Design and Analysis of Algorithms

Assignment - IV

Members:

Lekhana Mitta - IIT2019204

Sanskar Patro - IIT2019205

Aamin Chaudhari - IIT2019206

Contents:

Friends Pairing Problem

Given n friends, each one can remain single or can be paired up with some other friend. Each friend can be paired only once. Find out the total number of ways in which friends can remain single or can be paired up. Solve using Dynamic programming.

Algorithm:

Algorithmic Steps for calculating Friends Pairing Problem. :

To find the no. of ways given n friends are paired or remain single:

1) we input the no. of test cases and store it in variable t.

2) For every test case we use random number generation function to input the no. of friends for which we have to find the no. of ways they remain single or paired.

3) For every such input of t the rand() function returns a value which is stored in n, and passed as an argument to the function numOfCombos.

4) This function, we initialise a DP table locally.Here,we develop a bottom-up solution to find the no. of ways the friends remain single or paired up

Algorithm:

5) Iterate the value of i from 0 to n.If the value of i is less than or equal to 0,then store its value at index of i in DP table and return its value.

6) The above step is basically a base case if there are no friends or only 1 friend or 2 friends where no. of ways is possibly equal to no. of friends.

7) If the value of i is greater than 2 then there are two choices:

(i) if ith friend remains single , then we have to recur for i-1 friends which is basically finding the no. of ways i-1 friends are paired or remained single, it is given by value at index i-1 of DP table.

Algorithm:

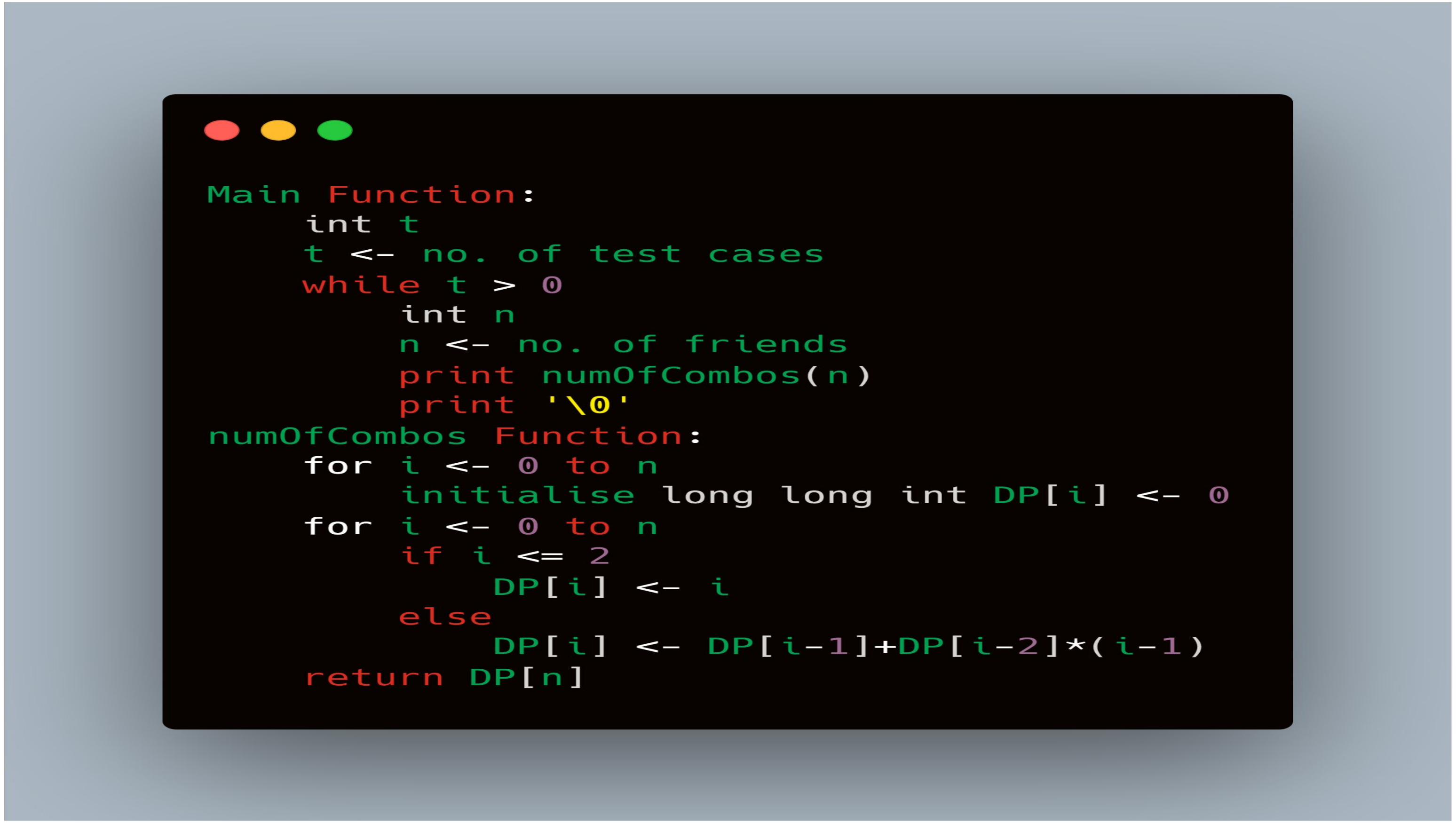
(ii) The ith friend pairs up with any of the i-1 friends then we solve it by finding number of ways a friend from i-1 friends is selected and no. of ways the remaining i-2 friends are paired up or remained single(i.e., the i-2 th index of DP table).

8) The equation for this explanation is DP[i] = DP[i1]+DP[i-2]*(i-1).

