1) Given an unlimited supply of coins of denominations x_1, x_2, ..., x_n, we wish to make change for a value v using at most k coins; that is, we wish to find a set of k coins whose total value is v. This might not be possible: for instance, if the denominations are 5 and 10 and k = 6, then we can make change for 55 but not for 65. Give an efficient algorithm for the following problem.

Input: $x_1, x_2, ..., x_n, k, v$.
Question: Is it possible to make change for v using at most k coins, of denominations $x_1, x_2, ..., x_n$? (No proof of correctness required)

Solution:

1) OPT[v] will denote the minimum number of coins required to make change for value "v". The recursive formula would be:

$$OPT[T] = \min_{1 \le i \le n} \{OPT[T - X_i] + 1\}$$

The boundary values will be as follows:

$$OPT[0] = 0$$
$$OPT[v] = \infty \; if \; v < 0$$

We fill in the *OPT[]* array in the following order:

```
for (int i = 1; i <= v; i++) {
    int min = infinity;
    for (int x : {$X_1, X_2, \dots, X_n$}) {
        if (OPT[i – x] + 1 > min) min = OPT[i – x] + 1;
    }
    OPT[i] = min;
}
```

At the end if $OPT[v] \le k$ we return success, otherwise we return failure.

2) A table composed of $N \times M$ cells, each having a certain quantity of apples, is given. You start from the upper-left corner. At each step you can go down or right one cell. Give an algorithm to find the maximum number of apples you can collect.

Solution outline:

Optimal cost equation is
S[i][j] = A[i][j] + max(S[i][j-1], if j>0 ; S[i-1][j], if i>0)

```
S[i][j] = 0, for i <0 or j<0
For i = 0 to N - 1
        For j = 0 to M - 1
                S[i][j] = A[i][j] +  max(S[i][j-1], if j>0 ; S[i-1][j], if i>0)
```

Output S[n-1][m-1]

3) Suppose you have a DAG with costs $c_e > 0$ on each edge and a distinguished vertex s. Give a dynamic programming algorithm to find the most expensive path in the graph that begins at s. Prove your algorithm's runtime and correctness. For full credit, your algorithm's runtime should be linear.

Solution:

Let's consider the vertices in topological order. Our recurrence is then:

LP(v) = max {LP(u) + c(u,v)} if (u,v) ∈ adj[u]

We can fill in this table in increasing order of v, with a base case that s has LP(s) = 0 and each vertex has a starting value LP(v) = -∞ to ensure that any that aren't reachable from s aren't considered to be on the maximum path.

We want the largest LP(v) value; if we track the value of the maximal u, we can use that to backtrack and find the longest path. Our total runtime is O(m + n).

4) Consider a two-dimensional array A[1:n,1:n] of integers. In the array each row is sorted in ascending order and each column is also sorted in ascending order. Our goal is to determine if a given value x exists in the array.

   a. One way to do this is to call binary search on each row (alternately, on each column). What is the running time of this approach? [2 pts]
   b. Design another divide-and-conquer algorithm to solve this problem, and state the runtime of your algorithm. Your algorithm should take strictly less than O(n^2) time to run, and should make use of the fact that each row and each column is in sorted order (i.e., don't just call binary search on each row or column). State the run-time complexity of your solution.

Solution:

a) O($n\log n$).
b) Look at the middle element of the full matrix. Based on this, you can either eliminate A$[1..\frac{n}{2}, 1..\frac{n}{2}]$ or A$[\frac{n}{2}..n, \frac{n}{2}..n]$. If x is less than middle element then you can eliminate A$[\frac{n}{2}..n, \frac{n}{2}..n]$. If x Is greater middle element then you can eliminate A$[1..\frac{n}{2}, 1..\frac{n}{2}]$. You can then recursively search in the remaining three $\frac{n}{2} \times \frac{n}{2}$ matrices. The total runtime is T(n) = 3T($\frac{n}{2}$)+O(1), T(n) = O($n^{\log_2 3}$).

5) A tourism company is providing boat tours on a river with n consecutive segments. According to previous experience, the profit they can make by providing boat tours on

segment i is known as a_i. Here a_i could be positive (they earn money), negative (they lose money), or zero. Because of the administration convenience, the local community of the river requires that the tourism company should do their boat tour business on a contiguous sequence of the river segments, i.e, if the company chooses segment i as the starting segment and segment j as the ending segment, all the segments in between should also be covered by the tour service, no matter whether the company will earn or lose money. The company's goal is to determine the starting segment and ending segment of boat tours along the river, such that their total profit can be maximized. Design an efficient algorithm to achieve this goal, and analyze its run time (Note that brute-force algorithm achieves $\Theta(n^2)$, so your algorithm must do better.)

Solution:

Using divide and conquer.

i) Divide operation: Divide the profit sequences of the n consecutive segments, denoted as A[1,n], as two part:
a_1,…..a_{n/2} and a_{n/2+1},…..a_n, denoted as A[1,n/2] and A[n/2+1, n] respectively.

ii) Merge operation:
Suppose you successfully find the maximum profit in A[1,n/2] and A[n/2+1, n], denoted as P_left(1,n), P_right(1,n), and the corresponding contiguous sequence of segements, denoted as B_left(1,n) and B_right(1,n)

Then the optimal contiguous segment can be in one of the three cases:

a) B_left(1,n)
b) B_right(1,n)
c) An optimal contiguous segment sequence crossing A[1,n/2] and A[n/2+1, n], denoted as B_cross(1,n).

For B_cross(1,n), denote the corresponding profit as P_cross(1,n). The method to confirm B_cross(1,n) and P_cross(1,n) is as follows:

- Starting from sequence [a_{n/2},a_{n/2+1}], compute the summation S_left(1) = a_{n/2}+a_{n/2+1} as one candidate solution for P_cross(1,n).
- Next, including a_{n/2-1} into the above sequence as [a_{n/2-1}, a_{n/2}, a_{n/2+1}], compute the summation of the three elements in the sequence as the second candidate solution: S_left(2)=S_left(1)+a_{n/2-1}.
- Keep including the element one by one to the left until a_1 is included, during each step, compute and record the summation values.
- Find S_opt_left = max_{i} {S_left(i)}, record the corresponding index of the included element that achieves the maximum, say i_cross*. Then i_cross* is the optimal starting index for B_cross(1,n);

- Starting from [a_{i_cross*},……, a_{n/2+1}] includes the element to the right one by one until a_n is included, during each step, compute the summation values in the same way as in the left part, denote the value as S_right(i).
- Find P_cross (1,n) = max_{i} {S_right(i)}, record the corresponding index of the included element that achieves the maximum, say j_cross*. Then j_cross* is the optimal ending index for B_cross(1,n);

Then

Maximum Profit = max{P_left(1,n), P_right(1,n), P_cross(1,n)}, and the corresponding optimal starting index and ending index are the ones that achieve the maximum.

iii) Before doing step ii) for A[1,n], recursively run the above procedure in each array A[1,n/2] and A[n/2+1, n] and so on.

iv) Termination condition: for A[i,j], if i==j, return a_i as the maximum profit, return i as both the optimal starting and ending index in A[i,j].

Complexity:

The Divide operation takes time O(1).

The Merge operation takes time O(n).

Define T(n) as the running time,

T(n) = 2*T(n/2) + O(n)

The complexity is O(nlog(n)).