W1D4 solutions

Question 1:

```
package work.D3.prob1a;

import java.util.*;

public class Subsetsum {
    public static boolean[][] subsetSumDP(int[] set, int sum) {
        int n = set.length;
        boolean[][] dp = new boolean[n + 1][sum + 1];

        for (int i = 0; i <= n; i++)
            dp[i][0] = true;

        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= sum; j++) {
                if (set[i - 1] <= j)
                    dp[i][j] = dp[i - 1][j] || dp[i - 1][j - set[i - 1]];
                else
                    dp[i][j] = dp[i - 1][j];
            }
        }

        return dp;
    }


    public static void findSubsets(int[] set, int n, int sum,
                                   List<Integer> current, boolean[][] dp,
                                   List<List<Integer>> allSolutions) {

        if (sum == 0) {
            allSolutions.add(new ArrayList<>(current));
            return;
        }
        if (n == 0)
            return;


        if (dp[n - 1][sum])
            findSubsets(set, n - 1, sum, current, dp, allSolutions);

                if (sum >= set[n - 1] && dp[n - 1][sum - set[n - 1]]) {
            current.add(set[n - 1]);
            findSubsets(set, n - 1, sum - set[n - 1], current, dp, allSolutions);
            current.remove(current.size() - 1);
        }
    }

    public static void main(String[] args) {
        int[] S = {3, 4, 7, 8};
        int K = 15;

        boolean[][] dp = subsetSumDP(S, K);
        int n = S.length;


        System.out.println("(a) Subset with sum " + K + " exists: " + (dp[n][K] ?
"True" : "False"));

        if (!dp[n][K]) {
            System.out.println("No subsets found.");
            return;
        }
```

```
        List<List<Integer>> allSolutions = new ArrayList<>();
        findSubsets(S, n, K, new ArrayList<>(), dp, allSolutions);

        System.out.println("(b) One solution: " + allSolutions.get(0));

        System.out.print("(c) All solutions: ");
        for (List<Integer> subset : allSolutions)
            System.out.print(subset+", ");
    }
}
```

Output:

(a) Subset with sum 15 exists: True

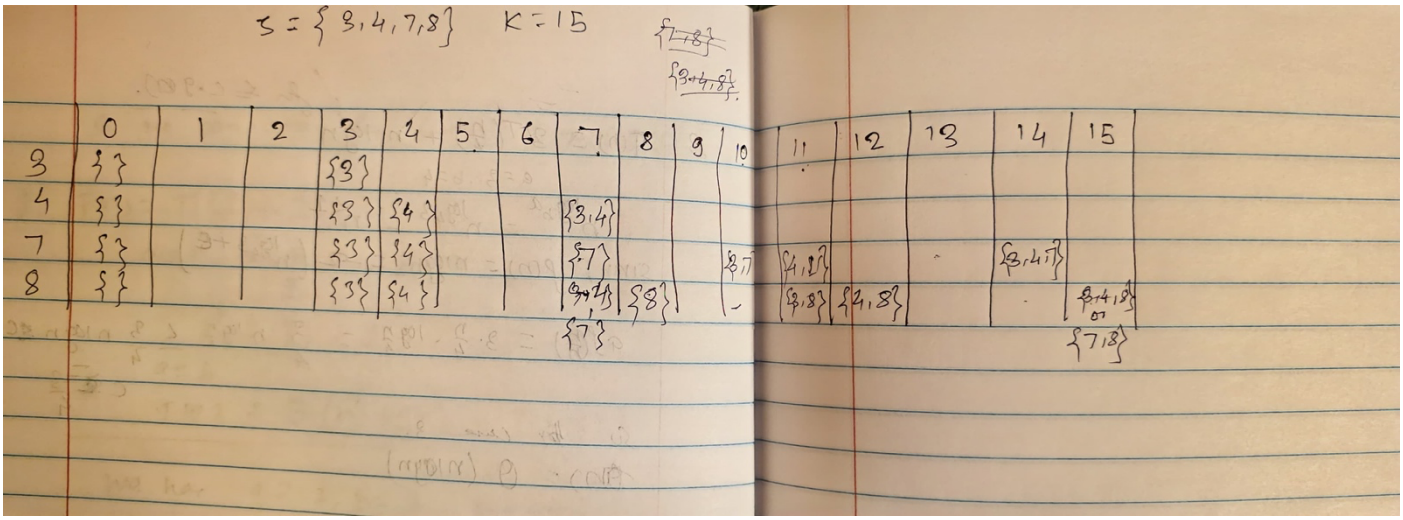
(b) One solution: [8, 4, 3]

(c) All solutions: [8, 4, 3], [8, 7],

Question 2:

a

Element	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
–	T	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
3	T	F	F	T	F	F	F	F	F	F	F	F	F	F	F	F
4	T	F	F	T	T	F	F	T	F	F	F	F	F	F	F	F
7	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	F
8	T	F	F	T	T	F	F	T	T	F	F	T	T	F	T	**T**



b. One solution : {3,4,8}

c. All solutions : {3,4,8} and {7,8}

Question 3. 0/1 knapsack

Weight:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
d,2,16	0	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16
a,5,25	0	16	16	16	25	25	41	41	41	41	41	41	41	41	41	41	41	41	41	41
b,6,12	0	16	16	16	25	25	41	41	41	41	41	41	53	53	53	53	53	53	53	53
e,7,28	0	16	16	16	25	25	41	41	44	44	44	53	53	69	69	69	69	69	69	81
c,8,24	0	16	16	16	25	25	41	41	44	44	44	53	53	69	69	69	69	69	69	81

Maximum value = 81, Items included ={d,a,b,e}

Question 4 : Fractional knapsack

The maximum allowable total weight in the knapsack is Wmax = 20.

Item	a	b	c	d	e
value	**25**	**12**	**24**	**16**	**28**
Weight	**5**	**6**	**8**	**2**	**7**

Now value:weight :- a:5, b:2, c:3, d:8, e:4

items	d	a	e	¾ of c
Wmax - weight	20 – 2 = 18	18 – 5 = 13	13 – 7 = 6	6 – 6 (¾ *8)
value	16	25	28	18

Items selected : d,a,e, and ¾ c

Total weight used : 2+5+7+6 = 20

Maximum value : 16+25+28+18(¾ of 24) = 87

Question 5:

```
class Solution {
    public int climbStairs(int n) {
        if (n<=0) return 0;
        if(n==1) return 1;
        if(n==2) return 2;
        int[] arr = new int[n+1];
        arr[1] =1;
        arr[2] = 2;
        for(int i=3;i<=n;i++){
            arr[i]=arr[i-1]+arr[i-2];
        }
        return arr[n];
    }
}
```

Question 6:

```java
class Solution {
    public int rob(int[] nums) {
        int len = nums.length;
        if(len==1) return nums[0];
        int[] loot= new int[len];
        loot[0] = nums[0];
        loot[1]=Math.max(nums[0],nums[1]);
        for(int i=2;i<len;i++){
            loot[i]=Math.max(loot[i-1],loot[i-2]+nums[i]);
        }
        return loot[len-1];
  }
}
```