3.4 Documentation

Group Members and Contribution

Ben Cassidy - Divide and Conquer
Audrey Chavarria - Brute Force
Dennis Kenyon - Dynamic Programming, Testing Arrays, Reading and Writing Methods
Zheng Yang - Brute Force

Brute Force Solution

Solution Description:
The brute force solution checks for every possible solution that is based on the elements of the first row. As this makes sense, that is it starts from the initial post to any of the other posts until it reaches the final post. The algorithm is based on the idea of a truth table but yet we only care about a set of these “truths” paths. First we halve the set that consists of initializing at posts other than the first one, since these would not make sense. That cuts half of the possible solutions. Then out of the ones that start at the initial position we only keep the ones that get to the final post. out of the 2^k, where k is the number of posts, we first halved it, (2^k)/2 and then halved it again (2^k)/4.

Complexity Analysis:
The brute force algorithm runs in triple for loops. So run time complexity is O(n^3).

Results Analysis:
Our brute force algorithm shows the minimum cost and We have backtracking method to show the path of the lowest cost.

Errors in Code:
No known errors in code.

Divide and Conquer Solution

Solution Description:
The divide and conquer solution uses two methods, divideAndConquer and divideAndConquerHelper. The reason for the helper is to pass the start and end points of the array which simplifies the logic of the problem. Like most divide and conquer solutions, this uses recursion to move from column to column iteratively. However, it is subject to re-calculating each possible path as it moves forward. For example, a 10x10 table would have about 1000 overlapping subproblems.

Complexity Analysis:
The divide and conquer solution runs in exponential time due to the overlapping subproblems. While it only calculates for the half of the array with values, it is still in exponential time.

Results Analysis:
Divide and conquer runs on an average of 13ms for a 15x15 table. However as the table size increases the time increases at an exponential rate as is expected. Our graphs show this as the only input we could run in a reasonable amount of time was n=25, which took 5 minutes to calculate. An input of n=30 would take approximately 4 hours to compute. The runtime of the recovery algorithm is linear.

Errors in Code:
No known errors in code.

Dynamic Programming Solution

Solution Description:
Since this was a dynamic programming problem, a helper two-dimensional matrix was created with the same dimensions as the original problem's two-dimensional matrix in order to keep a 'running tally' of the current minimum value for that particular path between posts by adding previously generated costs with the value in the original's array. Once the helper 2D was generated and filled out, the same recursive approach used in subset sum calculation was used to get the exact sequence of canoe posts (backtrackTrace() and backTrack() methods in the code).

Complexity Analysis:
This approach probes the 2D array and fills out the helper 2D array only as many times as there are elements in the original 2D array. In this case, it's the number or rows multiplied by the number of columns, giving a time complexity of n^2.

Results Analysis:
The results for the dynamic programming approach are uniform and nearly instant all the way up to even the largest sized array (800 x 800), costing at most about 7 milliseconds to compute and go through for the largest sample size and around 0 to 1 milliseconds to compute for the smallest sample size of 100. The reason why the dynamic programming approach doesn't blow up when the sample size is scaled up is because it doesn't have overlapping subproblem computation, and therefore doesn't exponentiate the amount of work needed; every solution to every problem is solved exactly once.

Errors in Code:
No known errors in code.

3.6 Analysis of Results

1. Quality of Solutions - All solutions provide the same results with the same path. For example, for a 15x15 array, all algorithms provided a minimum cost of 10 with a path of Post 0 to 4, Post 4 to 7, and Post 7 to 19.

2.Running time of algorithms: