Phuc Dang
Ptd328
EE360C
Lab 3

Lab 3 Report

Part 1a:

The pseudocode for determining the maximum fun level: Implementing Knapsack Algorithm

M[i, r] represent max fun level at activity i at a risk budget r

f[i] represent fun level of activity i

r[i] represent risk level of activity i

```
//Initializing fun level at all risk budget for no item to 0

for (r = 0 to r = R)

     M [0, r] = 0;

//KnapSack algorithm

for (i = 1 to i = n)

    for (r = 0 to r = R)

        if (r_i > r)

                M[i, r] = M[i – 1, r];

        else

                M[i, r] = max(M[i – 1, r] , f_i + M[i – 1, r – r_i];

Return M[n , R];
```

The time complexity of the algorithm shoyld be O(nR) with ne being the number of activities and R being the budget risk.

Part 1b:

M[i, r] represent max fun level at activity i at a risk budget r

Name[i, r] represent activities included in M[i, r]

f[i] represent fun level of activity i

r[i] represent risk level of activity i

```
//Initializing fun level at all risk budget for no item to 0
for (r = 0 to r = R)
    M [0, r] = 0;
    Name[0, r] = "";
//KnapSack algorithm
for (i = 1 to i = n)
    for (r = 0 to r = R)
        if (r_i > r)
            M[i, r] = M[i – 1, r];
            Name[i, r] = Name[i – 1, r];
        else
            if not select item i
                Name[i, r] = Name[i – 1, r];
                M[i, r] = M[i – 1, r];
            If select item i
                Name[i, r] = nameList[i – 1, r – r_i] + item i;
                M[i, r] = f_i + M[i – 1, r – r_i];
Set = Name[n, R];
Return M[n , R];
```

For part 1b, I added a small modification which is having a 2d string array to keep track of what activities is included. The run time complexity should not changed because updating the array will only take O(1). I need to use a string split at the end, and the time complexity for the string split should be O(n) with n being the item. So the time complexity shouldn't change.

Part 2:

result[] is the schedule

M[i] is the min cost of staying in Maui at ith day

O[i] is the min cost of staying in Oahu in ith day

costM[i] = cost to stay at Maui in day i

costO[i] = cost to stay at Oahu in day i

fee is the transfer cost

```
//Initialize
stM[n] = true; //last day at Maui
stO[n] = false; //last day at Oahu
M[0] = N[0] = 0;
M[1] = costM[1]; //1st day at Maui
O[1] = costO[1]; //1st day at Oahu
for i = 2 to n
        M[i] = costM[i] + min(M[i – 1] , fee + O[i – 1]);
        O[i] = costO[i] + min(O[i – 1],  fee + M[i -1]);
if O[n] > M[n] then result[n] = staying at Maui else result[n] = staying at Oahu;
for i = n – 1 to 1
        if day i + 1 staying in Maui
                if cost at day i + 1 = cost staying in Maui at day i + 1+ cost at day i
                        then day i Fruitcake staying at Maui
                else day i Fruitcake staying at Oahu
        if day i + 1 staying at Oahu
                if cost at day i + 1 = cost staying at Oahu at day i + 1 + cost at day i
                        then day i Fruitcake staying at Oahu
                else day I Fruitcake staying at Maui
return result;
```

The run time of this algorithm is O(n) with n being the total day