Dynamic Programming | Set 1 (Overlapping Subproblems Property)

Dynamic Programming is an algorithmic paradigm that solves a given complex problem by breaking it into subproblems and stores the results of subproblems to avoid computing the same results again. Following are the two main properties of a problem that suggest that the given problem can be solved using Dynamic programming.

In this post, we will discuss first property (Overlapping Subproblems) in detail. The second property of Dynamic programming is discussed in next post i.e. Set 2.

1) Overlapping Subproblems
2) Optimal Substructure

1) Overlapping Subproblems:
Like Divide and Conquer, Dynamic Programming combines solutions to sub-problems. Dynamic Programming is mainly used when solutions of same subproblems are needed again and again. In dynamic programming, computed solutions to subproblems are stored in a table so that these don't have to recomputed. So Dynamic Programming is not useful when there are no common (overlapping) subproblems because there is no point storing the solutions if they are not needed again. For example, Binary Search doesn't have common subproblems. If we take example of following recursive program for Fibonacci Numbers, there are many subproblems which are solved again and again.

```
/* simple recursive program for Fibonacci numbers */
int fib(int n)
{
   if ( n <= 1 )
      return n;
   return fib(n-1) + fib(n-2);
}
```

Run on IDE

Recursion tree for execution of *fib(5)*

```
                         fib(5)
                     /             \
               fib(4)                fib(3)
             /      \                /     \
         fib(3)      fib(2)         fib(2)    fib(1)
        /     \        /    \       /    \
  fib(2)   fib(1)  fib(1) fib(0) fib(1) fib(0)
  /    \
fib(1) fib(0)
```

We can see that the function f(3) is being called 2 times. If we would have stored the value of f(3), then instead of computing it again, we could have reused the old stored value. There are following two different ways to store the values so that these values can be reused:

a) Memoization (Top Down)

b) Tabulation (Bottom Up)

a) Memoization (Top Down): The memoized program for a problem is similar to the recursive version with a small modification that it looks into a lookup table before computing solutions. We initialize a lookup array with all initial values as NIL. Whenever we need solution to a subproblem, we first look into the lookup table. If the precomputed value is there then we return that value, otherwise we calculate the value and put the result in lookup table so that it can be reused later.

Following is the memoized version for nth Fibonacci Number.

C/C++

```
/* C/C++ program for Memoized version for nth Fibonacci number */
#include<stdio.h>
#define NIL -1
#define MAX 100

int lookup[MAX];

/* Function to initialize NIL values in lookup table */
void _initialize()
{
  int i;
  for (i = 0; i < MAX; i++)
    lookup[i] = NIL;
}

/* function for nth Fibonacci number */
int fib(int n)
{
   if (lookup[n] == NIL)
   {
      if (n <= 1)
         lookup[n] = n;
      else
         lookup[n] = fib(n-1) + fib(n-2);
   }

   return lookup[n];
}

int main ()
{
  int n = 40;
  _initialize();
  printf("Fibonacci number is %d ", fib(n));
  return 0;
}
```

Run on IDE

```python
# Python program for Memoized version of nth Fibonacci number

# Function to calculate nth Fibonacci number
def fib(n, lookup):

    # Base case
    if n == 0 or n == 1 :
        lookup[n] = n

    # If the value is not calculated previously then calculate it
    if lookup[n] is None:
        lookup[n] = fib(n-1 , lookup)  + fib(n-2 , lookup)

    # return the value corresponding to that value of n
    return lookup[n]
# end of function

# Driver program to test the above function
def main():
    n = 34
    # Declaration of lookup table
    # Handles till n = 100
    lookup = [None]*(101)
    print "Fibonacci Number is ", fib(n, lookup)

if __name__=="__main__":
    main()

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Run on IDE

b) Tabulation (Bottom Up): The tabulated program for a given problem builds a table in bottom up fashion and returns the last entry from table. For example, for the same Fibonacci number, we first calculate fib(0) then fib(1) then fib(2) then fib(3) and so on. So literally, we are building the solutions of subproblems bottom-up.

Following is the tabulated version for nth Fibonacci Number.

C/C++

```c
/* C program for Tabulated version */
#include<stdio.h>
int fib(int n)
{
  int f[n+1];
  int i;
  f[0] = 0;   f[1] = 1;
  for (i = 2; i <= n; i++)
      f[i] = f[i-1] + f[i-2];

  return f[n];
}

int main ()
{
  int n = 9;
  printf("Fibonacci number is %d ", fib(n));
  return 0;
}
```

Run on IDE

```python
# Python program Tabulated (bottom up) version
def fib(n):

    # array declaration
    f = [0]*(n+1)

    # base case assignment
    f[1] = 1

    # calculating the fibonacci and storing the values
    for i in xrange(2 , n+1):
        f[i] = f[i-1] + f[i-2]
    return f[n]

# Driver program to test the above function
def main():
    n = 9
    print "Fibonacci number is " , fib(n)

if __name__=="__main__":
    main()

# This code is contributed by Nikhil Kumar Singh (nickzuck_007)
```

Run on IDE

Output:

```
Fibonacci number is 34
```

Both Tabulated and Memoized store the solutions of subproblems. In Memoized version, table is filled on demand while in Tabulated version, starting from the first entry, all entries are filled one by one. Unlike the Tabulated version, all entries of the lookup table are not necessarily filled in Memoized version. For example, Memoized solution of the LCS problem doesn't necessarily fill all entries.

To see the optimization achieved by Memoized and Tabulated solutions over the basic Recursive solution, see the time taken by following runs for calculating 40th Fibonacci number:

Recursive solution

Memoized solution

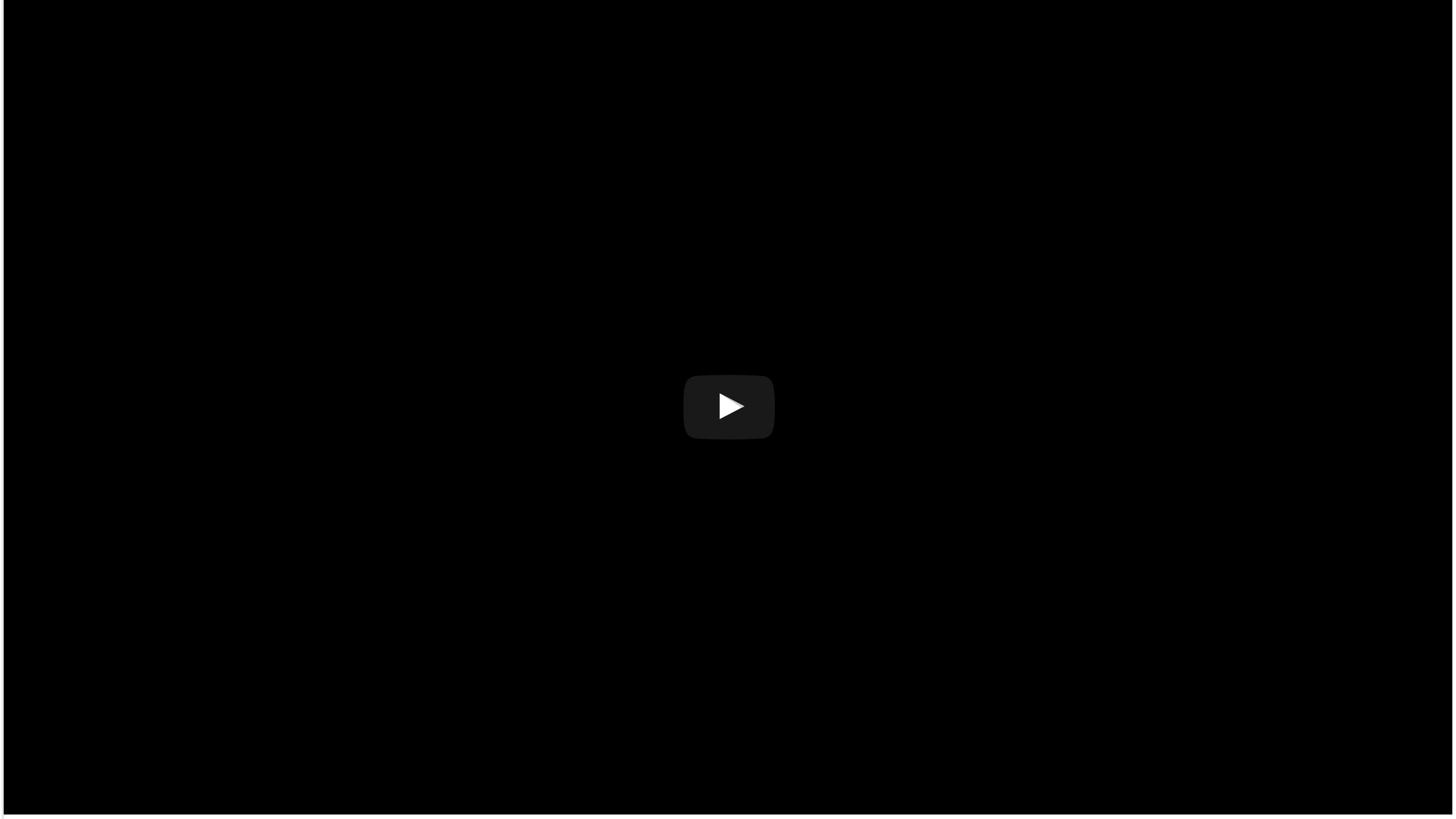
Tabulated solution

Time taken by Recursion method is much more than the two Dynamic Programming techniques mentioned above – Memoization and Tabulation!

Also see method 2 of Ugly Number post for one more simple example where we have overlapping subproblems and we store the results of subproblems.

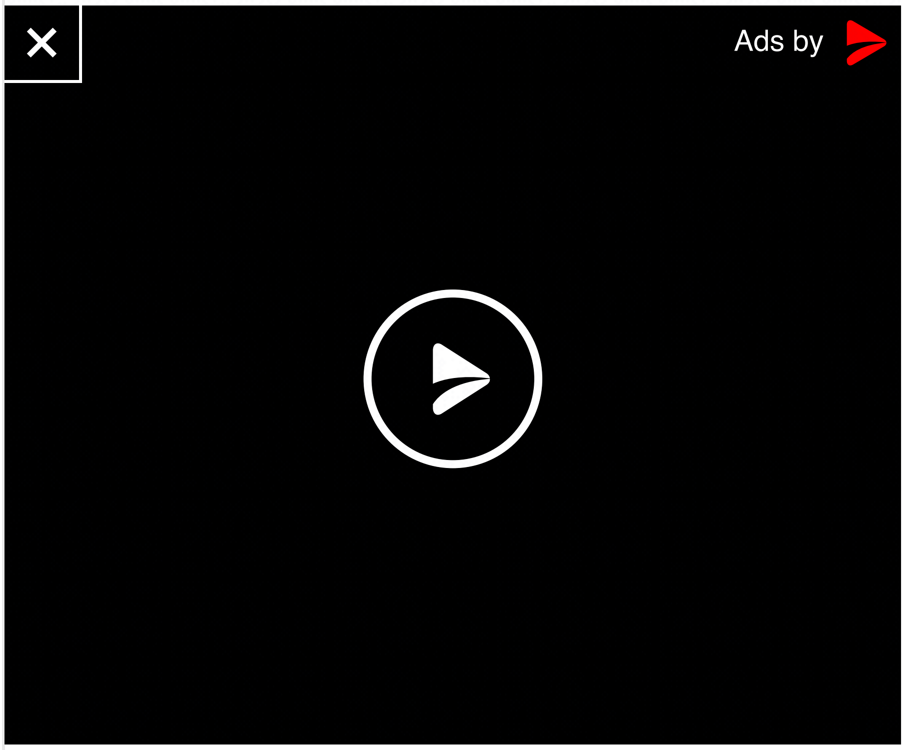
We will be covering Optimal Substructure Property and some more example problems in future posts on Dynamic Programming.

Try following questions as an exercise of this post.
1) Write a Memoized solution for LCS problem. Note that the Tabular solution is given in the CLRS book.
2) How would you choose between Memoization and Tabulation?



Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

References:
http://www.youtube.com/watch?v=V5hZoJ6uK-s



GATE CS Corner   Company Wise Coding Practice

Dynamic Programming Dynamic Programming Fibonacci

Recommended Posts:

Dynamic Programming | Set 2 (Optimal Substructure Property)
Dynamic Programming | Set 3 (Longest Increasing Subsequence)
Ugly Numbers
Dynamic Programming | Set 4 (Longest Common Subsequence)
Dynamic Programming | Set 5 (Edit Distance)

(Login to Rate and Mark)

1.6 Average Difficulty : **1.6/5.0**
Based on **156** vote(s)

Add to TODO List
Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Share this post!

85 Comments   GeeksforGeeks   Dagny T

Recommend 3 Share Sort by Newest

Join the discussion…

Ashish Ranjan • a month ago

we can use memset function for initializing the array with nil

Reply • Share ›

Anurag Mishra • a month ago

I have doubt about why memorization is faster than tabulation.Kindly anyone clear my doubt.

Reply • Share ›

Naseer Mohammad • 2 months ago

Please clear my doubt. Both the Memoized version and recursion Method are following same way of finding out fibonacci number. How come there is that much of difference between both

Reply • Share ›

Abrar Khan → Naseer Mohammad • 2 months ago

In memoized version of dynamic programming we store the result as top to down approach but in case of tabular we store results as bottom to up.

Reply • Share ›

Kaushal Gosaliya • 5 months ago

Solution to Java Language ::

http://code.geeksforgeeks.o...

Reply • Share ›

Kaushal Gosaliya • 5 months ago

JAVA Solution ::

