1 Problem 3

1.1 Algorithm

Let $f(i, j)$ denote the maximum cost of a path from $(0, 0)$ to (i, j) and $g(i, j)$ to be maximum cost of a path from (m, n) to (i, j). Let Backward-Space-Efficient-Path have an analogous recurrence relation as Space-Efficient-Path, but move backwards starting from (m, n). Assume n is a power of 2.

Divide-and-Conquer-Path(A, B)

1. Let m be the number columns
2. Let n be the number of rows
3. Let P be an empty list
4. If $m \leq 2$ or $n \leq 2$
   - Compute optimal path using the algorithm designed on the prelim
5. Call Space-Efficient-Path$(m, n/2)$
6. Call Backward-Space-Efficient-Path$(m, n/2 + 1)$
7. Let q be the index maximizing $f(q, n/2) + g(q, n/2)$
8. Add $(q, n/2)$ to global list P
9. Divide-and-Conquer-Path$(M[1 : q], M[1 : n/2])$
10. Divide-and-Conquer-Path$(M[q + 1 : n], M[n/2 + 1 : n])$
11. Return P

Space-Efficient-Path(m, n)

1. Array $M[1...m, 1...2]$
2. Set $M[1, 1] = v_{1,1}$
3. For $i = 2$ to m, set $M[i, 1] = v_{i,1} + M[i - 1, 1]$
4. For $j = 2$ to n
   - set $M[1, 2] = v_{1,2} + M[1, j - 1]$
   - For $i = 2, ..., m$
     - set $M[i, j] = v_{i,j} + max(M[i, j - 1], M[i - 1, j])$
5. Move row 2 of M to row 1 to make room for next iteration by updating $M[i, 1] = M[i, 2]$ for each i

1.2 Time and Space Complexity

The Space-Efficient-Path uses a m x 2 and 1 x n array to store the previous row and column to calculate the next row and column. Hence, this uses $O(m+n)$ space. Similarly, Backward-Space-Efficient-Path uses $O(m+n)$ space. In Divide-and-Conquer-Path, we work on one recursive call at a time and reuse the working space from one call to the next. We also maintain a globally accessible list P which holds the nodes of the robot's optimal path as they are calculated. P will be at most $m+n$ entries, as the robot's path cannot use more than this many edges. Thus, the total space usage is $O(m+n)$.
The total algorithm runs in $O(mn)$. Let $T(m,n)$ denote the maximum running time of the algorithm given a board of values of size m x n. Space-Efficient-Path and Backward-Space-Efficient-Path use $O(mn)$ time to build arrays M (for Space-Efficient-Path) and M' (for Backward-Space-Efficient-Path) as proved on the prelim. To find index q, we iterate through columns $n/2$, $n/2+1$ and for $q = 1...m$ find the maximum sum of pairs $(q, n/2), (q, n/2+1)$. This is bounded by $O(m)$ as each iteration takes constant time. The rest of the algorithm then runs recursively on boards of size $qxn/2$ and $m - qxn/2$. Then, for some constant c and index q, $T(m,n)cmn + T(q, n/2) + T(mq, n/2) = 2cmn$ based on the recursion tree.

1.3 Correctness

From the prelim, we are given that Space-Efficient-Path and Backward-Space-Efficient-Path will return the maximum reward possible to square $M[i', j']$ for some integers i',j'. We can show by induction on $i+j$ that the algorithm outputs the correct answer for $M[i,j]$. In the base case, let $i = 1$ or $j = 1$. Our base case holds as $M[1,1] = v_{1,1}$ is the maximum value. Then, assume that for some $q < i$ the algorithm returns the maximum reward possible to square $M[q, j/2]$. To find the maximum value to position (i,j) we find the maximum value to position $(q, j/2)$ using the Space-Efficient-Path algorithm and $(m-q, j/2)$ using the Backward-Space-Efficient-Path algorithm. Thus, by the inductive hypothesis, the maximum reward possible to square M[i, j] is also correct.