Find out the total number of ways in which friends can remain single or can be paired up

IV semester - Bachelor's of Technology in Information technology with specialization in Business Informatics,

Indian Institute of Information Technology Allahabad, India

1st Aditya Raj
IIB2019007
iib2019007@iiita.ac.in
Adityahulk

2nd Shyam Tayal
IIB2019008
iib2019008@iiita.ac.in
shyamTayal

3rd Abhijeet Sonkar
IIB2019009
iib2019009@iiita.ac.in
Abhijeet-sonkar

***Abstract*—In this paper, we are devising an algorithm to find the total number of ways in which friends can remain single or can be paired up given n friends and every one of them can pair with each other using Dynamic Programming. This paper also analyzes the time and space complexity of the algorithms used and provides the most efficient approach to solve the given problem.**

***Index Terms*—iterative, recursion, bottom up, memoization, time complexity, space complexity, Dynamic Programming**

I. Introduction

Dynamic Programming is mainly an optimization over plain recursion. Wherever we see a recursive solution that has repeated calls for same inputs, we can optimize it using Dynamic Programming. The idea is to simply store the results of subproblems, so that we do not have to re-compute them when needed later. This simple optimization reduces time complexities from exponential to polynomial.

Steps of doing Dynamic Programming

Characterize the structure of an optimal solution. Recursively define the value of an optimal solution. Compute the value of an optimal solution in a bottom-up fashion. Construct an optimal solution from computed information.

II. Algorithm Design

We have devised two algorithms to Find the possible pairs among n friends.

Implementing steps of Naive Algorithm-¿ All possible combination of pair of n friends can be calculated with the help n-1 and n-2 friends We can solve this dependency with the help of recursion The recursion dependency calculates n as pair(n-1)*((n-1)*pair(n-2))

Algorithm 1: Naive Algorithm (Recursion)

```
function makePair(n) :

    if(n = 0 or n = 1)
        return 1

    selected = (n-1) * makePair (n-2)
    notSelected = makePair(n-1)

    return selected + notSelected;
```

The dependency relationship for any recursion problem can be solved using Dynamic Programming.

Here also for every recursion output,we can store it for future use,i.e memoization.

Every output is stored in variable,if called again,directly variable can be used,which reduces exponential time complexity to linear.

Algorithm 2: Eficient Algorithm (DP1)

```
friendsPairing(totPairs[],N) :

    totPairs[0]=1;
    totPairs[1]=1;

    for curr in range 2 to N :
        totPairs[curr] = totPairs[curr-1]
        totPairs+= (curr-1)*totalPairs[curr-2]

    return totPairs[N]
```

The dynamic programming can also be more optimized in space complexity by using bottom up approach.

For the recursion output,we can also use iteration to get current value from previous one.

The iteration apart from simplicity also reduces space complexity.

Algorithm 3: Eficient Algorithm 2 (DP2)

```
friendsPairing(n) :

    for curr in range 2 to n :
        totalPairs = prev+ (curr-1)*prev2
        prev2=prev
        prev=totalPairs

    if n = 0 or n = 1
         totalPairs=1

    return totalPairs
```

III. Algorithm Analysis

Time and Space Complexity Comparison		
Algorithms	**Time**	**Space**
Naive Algorithm	$O(2^N)$	O(N)
DP Algorithm 1	O(N)	O(N)
DP Algorithm 2	O(N)	O(1)

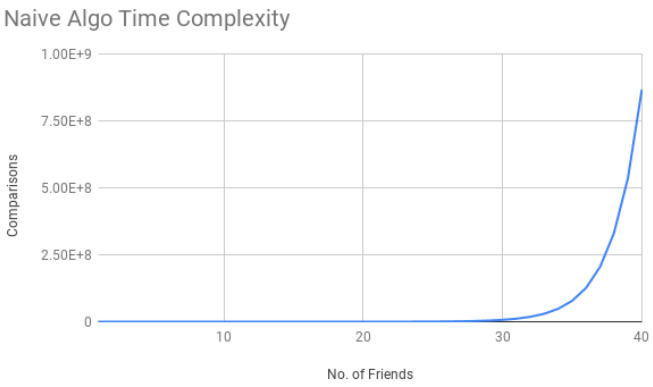

Fig. 1: Naive Algorithm

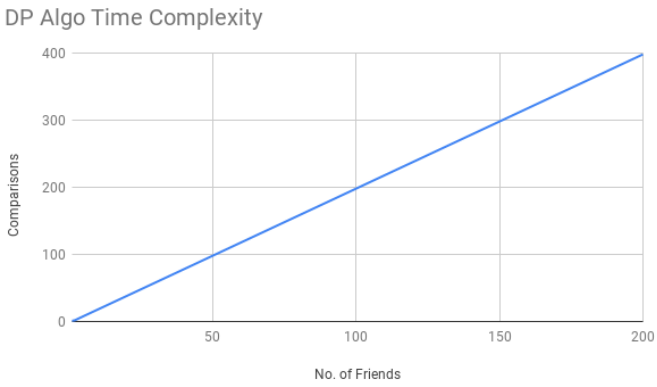

Fig. 2: DP Algorithm 1 VS N

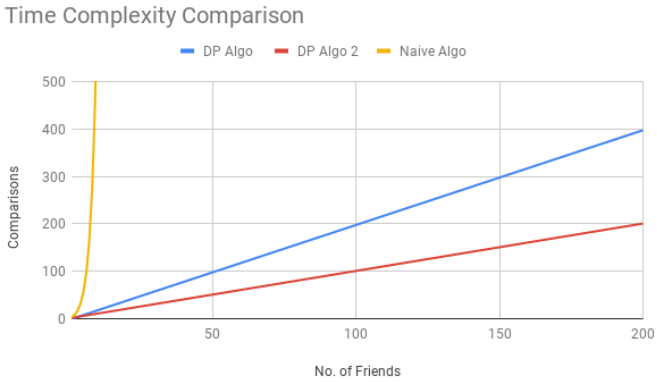

Fig. 3: Algorithm Analysis

IV. Conclusion

DP 2 is more efficient algorithm. The reason is both time complexity and space complexity equation. The second algorithm has O(N) i.e linear time dependency while first algorithm has $O(2^N)$ i.e. exponential time dependency.
The main reason behind this is the use of Dynamic Programming also by iteration that reduces both time complexity and space complexity. The recursion has time complexity problem but even DP1 i.e implemented by memoization has linear space complexity occurrence which is not present in iterative version i.e DP2.

References

[1] DP : https://www.geeksforgeeks.org/dynamic-programming/
[2] https://www.geeksforgeeks.org/tabulation-vs-memoization/