```
import java.io.*;
import java.util.*;

public class First {

private static int MAX = 0;
private static int look[];

static void Initialization(){
for(int i=0;i<max;i++) look[i]="-1;" }="" static="" int="" fib(int="" n){="" if(look[n]="=" -1){=""
```

if(n="" <="1){" look[n]="n;" }else{="" look[n]="fib(n-1)" +="" fib(n-2);="" }="" }="" return="" look[n];="" }="" public="" static="" void="" main(string="" args[]){="" scanner="" sc="new" scanner(system.in);="" int="" n="sc.nextInt();" max="n;" look="new" int[max];="" initialization();="" system.out.println("="" ans="" is="" ::="" "="" +="" fib(--n));="" }="" }="">
Reply · Share ›

Kaushal Gosaliya ➦ Kaushal Gosaliya • 5 months ago
This is optimal solution of Fibonacci number
Reply · Share ›

This comment is awaiting moderation. Show comment.

Kaushal Gosaliya ➦ Mahendra • 5 months ago
Thank you for suggestions

Link is ::

http://code.geeksforgeeks.o...
Reply · Share ›

Shiva Kumar • a year ago
In C:

We can avoid #define MAX 100 and #define NIL -1.
Using calloc we can avoid NIL and there won't be need to intialize lookup.

http://code.geeksforgeeks.o...

Also, the problem can be solved in O(n) time with O(1) extra space but that would not use dynamic programming.
1 Reply · Share ›

DAVINDER PAL SINGH ➦ Shiva Kumar • 8 months ago
We can solve Fibonacci problem in O(logn) time, see Mathematical Algorithms on GeeksforGeeks
Reply · Share ›

Ravindra Kholia ➦ DAVINDER PAL SINGH • 6 months ago
thats counting the number of fibonacci terms in given range............ not the same n man .... that n is range
Reply · Share ›

Mains ➦ Shiva Kumar • 9 months ago

if(lookup[n] == 0)
lookup[n] = _fib(n-1, lookup) + _fib(n-2, lookup);
return lookup[n];

This is dynamic initialization.

Reply · Share ›

Vinoth · a year ago

In this fibonacci problem, can someone explain me how to calculate time complexity for all the three methods: 1. recursion, memoization, bottom-up.

Thanks in advance

Reply · Share ›

Mriganka Ghosh Vinoth · a year ago

Recursion:

Let's consider the fibonacci tree structure that forms as part of this solution. Every node fib(n) has two sub-nodes fib(n-1) and fib(n-2). So we can see that this tree is a binary tree.
Let's assume the first function (node) fib(n) for a positive n, is at level 0. Then at level one, we find the solution to the problem fib(n-1) and fib(n-2). Going by the same logic, say at level 'h' we find the solution to fib(0), which is the base case of fibonacci. Meaning the tree stops growing at this level

Hence, the height of the solution tree is 'h'.
We know, height h is given by:

h = log (N) [log base 2]
where, N = total number of nodes in the tree.

Thus, N = 2 ^ h

Now, we know that h = n (because to find fib(n) we have to traverse a tree of level n

see more

2 · Reply · Share ›

hassan Mriganka Ghosh · 8 months ago

thanks,for giving clear description.....

Reply · Share ›

Rahul Singal · a year ago

http://shadowhackit.blogspo...

must see these tricks and knowledgeable stuff

Reply • Share ›

Asen • a year ago

Is there optimal substructure property in fibonacci no

Reply • Share ›

shiva • a year ago

check http://cedric005.blogspot.i...
to continuously check output of programs just copying text.
easy to use

Reply • Share ›

Joshua Lamusga • 2 years ago

For reference, recursive time is 0.88s and memoized/tabulated at 0.00s, so it's fast.

Reply • Share ›

Akshay Aradhya • 2 years ago

The program to compute the 40th Fibonacci Number, is 4 years old and the link doesn't show the code :) Please Update it. Thank You

1 • Reply • Share ›

