W1D4

QUESTION 1.

Write a Java program to solve the subset problem.

(a) T or F.

```java
public class Subset {
    public static boolean[][] subset(int[] s, int n) {
        int rows = s.length;
        // Initialize 2D boolean array with default value false (replaces '-')
        boolean[][] res = new boolean[rows][n + 1];

        // Fill the array based on the subset logic
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j <= n; j++) {
                if (i == 0) {
                    // First row: true if j is 0 or j equals s[0]
                    if (j == 0 || j == s[i]) {
                        res[i][j] = true;
                    }
                } else {
                    // If j < s[i], copy the value from the previous row
                    if (j < s[i]) {
                        res[i][j] = res[i - 1][j];
                    } else {
                        // Check if subset exists with or without s[i]
                        res[i][j] = res[i - 1][j - s[i]];
                    }
                }
            }
        }
        return res;
    }
}
```

(b) One solution.

```java
public class Subset {
    public static String[][] subsetOnSolution(int[] s, int n) {
        int rows = s.length;
        // Initialize 2D array with spaces
        String[][] res = new String[rows][n + 1];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j <= n; j++) {
                res[i][j] = " "; // Default value
            }
        }

        // Fill the array with subset logic
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j <= n; j++) {
                if (j == 0) {
                    res[i][j] = "{}";
                    continue;
                }

                if (i == 0) {
                    if (j == s[i]) {
                        res[i][j] = "{" + s[i] + "}";
                    }
                } else {
                    if (j < s[i]) {
                        if (res[i][j].equals(" ")) {
                            res[i][j] = "" + res[i - 1][j] + "";
                        } else {
                            res[i][j] = "" + res[i - 1][j] + "";
                        }
                    } else {
                        if (res[i - 1][j - s[i]].equals(" ")) {
                            res[i][j] = " ";
                        } else if (!res[i - 1][j - s[i]].equals(" ")) {
                            if (res[i - 1][j - s[i]].length() == 2) {
                                if (j == s[i] && res[i - 1][j].length() > 2) {
                                    res[i][j] = "" + res[i - 1][j];
                                } else {
                                    res[i][j] = res[i - 1][j - s[i]].replace("}", "") + s[i] +
"}";
                                }
                            } else {
                                res[i][j] = res[i - 1][j - s[i]].replace("}", ",") + s[i] + "}";
                            }
                        }
                    }
                }
            }
        }

        return res; // Optional: return the array if needed
    }
}
```

(c) All solutions.

```java
public class Subset {
    public static String[][] subsetAllSolution(int[] s, int n) {
        int rows = s.length;
        // Initialize 2D array with spaces
        String[][] res = new String[rows][n + 1];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j <= n; j++) {
                res[i][j] = " "; // Default value
            }
        }

        // Fill the array with subset logic
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j <= n; j++) {
                if (j == 0) {
                    res[i][j] = "{}";
                    continue;
                }

                if (i == 0) {
                    if (j == s[i]) {
                        res[i][j] = "{" + s[i] + "}";
                    }
                } else {
                    if (j < s[i]) {
                        if (res[i][j].equals(" ")) {
                            res[i][j] = "" + res[i - 1][j] + "";
                        } else {
                            res[i][j] = "" + res[i - 1][j] + "";
                        }
                    } else {
                        String cur = res[i - 1][j - s[i]];
                        if (cur.equals(" ")) {
                            res[i][j] = " ";
                        } else {
                            if (cur.length() == 2) {
                                if (j == s[i] && res[i - 1][j].length() > 2) {
                                    res[i][j] = "" + res[i - 1][j] + ",{" + s[i] + "}";
                                } else {
                                    res[i][j] = cur.replace("}", "") + s[i] + "}";
                                }
                            } else {
                                if (cur.indexOf(",{") != -1) {
                                    int startIndex = cur.indexOf(",{") - 1;
                                    res[i][j] = cur.substring(0, startIndex + 1) + "," + s[i] +
"}";                                } else {
                                    res[i][j] = cur.replace("}", ",") + s[i] + "}";
                                }
                            }
                        }
                    }
                }
            }
        }

        return res; // Optional: return the array if needed
    }

}
```

QUESTION 2.

Solve subset problem where S = {3, 4, 7, 8} and k = 15.

(a) T or F.

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

(b) One solution.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	{}			{3}												
4	{}			{3}	{4}			{3,4}								
7	{}			{3}	{4}			{3,4}			{3,7}	{4,7}			{3,4,7}	
8	{}			{3}	{4}			{3,4}	{8}			{4,7}	{4,8}			{3,4,8)

(c) All solutions.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	{}			{3}												
4	{}			{3}	{4}			{3,4}								
7	{}			{3}	{4}			{3,4},{7}			{3,7}	{4,7}			{3,4,7}	
8	{}			{3}	{4}			{3,4},{7}	{8}			{4,7}	{4,8}			{3,4,8)

QUESTION 3

Solve the integer Knapsack problem given below:

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

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
a	0	0	0	0	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25
b	0	0	0	0	25	25	25	25	25	25	37	37	37	37	37	37	37	37	37	37
c	0	0	0	0	25	25	25	25	25	25	37	37	49	49	49	49	49	49	61	61
d	0	0	16	16	25	25	41	41	41	41	41	41	53	53	65	65	65	65	65	65
e	0	0	16	16	25	25	41	41	41	44	44	53	53	69	69	69	69	69	69	81

QUESTION 4. Solve the fractional Knapsack problem given below:

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

Total weight: 20

Max value approach: e + a + c = 28 + 25 + 24 = 77

Min weight approach: d + a + b + e = 2 + 5 + 6 + 7 = 20

Value per weight: a: 5, b: 2, c: 3, d: 8, e: 4

Select: d a e 0.75c = 16 + 25 + 28 + 18 = 87

QUESTION 5.

https://leetcode.com/problems/climbing-stairs/description/

```python
class Solution:
    def climbStairs(self, n: int) ->
int:    if n < 3:
            return n
        dp = [0] * (n + 1)
        dp[0] = 1
        dp[1] = 1
        for i in range(2, n + 1):
            dp[i] = dp[i - 1] + dp[i -
2]

        return dp[n]
```

QUESTION 6.

https://leetcode.com/problems/house-robber/description/

```python
def rob(self, nums: List[int]) -> int:
        n = len(nums)
        if n == 0:
            return 0
        if n <= 1:
            return nums[0]

        dp = [0] * n
        dp[0] = nums[0]
        dp[1] = max(nums[0], nums[1])
        for i in range(2, n):
            dp[i] = max(dp[i - 2] + nums[i], dp[i -
1])
        return dp[n - 1]
```