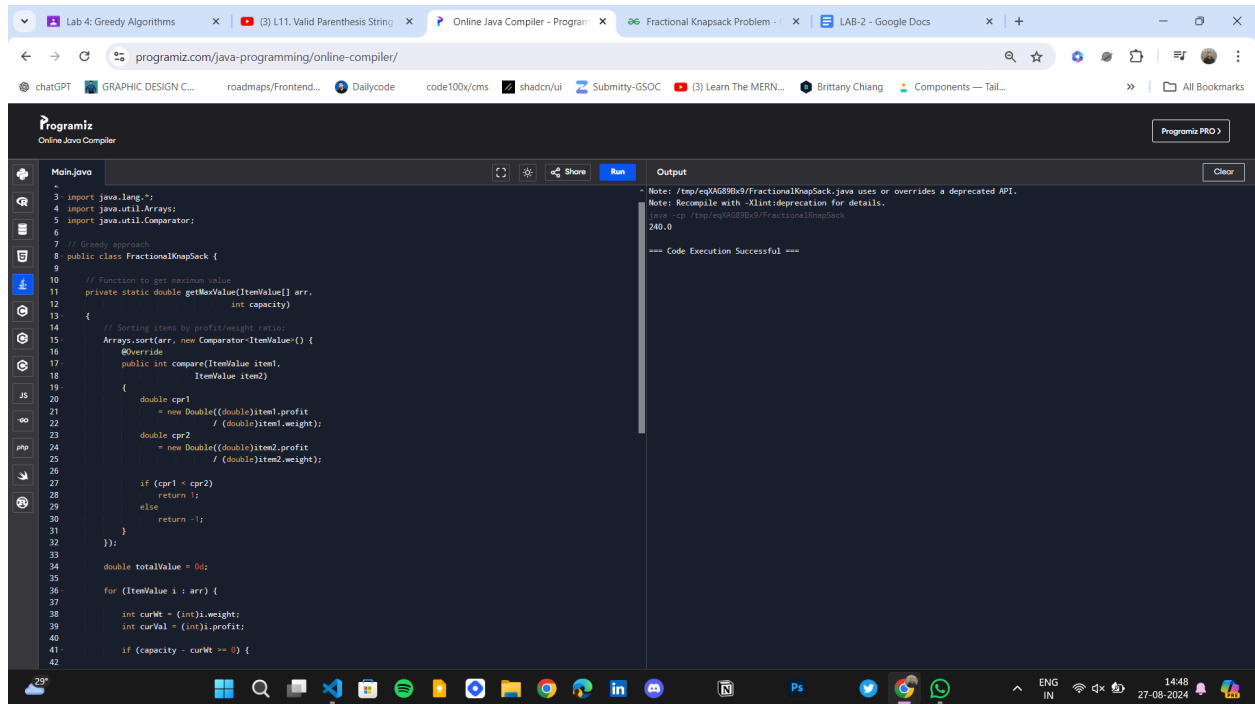


```
40 hn.data = charfreq[i];
41
42 hn.left = null;
43 hn.right = null;
44
45 q.add(hn);
46 }
47
48 HuffmanNode root = null;
49
50 while (q.size() > 1) {
51     HuffmanNode x = q.peak();
52     q.poll();
53     HuffmanNode y = q.peak();
54     q.poll();
55     HuffmanNode f = new HuffmanNode();
56     f.data = x.data + y.data;
57     f.c = '-';
58     f.left = x;
59     f.right = y;
60     root = f;
61     q.add(f);
62 }
63
64 printCode(root, "");
65 }
66
67 class HuffmanNode {
68     int data;
69     char c;
70     HuffmanNode left;
71     HuffmanNode right;
72 }
73
74 class MyComparator implements Comparator<HuffmanNode> {
75     public int compare(HuffmanNode x, HuffmanNode y)
76     {
77         return x.data - y.data;
78     }
79 }
80 }
```

```
f:0
c:100
d:101
a:1100
b:1101
e:111

=== Code Execution Successful ===
```

