Com S 311

Homework 3

1. Elevator Problem
   1. I am not sure how to do this problem. I thought it through but I am still not able to grasp how to solve it without knowing what the buttons could be.
2. Cookie problem
   1. Turn = 0 /\*static data\*/
   2. Winner[][] /\*array of size n by m\*/
   3. If : n < 2 && m < 2 **print no winner then quit**
   4. Else:
      1. For I = 0 to n
      2. For j = 0 to m
      3. Winner[i][j] = null
      4. MemoizedFindWinnerRec(n,m)
      5. If: winner[n][m] == 1 print Morgan
      6. Else : print Riley
   5. ---------------------------------------------
   6. MemoizedFindWinnerRec(int n, int m)
   7. If: n -2 < 2 && m-1 < 2
   8. Turn = .
      1. If : Turn % 2 == 0 return 0
      2. Else : return 1
   9. Else if: m-2 < 2 && n-1 < 2
   10. Turn = .
       1. If : Turn % 2 == 0 return 0
       2. Else : return 1
   11. Else :
       1. If : winner[n][m] == null
       2. Winner[n][m] = max(MemoizedFindWinnerRec(n-2 ,m-1), MemoizedFindWinnerRec(n-1, m-2))

This algorithm to find the winner behaves like a divide and conquer which would give it a runtime of O(logn) prior to the recursive call it runs through a double for loop of size n and m so my final runtime I believe would be **O(n\*mlog(n) )**

1. String and alphabet problem
   1. Let X = <x1, x2, …xm> let Y = <y1, y2, …yn>
   2. Using the recursive model solution it can be said that the structure still holds because for the first condition nothing needs to be changed but for the second and third condition it just adds another check where for ii) it will make sure it isn’t in Γ before adding it to the sequence and for iii) it will check again to make sure both are not inside of Γ and then it will take the longest sequence
      1. C[i, j] = 0 when I = 0 or j = 0;
      2. C[i, j] = C[i-1, j-1] when i, j > 0 and xi = yi && xi is not in Γ
      3. C[i, j] = max(c[i, j-1], c[i-1, j]) if i, j > 0 xi ≠ yi && xi is not in Γ && yi not in Γ
2. Minimum Spanning Tree Problem
   1. Kruskal’s algorithm creates a minimum spanning tree using the idea from *Theorem 23.1* which states that the current Graph will always be acyclic and uses the idea of only adding safe edges to the forest. During this process, many minimum spanning trees are created until there are no more vertices then loops through again and then will finally terminate once there is only one big tree remaining. During the process of connecting the forest (many trees) into a singular tree Kruskal’s algorithm goes through all the edges that connect any two trees together and picks least weighted edge. (Can also be thought of as a Greedy Algorithm because at each step it grabs the least weighted edge). So, given a Graph G(V, E) with a weight function c : E → R that for some vertex v ∈ V the smallest weighted edge for v will be in some MST based on Kruskal’s Algorithm which uses Theorem 23.1 and adds safe edges with the minimum weight and Corollary 23.2 which ensures the graph stays acyclic and that the edge to add is a safe weight