Lab 10

Day 1: do problems 2, 3, 4, (7, 8)

Day 2: finish the rest of the problems

1. In the slides, Greedy Strategy #2 for solving the Knapsack Problem was the following:

   Greedy Strategy #2. Try arranging S in decreasing order of *value per weight*. For each i, let $b_i = v_i/w_i$. Scan the new arrangement S′ of S and put in items as long as the weight restriction permits; skip over items that will cause the weight to exceed W.

   Give an example of a Knapsack problem for which this strategy does *not* give an optimal solution.

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.

```
Algorithm binSearch(A, x, lower, upper)
    Input: Already sorted array A of size n, value x to be
           searched for in array section A[lower]..A[upper]
    Output: true or false

  if lower > upper then return false
  mid ← (upper + lower)/2
  if x = A[mid] then return true
  if x < A[mid] then
      return binSearch(A, x, lower, mid – 1)
  else
      return binSearch(A, x, mid + 1, upper)
```

```
Algorithm fib(n)
    Input: a natural number n
    Output: F(n)

  if (n = 0 || n = 1) then return n
  return fib(n-1) + fib(n-2)
```

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.)

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[i,j]	0	1	2	3	4	5	6	7	8
0	{}	NULL	NULL	NULL	{4}	NULL	NULL	NULL	NULL
1	{}	NULL	NULL	{3}	{4}	NULL	NULL	**??**	
2								**??**	
3									

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[i,j]	0	1	2	3	4	5	6	7
0	{}	{}	{}	{s0}	{s0}	{s0}	{s0}	{s0}
1	{}	{s1}	{s1}	{s0}	{s0, s1}	{s0, s1}	{s0,s1}	{s0,s1}
2	{}	{s1}	{s1}	{s0}	{s0, s1}	{s0, s1}	{s0,s2}	??
3								??

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.

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.

8. *(Optional Interview Question)* Devise a dynamic programming solution for the following problem:

   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)