user007 → Akshay Aradhya • 2 years ago

But, it does show the time of execution at the bottom. Probably that was the point of all that!

Reply • Share ›

Holden → user007 • 2 years ago

there is no time of execution there ...

Reply • Share ›

dumborakesh → Holden • a year ago

Success #stdin #stdout 0.88s 2728KB

1 • Reply • Share ›

Guangming Wang • 2 years ago

Here is my space optimized O(1) solution using tabulation:

```
// bottom up space optimized
public static int fibTabulationOptimized(int n){
if(n < 2){
return n;
}
int prePre = 0, pre =1;
int ret = 0;
for(int i=2; i <=n; i++){
ret = prePre + pre;
prePre = pre;
pre = ret;
}

return ret;
}
```

Reply • Share ›

Hatem Faheem → Guangming Wang • a year ago

Sure this solution is more memory optimized, but the main idea behind the table is to save it outside the function (global) and check it everytime you call the function before doing any calculations.

So, if I call fib(50) it will fill the table from 0 to 50 then any call to fib with n>=0 and n<=50 should be done in O(1) by checking the table.

Reply • Share ›

Nguyễn Minh Tuấn → Hatem Faheem • 5 months ago

you good.. thanks for explain...

Reply • Share ›

Vibhor Garg • 2 years ago

In the memoization part, there should be a precondition to check that n is greater than equal to one, only then the loop should run.

1 • Reply • Share ›

Anil Bhaskar • 2 years ago

The tabulation method is a simple iterative way of computing Fibonacci, isn't it?

1 • Reply • Share ›

Navroze • 2 years ago

The answer to the second question is that we can use memoization in order to store the answer of some complex calculation and the tabulated version can be used in case we know an arithmetic progression.

Reply • Share ›

Jeremy Shi • 2 years ago

There is another O(lgn) solution .

Reply • Share ›

Ishan Gupta → Jeremy Shi • 9 months ago

Yeah, it's Matrix Exponentiation. It's a great method for competitive programming but not so good in interview preparation.

Reply • Share ›

vaibhav • 2 years ago

how is this code handling negative numbers like if i give n = -40 the output still comes out to be 0 ,why can someone please explain ?

Reply • Share ›

Manraj Singh → vaibhav • 2 years ago

How can you find fibonacci number -40th?

Reply • Share ›

vaibhav → Manraj Singh • 2 years ago

if (n <= 1)

lookup[n] = n;

this means that if i give n =-20 it will return lookup[-20]=-20 right?but my qustion is lookup[-20] aint possible only yet the output on n=-20 is 0

1 • Reply • Share ›

Sriram Ganesh → vaibhav • 2 years ago

I understsand ur question. Its not relevant to fibonacci.. still good qn..

Reply • Share ›

Manraj Singh → vaibhav • 2 years ago

It will give Segment fault!

1 • Reply • Share ›

SHIVAM DIXIT • 2 years ago

Can you please provide a post on space optimised knapsack?? In that we dont have to take such a large 2-d array....Its sometimes not possible to take array of size arr[n+1][capacity+1] for dp...example for such a problem is http://www.spoj.com/problem...

Reply • Share ›

Goky → SHIVAM DIXIT • 2 years ago

What we can do is we take array as arr[2][capacity+1].If you notice carefully we always require the results of row just one above the current row.So instead of using memory for all n items.We only consider the result of current and prev row only.So after computing current row,we will copy the content of current row to prev. row.We will continue to work this way till N items.If you have any question,please feel free to ask :)

2 ^ | ˅ • Reply • Share ›

Sivakumar K R ➦ SHIVAM DIXIT • 2 years ago

