CSE 222 (ADA) Homework Assignment 2 (Theory)

Shahzan Ahmad(2020117)
Divyansh Singh(2020060)

Problem 1:

Description of the subproblem:

Let F(i, j) denote the Maximum total length the monkey can jump considering trees i, i+1, . . . n-1,n, n+1 and tj is the tree just to the right of the monkey for all i from 1 to n+1 and all j from 1 to n+1. Where (n+1)th tree is an Imaginary tree at infinite distance from s and it has a banana with 0 jump value.

Recurrence:

base case:

F(i, j) = 0 *If (j > n) or (i > n)*

Recurrence:

F(i, j) = F(i, j+1) *if j < i*

F(i, j) = F(i+1, j) *if j > i*

F(i, j) = max{ (F(i+1, j+1)) , (bi + F(i+1, j'))} *if j == i*

where j' = index of tree just to the right of the coordinate (dist(j) + jump(j))

Subproblem that solves the original problem:

F(1, 1)

Pseudo Code:

```
// NOTE: I've used 1 based indexing everywhere.
// Pre calculate the indices of the trees just to the right of the coordinate
// dist[i] + jump[i]  for all i = 1 to n and store them in helper_array
int [ ] helper_Array = new int[ n ]
For i = 1 to n:
        j = i;
        While (dist[ j ] < dist[ i ] + jump[ i ]):
                if (j > n):
                        break;
                else:
                j = j + 1;
        helper_Array[i] = j;
```

```
// pseudo code for the recurrence
Let M be a 2-d array of size (n+1) x (n+1);
For i = n+1 to 1:
        For j = n+1 to 1:
                if( (j > n) or (i > n) ):  //base case
                        M( i, j) = 0;
                else if ( j < i):
                        M( i, j ) = M(i, j+1);
                else if ( j > i):
                        M( i, j ) = M(i+1, j);
                else if ( i == j):
                        M( i, j) = max{M(i+1, j+1),( M(i+1, helper_array[ i ])+jump[ i ] )}
return M(1, 1);
```

Time & Space complexity:

Time complexity: Filling the helper array will be O(n^2) because for each tree at worst we traverse the whole dist array.

And the code after that is obviously O(n^2) since two nested loops of size n+1.
Space complexity: Also we're using a 2-d array of size (n+1)x(n+1) hence space Complexity is O(n^2)

Problem 2:

Description of the subproblem:

Let F(i, j, k) denote the maximum distance monkey1 and monkey2 can jump in Total, considering trees i, i+1, . . . n+1, the tree just to the right of monkey1 is jth tree and the tree just to the right of monkey2 is kth tree. Where i varies from 1 to n+1, j varies from 1 to n+1 and k varies from 1 to n+1, (n+1)th tree is an imaginary tree at infinite distance from s and it has a banana with 0 jump value.

Recurrence:

base case:

F(i, j, k) = 0 *if (i>n) or ((j>n) and (k>n))*

recurrence:

F(i, j, k) = F(i+1, j, k) *if (j>i) and (k>i)*

F(i, j, k) = F(i, j+1, k) *if (j<i)*

F(i, j, k) = F(i, j, k+1) *if (k<i)*

F(i, j, k) = F(i+1, j+1, k) *if (j==i) and (j' == k)*

F(i, j, k) = max{F(i+1, j, k), jump[j]+F(i+1, j',k) } *if (j==i) and (j' != k)*

F(i, j, k) = F(i+1, j, k+1) *if (k==i) and (k' == j)*

F(i, j, k) = max{F(i+1, j, k), jump[k]+F(i+1, j, k') } *if (k==i) and (k' != j)*

F(i, j, k) = max{ F(i+1, j+1, k+1), jump[j]+F(i+1, j', k), jump[k]+F(i+1, j, k') }
if (j==i) and (k==i)

where j' is the tree just to the right of the coordinate dist(j) + jump(j)

and k' is the tree just to the right of the coordinate dist(k) + jump(k)

Subproblem that solves the original problem:

F(1,1,1)

Pseudo Code:

```
// NOTE: I've used 1 based indexing everywhere.
// Pre calculate the indices of the trees just to the right of the coordinate
// dist[i] + jump[i]  for all i = 1 to n and store them in helper_array
int [ ] helper_Array = new int[ n ]
For i = 1 to n:
        j = i;
        While (dist[ j ] < dist[ i ] + jump[ i ]):
                if (j > n):
                        break;
```

```
            else:
            j = j + 1;
        helper_Array[i] = j;

Let M be an array of size: (n+1)x(n+1)x(n+1)
For i = n+1 to 1:
        For j = n+1 to 1:
                For k = n+1 to 1:
                        if ( i>n) or (( j>n) and ( k>n)):
                                F(i, j, k) = 0;
                        else if ( j>i) and ( k>i):
                                F( i, j, k) = F( i+1, j, k);
                        else if ( j<i):
                                F( i, j, k) = F( i, j+1, k);
                        else if (k<i):
                                F( i, j, k) = F( i, j, k+1);
                        else if ( j==i ) and (k == helper_array[ j]): //2 has eaten jth bana
                                F( i, j ,k) = F( i+1, j+1, k);
                        else if ( j==i ) and (k != helper_array[ j]): //jth banana is there
                                F( i, j, k) = max{F( i+1, j, k),
                                                jump[ j]+F(i+1, helper_array[ j],k)};
                        else if ( k==i ) and ( j == helper_array[ k])://1 has eaten kth bana
                                F( i, j, k) = F( i+1, j, k+1);
                        else if ( k==i ) and ( j != helper_array[ k])://kth banana is there
                                F( i, j, k) = max{F( i+1, j, k),
                                                jump[k] + max{F( i+1, j, k),

                                                jump[k] + F(i+1, j, helper_array[k]) };
                        else if (i == j and i == k):
                                F(i, j, k) = max{ F(i+1, j+1, k+1),
                                                F(i+1, helper_array[ j], k),
                                                F(i+1, j, helper_array[ k]) }
Return M(1,1,1);
```

Time complexity:

Time complexity = $O(n^3)$, 3 nested loops of size n+1

Space complexity = $O(n^3)$, 3-d array of dimensions n+1 each.

Problem 3:

Description of the subproblem:

Recurrence:

Subproblem that solves the original problem:

Pseudo Code:

Time complexity: