Yuliang jin 986381

Problem 1

Knapsack problem S = {s0, s1, s2}, V = {100, 15, 10, 10}, w = {50, 11, 10, 10}, W = 70.

Problem 2

Below, the BinarySearch and Recursive Fibonacci algorithms are shown. In each case, what are the subproblems? Why do we say that the subproblems of BinarySearch do not overlap and the subproblems of Recursive Fibonacci overlap? Explain. The self calls invoked are the sub problems. The input of binary search will always be half of the original input, but the input of the fibonacci will always be the current input n minus one (n -1).

Problem 3

Consider the following SubsetSum problem: S = {4, 2, 5, 3}, k = 5. Fill in the first row of the table for the bottom-up dynamic programming solution for this problem. (Locate the formula for this in the slides.)

Problem 4

Consider the following SubsetSum problem: S = {4, 3, 5, 6}, k = 8. Part of the table A[i,j] for the bottom-up dynamic programming solution is provided. Use the recursive formula given in the slides to compute the values of A[1,7] and A[2,7]. A[1,7] = {4, 3} A[2,7] = {4, 3}

Problem 5

(Optional) Consider the following Knapsack problem: S = {s0, s1, s2, s3}, w[] = {3, 1, 3, 5}, v[] = {4, 2, 3, 2}, W = 7. Part of the table A[i,j] for the bottom-up dynamic programming solution is provided. Use the recursive formula given in the slides to compute the values of A[2,7] and A[3,7]. A[2,7] = {s0, s1, s2} A[3,7] = {s0, s1, s2}

Problem 6

Use the Knapsack problem you created in Problem 1 as a starting point, but now find an optimal solution for the fractional knapsack problem based on the same input data. Use fractional knapsack, we can split each into unit of weight.

Problem 7

(Interview Question) Devise a dynamic programming solution for the following problem: Given two strings, find the length of longest subsequence that they share in common. Different between substring and subsequence: Substring: the characters in a substring of S must occur contiguously in S. Subsequence: the characters can be interspersed with gaps. For example: Given two Strings - “regular” and “ruler”, your algorithm should output 4.

Solution: The length of the two inputs is m and n. Create array, a[m+1][n+1], for each value a[i][j] denotes the longest common string between the substring of the first i character in the first string and that of the first j characters in the second string. So a[m][n] is the final result. Use the buttom up interative way to compute each value in the array. Compute a[i][j], if(s1[i] == s2[j]), then a[i][j] = a[i-1][j-1] + 1; if(s1[i] != s2[j]), a[i][j] = Math.max(a[i-1][j], a[i][j-1]) .

Problem 8

Given a positive integer n, find the least number of perfect square numbers which sum to n. (Perfect square numbers are 1, 4, 9, 16, 25, 36, 49, ...) For example, given n = 12, return 3; (12 = 4 + 4 + 4) Given n = 13, return 2; (13 = 4 + 9) Given n = 67 return 3; (67 = 49 + 9 + 9)

// A dynamic programming based JAVA program to find minimum // number of squares whose sum is equal to a given number class squares { // Returns count of minimum squares that sum to n static int getMinSquares(int n) { // Create a dynamic programming table // to store sq int dp[] = new int[n+1];

```
        // getMinSquares table for base case entries
        dp[0] = 0;
        dp[1] = 1;
        dp[2] = 2;
        dp[3] = 3;

        // getMinSquares rest of the table using recursive
        // formula
        for (int i = 4; i <= n; i++)
        {
            // max value is i as i can always be represented
            // as 1*1 + 1*1 + ...
            dp[i] = i;
```

```
            // Go through all smaller numbers to
            // to recursively find minimum
            for (int x = 1; x <= i; x++) {
                int temp = x*x;
                if (temp > i)
                    break;
                else dp[i] = Math.min(dp[i], 1+dp[i-temp]);
            }
        }

        // Store result and free dp[]
        int res = dp[n];

        return res;
    }
    public static void main(String args[])
    {
       System.out.println(getMinSquares(6));
    }
```
}/* This code is contributed by Rajat Mishra */