```
int knapSack1(int W, int *wt, int *val, int n)
{
int dp[W + 1];
memset(dp, -1e9, sizeof dp);
for (int i = 0; i < n; i++)
{
for (int j = W; j >= 0; j--)
{
int k = j + wt[i];
if (k <= W)
{
dp[k] = max(dp[k], dp[j] + val[i]);
}
}
cout<<'\n';
}
cout << '\n';
/*for (int j = 0; j <= W; j++)
cout << dp[j] << " ";*/
return dp[W];
}
```

^ | ˅ • Reply • Share ›

lucky • 2 years ago

lookup must be of long type

2 ^ | ˅ • Reply • Share ›

arsh • 3 years ago

First two links are broken

http://www.cs.uiuc.edu/clas...
http://web.iiit.ac.in/~avid...

^ | ˅ • Reply • Share ›

GeeksforGeeks Mod ➦ arsh • 3 years ago

Thanks for pointing this out. Looks like the pdf files have been removed from servers. We have updated the post.

Reply • Share ›

quantized • 3 years ago

First two links in References are broken.

2 • Reply • Share ›

dg • 3 years ago

Please give comparison between top down approach(memorization) and bottom up approach.
As in memorization(top down) we can avoid computation of some sub-problems(in some cases) on the other hand in bottom up approach we need to find solution of every sub problem.
And how should we choose between Memoization and Tabulation?

3 • Reply • Share ›

Ealham Al Musabbir → dg • 7 months ago

When the number of recursive call gets too large to reach the base case, we should avoid top down. For example, http://www.spoj.com/problem... , this problem can be solved with memoization for smaller input only, but for larger input (i.e greater than 1000), stack memory gets overflowed. So in this case, bottom up is the only way.

Reply • Share ›

ankit • 3 years ago

@GFG
which approach is better bottom-up or top-down?

Reply • Share ›

Anand Barnwal → ankit • 2 years ago

Memoization(top down) usually becomes more complicated to implement but it has some advantages in problems, mainly those which you do not need to compute all the values for the whole matrix to reach the answer like: LCS.

Tabulation(bottom up) is easy to implement but it may compute unnecessary values sometimes. It also has benefits of less overhead.

2 • Reply • Share ›

rihansh → Anand Barnwal • 2 years ago

Agree:D

Reply • Share ›

Gaurav Kapoor • 3 years ago

With Normal recursion OUTPUT for n=30 : time taken to calculate the fibonacci of 30 is 11 microseconds and value is 832040

With Memorization OUTPUT for n=30 : time taken to calculate the fibonacci of 30 is 1647 microseconds and value is 832040

Why its taking less time in Recursion ?? It should be the reverse way

12 • Reply • Share ›

Load more comments