9) And finally return the value at index i of DP table.

- Similarly,it follows for every test case.This way can find the no. of ways in which n friends are paired up or remained single

Pseudo Code:



```
Main Function:
    int t
    t <- no. of test cases
    while t > 0
        int n
        n <- no. of friends
        print numOfCombos(n)
        print '\0'
numOfCombos Function:
    for i <- 0 to n
        initialise long long int DP[i] <- 0
    for i <- 0 to n
        if i <= 2
            DP[i] <- i
        else
            DP[i] <- DP[i-1]+DP[i-2]*(i-1)
    return DP[n]
```

Algorithm Analysis

A. Time Complexity Analysis:

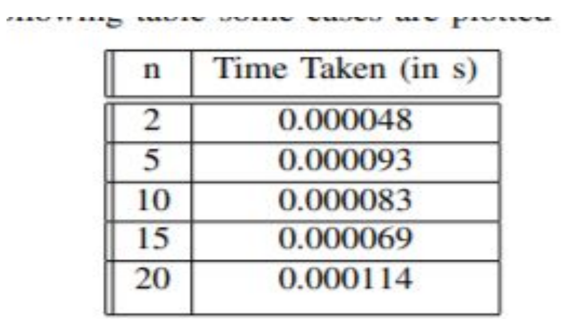
We can observe that, we just used two for loops ,in which one for loop is used for initialising DP and other for storing solutions of DP.Hence,complexity is calculated as O(n) where n is no. of friends.

B. Space Complexity :

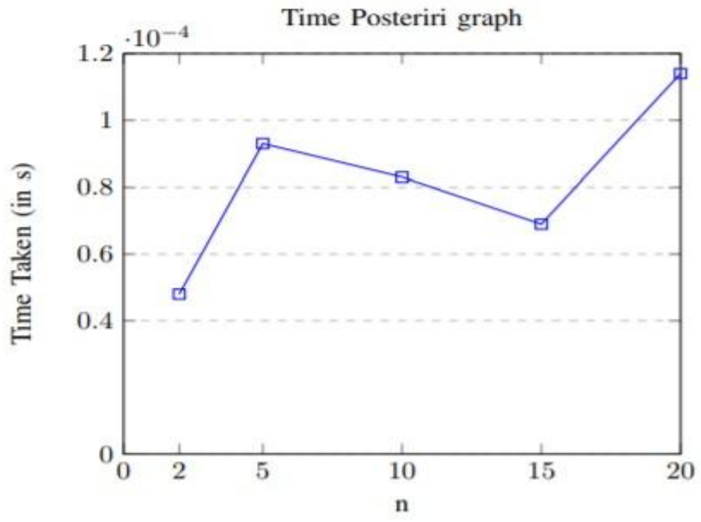
The space complexity of the Program is O(n) Because of the space allocation for resultant DP in each subproblem.We solve every sub-problem till the end , that can be produced

Algorithm Analysis:

Time Complexity:

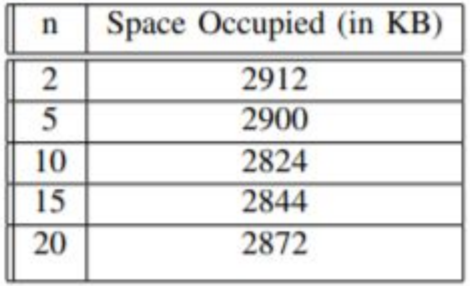

llowing table some cases are plotted

n	Time Taken (in s)
2	0.000048
5	0.000093
10	0.000083
15	0.000069
20	0.000114

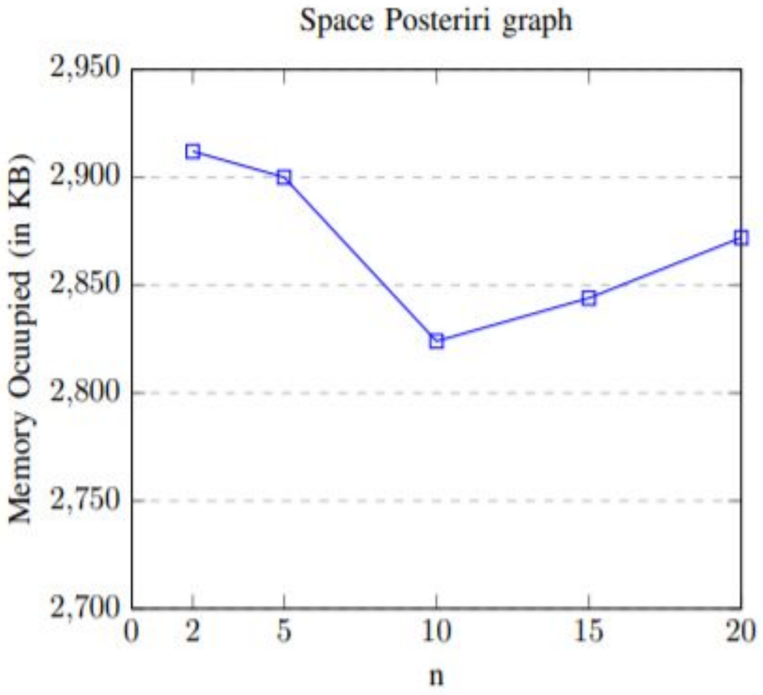


Algorithm Analysis:

Space Complexity:



n	Space Occupied (in KB)
2	2912
5	2900
10	2824
15	2844
20	2872



Conclusion:

We see that we are evaluating many sub-problem several times.It’s easily understandable that for such small input there are so many repeated computations, then for higher inputs, how many overlapping there will be. Here comes the need for DP, where we can use tabulation to solve bigger sub problems using smaller ones