4. Fractional Knapsack Problem

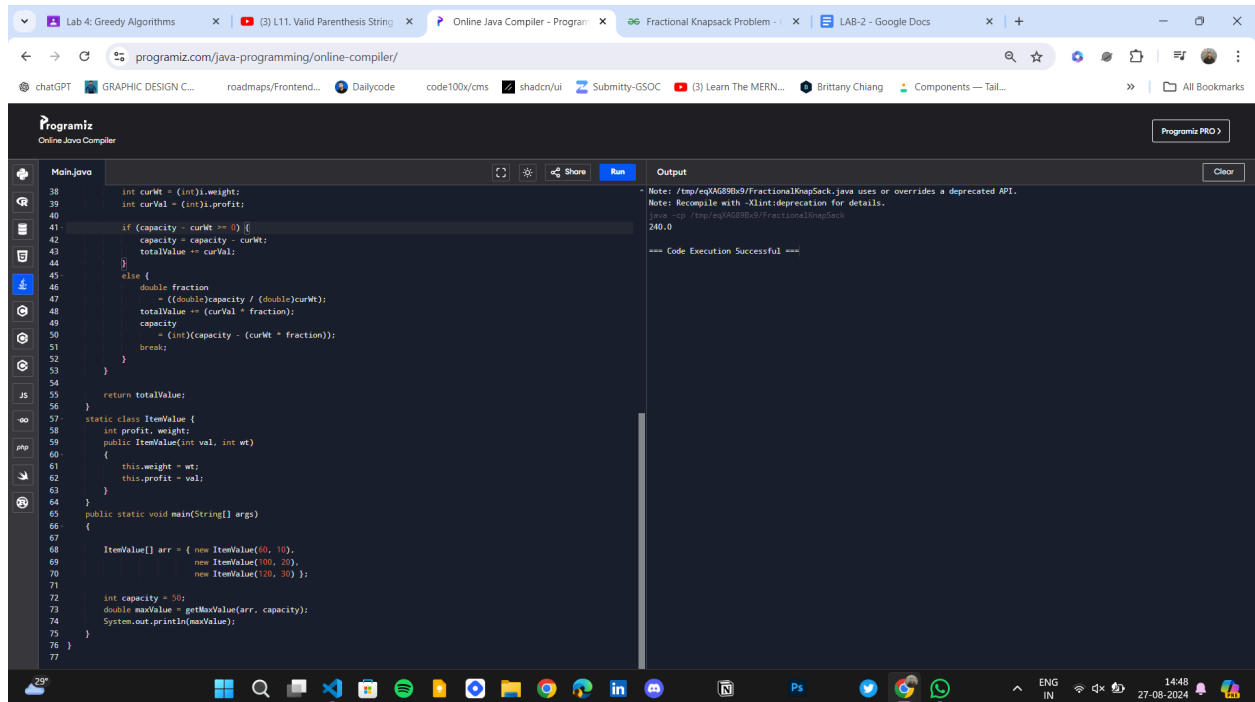


```
3 import java.lang.*;
4 import java.util.Arrays;
5 import java.util.Comparator;
6
7 // Greedy approach
8 public class FractionalKnapsack {
9
10     // Function to get maximum value
11     private static double getMaxValue(ItemValue[] arr,
12                                     int capacity)
13     {
14         // Sorting items by profit/weight ratio:
15         Arrays.sort(arr, new Comparator<ItemValue>() {
16             @Override
17             public int compare(ItemValue item1,
18                               ItemValue item2)
19             {
20                 double cpr1
21                     = new Double(((double)item1.profit
22                                   / (double)item1.weight));
23                 double cpr2
24                     = new Double(((double)item2.profit
25                                   / (double)item2.weight));
26
27                 if (cpr1 < cpr2)
28                     return 1;
29                 else
30                     return -1;
31             }
32         });
33
34         double totalValue = 0;
35
36         for (ItemValue i : arr) {
37             int curWt = (int)i.weight;
38             int curVal = (int)i.profit;
39
40             if (capacity - curWt == 0) {
41
42 
```

Note: /tmp/eqAG8Bb9/FractionalKnapsack.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

```
java -cp /tmp/eqAG8Bb9/FractionalKnapsack
240.0

=== Code Execution Successful ===
```



```
41
42         capacity = capacity - curWt;
43         totalValue += curVal;
44     }
45     else {
46         double fraction
47             = ((double)capacity / (double)curWt);
48         totalValue += (curVal * fraction);
49         capacity
50             = (int)(capacity - (curWt * fraction));
51         break;
52     }
53 }
54
55 return totalValue;
56 }
57
58 static class ItemValue {
59     int profit, weight;
60     public ItemValue(int val, int wt)
61     {
62         this.weight = wt;
63         this.profit = val;
64     }
65 }
66
67 public static void main(String[] args)
68 {
69     ItemValue[] arr = { new ItemValue(60, 10),
70                       new ItemValue(100, 20),
71                       new ItemValue(120, 30) };
72
73     int capacity = 50;
74     double maxVal = getMaxValue(arr, capacity);
75     System.out.println(maxVal);
76 }
77 }
```

Note: /tmp/eqAG8Bb9/FractionalKnapsack.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

```
java -cp /tmp/eqAG8Bb9/FractionalKnapsack
240.0

=== Code Execution Successful ===
```

5. Job Sequencing Problem

The screenshot shows the Programiz Online Java Compiler interface. The code in `Main.java` defines a `Job` class with attributes `id`, `deadline`, and `profit`. It includes a `printJobScheduling` method that sorts jobs by profit and uses a greedy algorithm to select jobs that can be completed before their deadlines. The `main` method initializes an array of jobs: `a` (profit 2, deadline 2), `b` (profit 1, deadline 1), and `c` (profit 2, deadline 2).

```
1 import java.util.*;
2
3 class Job {
4     char id;
5     int deadline, profit;
6     public Job() {}
7
8     public Job(char id, int deadline, int profit)
9     {
10         this.id = id;
11         this.deadline = deadline;
12         this.profit = profit;
13     }
14
15     void printJobScheduling(ArrayList<Job> arr, int t)
16     {
17         int n = arr.size();
18         Collections.sort(arr,
19             (a, b) -> b.profit - a.profit);
20         boolean result[] = new boolean[t];
21         char job[] = new char[t];
22         for (int i = 0; i < n; i++) {
23             for (int j = 0; j < t; j++) {
24                 if (Math.min(n - i, arr.get(i).deadline - j) > 0) {
25                     result[j] = true;
26                     job[j] = arr.get(i).id;
27                     break;
28                 }
29             }
30         }
31         for (char job : job)
32             System.out.print(job + " ");
33         System.out.println();
34     }
35
36     public static void main(String args[])
37     {
38         ArrayList<Job> arr = new ArrayList<Job>();
39         arr.add(new Job('a', 2, 100));
40         arr.add(new Job('b', 1, 10));
41         arr.add(new Job('c', 2, 20));
42     }
43 }
```

The output shows the maximum profit sequence of jobs: `a b c`.

The screenshot shows the same Programiz Online Java Compiler interface, but with the completed code. The `main` method now includes all the jobs: `a` (profit 2, deadline 2), `b` (profit 1, deadline 1), `c` (profit 2, deadline 2), `d` (profit 1, deadline 1), `e` (profit 1, deadline 1), and `f` (profit 1, deadline 1). The output shows the maximum profit sequence of jobs: `a b c`.

```
12     this.profit = profit;
13 }
14 void printJobScheduling(ArrayList<Job> arr, int t)
15 {
16     int n = arr.size();
17     Collections.sort(arr,
18         (a, b) -> b.profit - a.profit);
19     boolean result[] = new boolean[t];
20     char job[] = new char[t];
21     for (int i = 0; i < n; i++) {
22         for (int j = 0; j < t; j++) {
23             if (Math.min(n - i, arr.get(i).deadline - j) > 0) {
24                 result[j] = true;
25                 job[j] = arr.get(i).id;
26                 break;
27             }
28         }
29     }
30     for (char job : job)
31         System.out.print(job + " ");
32     System.out.println();
33 }
34
35 public static void main(String args[])
36 {
37     ArrayList<Job> arr = new ArrayList<Job>();
38     arr.add(new Job('a', 2, 100));
39     arr.add(new Job('b', 1, 10));
40     arr.add(new Job('c', 2, 20));
41     arr.add(new Job('d', 1, 20));
42     arr.add(new Job('e', 1, 10));
43     arr.add(new Job('f', 1, 10));
44
45     System.out.println(
46         "Following is maximum profit sequence of jobs");
47
48     Job job = new Job();
49     job.printJobScheduling(arr, 3);
50 }
51 }
```

The output shows the maximum profit sequence of jobs: `a b c`.