CX 4140  
HW 4

Hardik Sangwan

Preamble:

1. People worked with: NA
2. Sources Used: Class Slides
3. Assignment Impressions: Liked the creative writing/description

**Problem 1: Ice Cream Buckets Equal Likeability**

Steps to prove NP-Completeness -

1.Given Problem Y is in NP:

This can be proved by showing that a potential solution can be verified in polynomial time for the given problem. For the given problem, given two set of flavors with likeability scores, we can calculate the total likeability score simply by adding all the scores in the given sets. This addition is polynomial time.

2. Choose another NP complete problem X: Subset Problem

3. X is poly-time reducible to Y:

* Convert Inputs of X to inputs of Y in polynomial time.
* Transformation – Split the input (set of numbers) of X so that one set of input sums up to k’ and another sums to sum–k’. Now add 3k –k’ to first set and 2k+k’ to the second to get a total sum of 3k and 2k+sum for both sets. At this point the input of X has been converted to the input of Y in polynomial time since only operations such as addition/splitting were done.
* Equivalence – If the subset problem X has a given value of k = sum, then the sum for both sets becomes 3k and 3k (2k+sum(=k)), which is a solution for Y.
* If the transformed input is a solution for Y then the original input must be a solution for X. Since 3k = 2k+sum, k = sum. So, when the two sets are equal sum = k = solution for problem X as well as the input for problem X.
* The 2 equivalences prove that the Ice Cream Buckets Likeability problem is also NP-complete like the Subset Problem.

**Problem 2: Pygmalion**

1. Given problem Y is in NP:

This can be proven by showing that given a certificate, we can verify if it is a solution in polynomial time.

Given a certificate as a list of k bunkers and knowledge of graph G (u,v), we need to verify if each town has a bunker of its own or is connected directly to a bunker town. For checking this do the following:

For each town, check if it has a bunker.

If yes, go to next town.

If no, check if every neighbor of this town has a bunker. If one of the neighboring towns has a bunker, we check the next town. If not, we return false.

The time complexity of the above algorithm is O(n2) since we check each town and all of its neighbours for worst case. This is polynomial time.

2. Choose another NP complete problem X: Vertex Cover

Vertex Cover: Given a graph G, we have a set of vertices <=k such that this set covers each edge i.e. for each edge(u,v), either u or v belongs to the set of k vertices.

3. Problem X is poly-time reducible to Problem Y:

* Convert Inputs of X to inputs of Y in polynomial time.
* Transformation – If we consider each node given in vertex cover as a town then set of vertices k is the k bunkers. This is constant time computation. Connection between towns is the edge.
* Equivalence – If input of X is a solution then Y also has a solution. If we are given a set of k vertices such that for each edge either u or v lies in the set then we can prove Y has a solution as well using contradiction. Let’s say the given solution is not a solution for the Pygmalion problem This mean that there exists a town which neither has a bunker nor is connected to a town with a bunker. This means the case where a node is neither in the k set of vertices nor is it a part of any edge where the other node is in the set of k vertices. But this would mean that the given set k is not a solution for the Vertex Cover problem and hence the contradiction leads to the fact that the solution must also be a solution for the Pygmalion problem.
* Equivalence – If input of Y is a solution then X also has a solution. Given a set of k towns such that every town either has bunker or has a neighbor with a bunker then we can prove Vertex Cover has a solution using contradiction. Set of k town equates to k vertices for the vertex cover problem. So assume the k vertices is not a solution for the vertex cover problem. This means that there is a node that is neither in the set of k vertices nor is it a part of any edge for which the other node is part of the set. But this would mean that there is town with neither a bunker nor a neighbor with a bunker. This contradiction leads to the fact that the solution works for the Vertex Cover problem as well.
* The 2 equivalences show that the Pygmalion problem is NP complete like the Vertex Cover problem

**Problem 3: Investment**

Problem Description:

Given an array with a sequence of interest rates, find max contiguous sub-array with max sum

Divide & Conquer Algorithm:

Algorithm:

* Find middle point and split array into two halves using that middle point.
* Recursively split left half till we reach the base case and get the final sum for left half.
* Recursively split right half till we reach the base case and get the final sum for right half.
* Find the max sum from the sub-array overlapping both the halves.
* Find the max sum from these three values and return that sum while moving up.

Additionally, return the indices along with the sums through the recursion.

Result Cases -

Case 1: Max sum sub-array from left half

Case 2: Max sum sub-array from right half

Case 3: Max sum sub-array from combination of the left and the right half with middle point in the solution.

Time Complexity: T(n) = 2T(n/2) + O(n) = O(n\*logn) using Master Theorem.

Space Complexity: O(1), no extra space required except for recursive stack calls.

Dynamic Programming Algorithm:

Algorithm (from HW description):

For any day j, let B[j] be the max sum of interest rates that can be obtained till day j. Then B[j] is either the max interest rate Devan can obtain upto j-1 plus the interest rate for the jth day if that sum is greater than 0 or 0 if it is negative.  
Recurrence Relation:

If j = 0, B[j] = 0

Otherwise B[j] = max(B[j-1]+aj, 0)

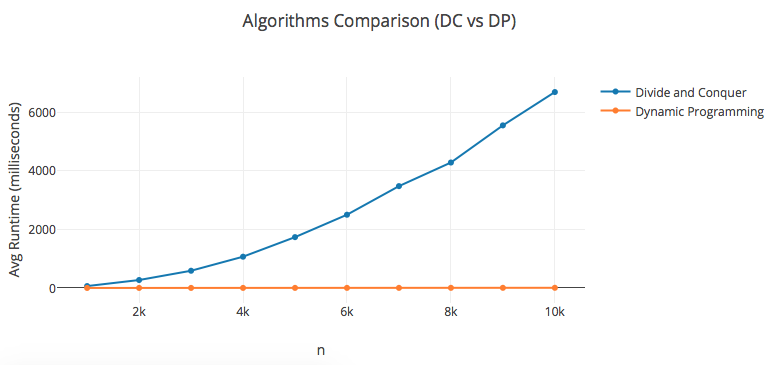
Using a bottom up approach a max sum can be calculated for each index and for each iteration. To find and keep track of the indices, a temporary start variable is used to keep track of when the max sum becomes negative so that the ending variable is then simply the current index when the current max sum is greater than the total max sum.

Time Complexity: O(n), loop runs for number of days n.

Space Complexity: O(1), constant space.

Graph/Comparison:

Curved graph for Divide and Conquer ( O(nlogn) complexity) and Linear Graph for dynamic programming ( O(n) time complexity). Additionally, divide and conquer takes much longer to run compared to dynamic programming due to additional comparisons and loops in the recursion and rises faster due to the bigger complexity O(nlogn) > O(n).



A separate graph zoomed in at the dynamic programming results shows a linear relationship with increasing n:

