

Sequences of given length where every element is more than or equal to twice of previous

1st Sumit Kumar Sahu (IIT2019069) 2nd Adetik Om Tyagi (IIT2019070) 3rd Subham Kumar (IIT2019093)
B.Tech Information Technology B.Tech Information Technology B.Tech Information Technology
IIIT Allahabad IIIT Allahabad IIIT Allahabad

Abstract—In this paper, we have devised 2 algorithms to find the number of possible sequences of length n such that each of the next element is greater than or equal to twice of the previous element but less than or equal to m , while optimizing time and space complexity. We have determined both the time, and the space complexity of our algorithm by illustrating through some graphs.

Index Terms—component, formatting, style, styling, insert

I. INTRODUCTION

For finding the number of possible sequences of length n such that each of the next element is greater than or equal to twice of the previous element but less than or equal to m can be solved using recursion. Recursion is the process of repeating items in a self-similar way. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function.

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

As per the given condition the n -th value of the sequence can be at most m . There can be two cases for n -th element.

- 1) If it is m , then the $(n-1)$ th element is at most $m/2$. We recur for $m/2$ and $n-1$.
- 2) If it is not m , then it is at most $m-1$. We recur for $(m-1)$ and n .

The total number of sequences is the sum of the number of sequences including m and the number of sequences where m is not included. Thus the original problem of finding number of sequences of length n with max value m can be subdivided into independent sub-problems of finding number of sequences of length n with max value $m-1$ and number of sequences of length $n-1$ with max value $m/2$.

II. ALGORITHM 1

A. ALGORITHM DESIGN

- 1) Take input of integers n and m in variables n and m respectively.
- 2) Call for the solve function with parameters as n and m which will give the answer.
- 3) Solve function is using a recursive approach to find the answer.

- 4) Base cases for this function is when m becomes zero then we will return zero and also when n will become one then we will return m .
- 5) Create a new variable ret initialized with zero and looping from 1 to m and adding answers from 1 to m in ret by recursively calling the solve function for smaller input.
- 6) Return the value of ret which has the final answer.

B. ALGORITHM

Input: N (length of sequence) and M (upper bound on sequence elements)

Output: Number of possible sequences of length n such that each of the next element is greater than or equal to twice of the previous element but less than or equal to m .

Method:

- 1) call the solve function with parameters as n and m : $solve(n,m)$;
- 2) if $m \leq 0$ return 0
- 3) if $n == 1$ return m
- 4) Declare and Initialise ret with 0.
- 5) Loop the current element(i) of the sequence from 1 to m .
 - 6) call the solve function with parameters $n-1$ and $i/2$. Add the returned value to ret
- 7) return ret ;

C. APRIORI ANALYSIS

Step	Time	FreqBest	Freq
Step1	1	1	1
Step2	1	1	m^n
Step3	1	1	m^n
Step4	1	1	m^n
Step5	1	1	m^n
Step6	1	1	m^n
Step7	1	1	m^n

For Best Case

$$T\Omega \propto 1*1+1*1+1*1+1*1+1*1+1*1+1*1$$

$$T\Omega \propto 7$$

Thus, the best case time complexity is $O(1)$ when $n=1$ OR m is non positive

For Worst Case

$$T\Omega \propto 1*1+1* m^n+1* m^n+1* m^n+1* m^n+1* m^n$$

$$T\Omega \propto 6* m^n$$

Clearly this equation will come out to be of the exponential

form .

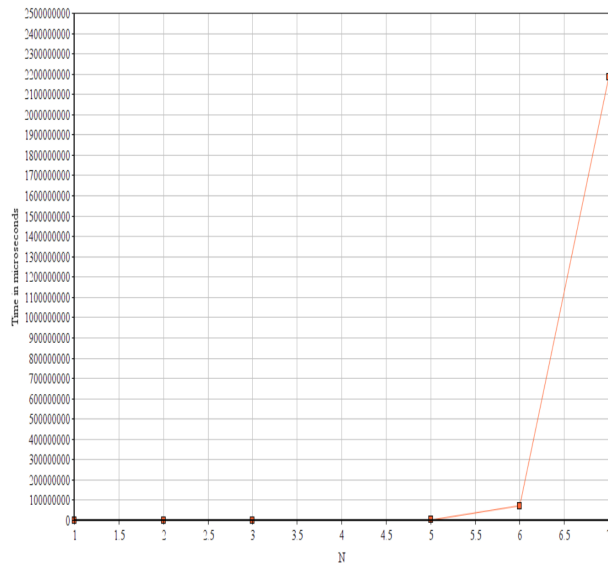
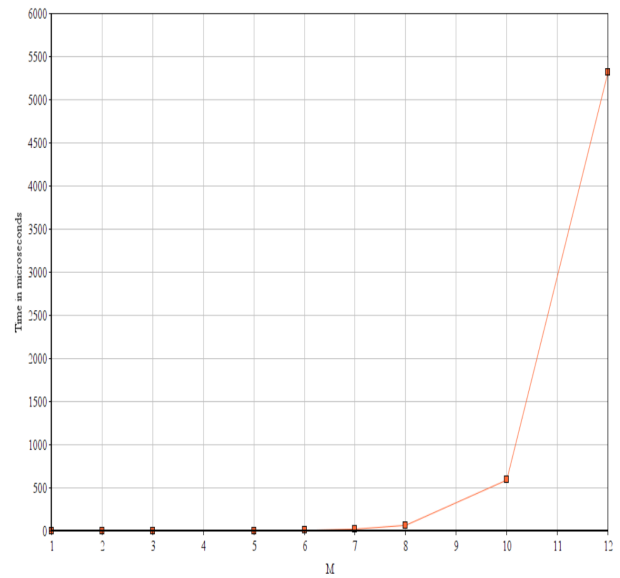
Thus , worst case time complexity is $O(m^n)$

Space complexity is $O(1)$

Time complexity –

Fix $m=30$ and vary n

n	Time
1	0.3
2	0.9
3	270
5	243000
6	7290000
7	218700000



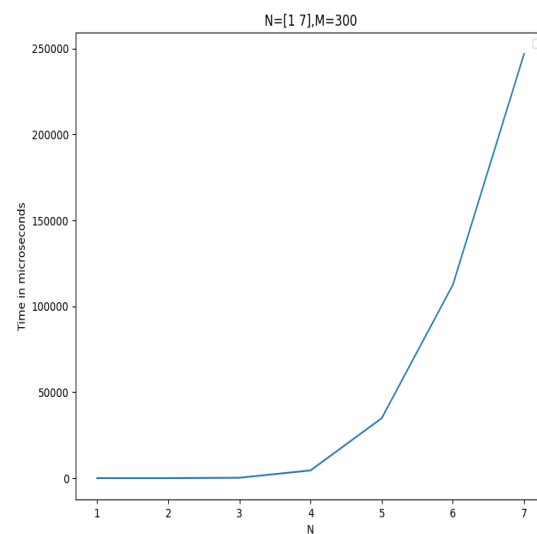
Fix $n=3$ and vary m

m	Time
1	0.03
2	0.09
3	0.27
5	2.43
6	7.29
7	21.87
8	65.61
10	590.49
12	5314.41

D. APOSTERIORI ANALYSIS

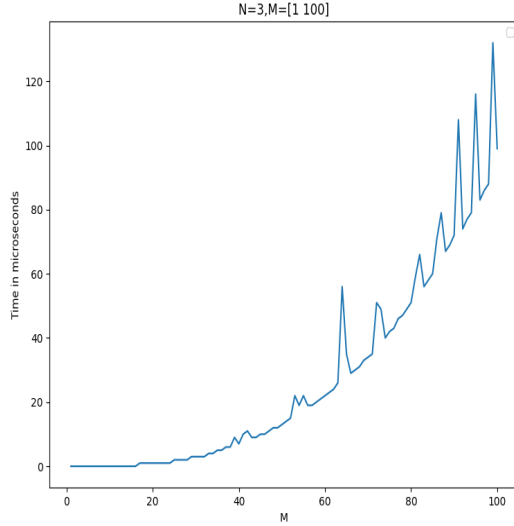
First we Fix $M=300$ and vary N from 1 to 7

N	Time
1	0
2	3
3	224
4	4493
5	34892
6	112558
7	246904



Now we Fix $n=4$ and vary M from 1 to 100

N	Time
1	0
16	0
30	3
40	7
50	13
60	22
100	99



From these two graphs we can conclude that complexity is exponential in 'n' and polynomial in 'm'

E. TIME COMPLEXITY

Worst case : $O(m^n)$

Best case : $\Omega(1)$ when $n=1$

III. ALGORITHM 2

A. ALGORITHM DESIGN

- 1) Take input of integers n and m in variables n and m respectively.
- 2) Call for the isZero function with parameters as n and m which will check whether the answer exists or not.
- 3) If $n \geq 60$ then return true as our algorithm works till long long int. Also if $m < 2$ to the power $n-1$ then we cannot have the sequence of length n having max value upto m so return true. And if the return value is true or it does not exist then we exit after printing 0.
- 4) Else using we create a vector of vectors mem of $n+1$ rows and $m+1$ columns and initializing it to -1.
- 5) Call the solve function with parameters n,m,mem which uses recursion with memoization (Dynamic Programming) to find the answer.
- 6) Base cases for this function is when the isZero function with n,m as parameters return true then return zero and also when n will become one then we will return m.

- 7) Check if the value at n,m in the mem vector is not -1 then we return mem[n][m] as answer for this n,m is stored in the previous call of the solve function on n,m.
- 8) As per the given condition the n-th value of the sequence can be at most m. There can be two cases for n-th element.
 - (i) If it is m, then the (n-1)th element is at most $m/2$. We recur for $m/2$ and n-1.
 - (ii) If it is not m, then it is at most m-1. We recur for (m-1) and n.
 So we recursive call for the solve function with parameters n,m-1,mem and n-1,m/2,mem and add the return values from both and store it in mem[n][m] so if further this n,m is called then we can return its value without computation as we have stored.
- 9) Final answer is returned from the solve function and displayed.

B. ALGORITHM

Input: N(length of sequence) and M(upper bound on sequence elements)

Output: Number of possible sequences of length n such that each of the next element is greater than or equal to twice of the previous element but less than or equal to m.

Method of isZero(n,m)

- 1) if $n \geq 60$ return false;
- 2) if $m < 2^{n-1}$ return false;

Method of solve(n,m)

- 1) Check if the answer is 0. If found 0 return.
- 2) if $n==1$ return m
- 3) if mem[n][m]!=-1 return mem[n][m]
- 4) Call solve for n,m-1 and n-1,m/2 and assign their sum in mem[n][m]

C. APRIORI ANALYSIS

Step	Time	FreqBest	Freq
Step1	1	1	1
Step2	1	1	1
Step3	1	1	$n*m$
Step4	1	1	$n*m$
Step5	1	0	$n*m$
Step6	2	0	$n*m-m$

For Best Case

$$T\Omega \propto 1*1+1*1+1*1+1*1+1*0+2*0$$

$$T\Omega \propto 4$$

Thus, the best case time complexity is $O(1)$ when $n=1$

For Worst Case

$$T\Omega \propto 1*1+1*1+1*(n*m)+1*(n*m)+1*(n*m)+1*(n*m-m)$$

$$T\Omega \propto 3+4*(n*m)-m$$

Clearly this equation will come out to be of the quadratic form .

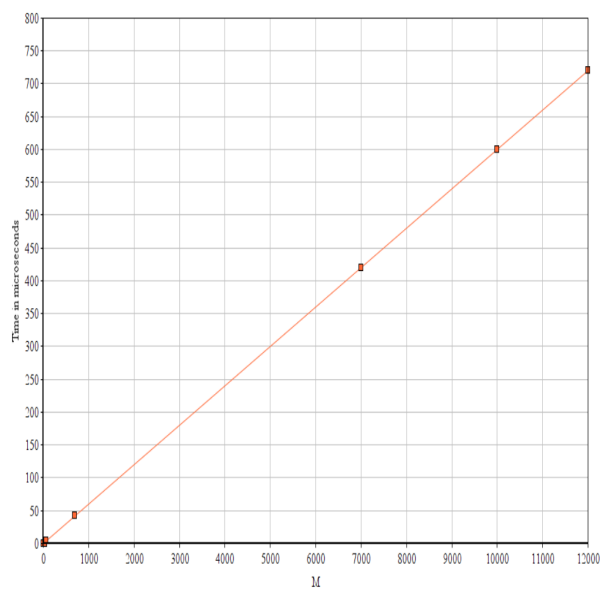
Thus , worst case time complexity is $O(n*m)$

Space complexity is $O(n*m)$

Time complexity –

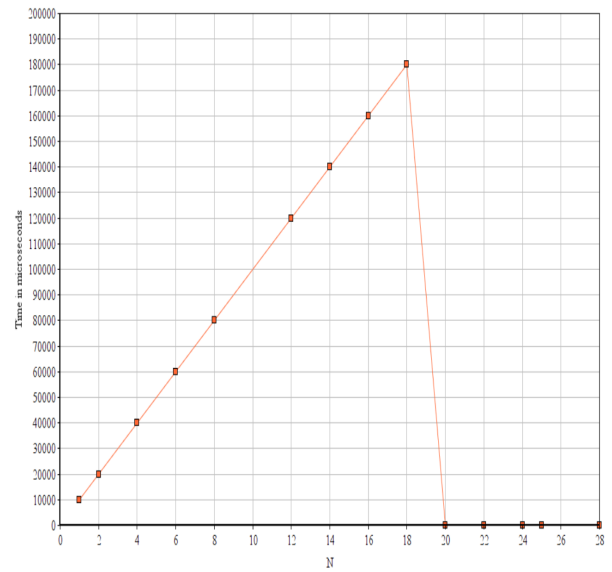
Fix $n=6$ and vary m

m	Time
1	0.01
70	4.2
700	42
7000	420
10000	600
12000	720



Fix $m=1000000$ and vary n

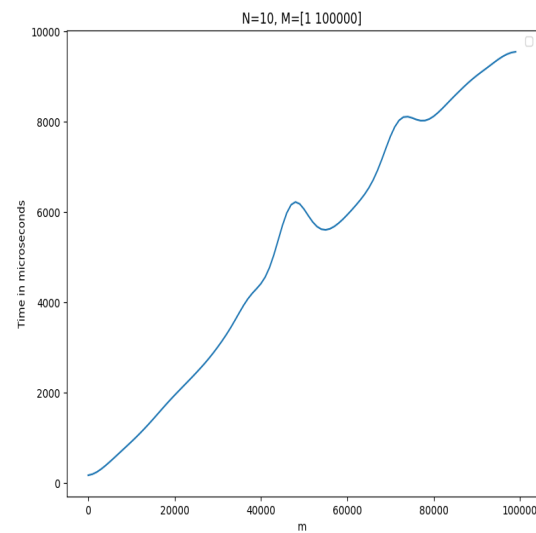
N	Time
1	10000
2	20000
4	40000
6	60000
8	80000
12	120000
14	140000
16	160000
18	180000
20	0.01
22	0.01
24	0.01
25	0.01
28	0.01



D. APOSTERIORI ANALYSIS

First we Fix $N=10$ and vary M from 1 to 100000

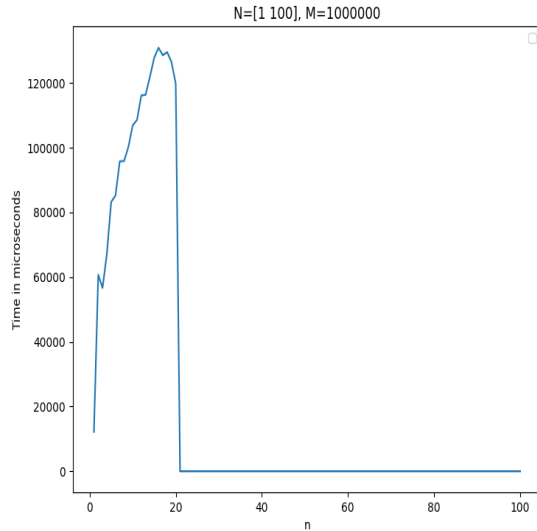
M	Time
1	0
2001	180
9001	224
43001	4282
78001	7857
88001	8701
99001	9704



From above graph we can conclude that complexity is linear in ' m '.

Now we Fix $m=1000000$ and vary n from 1 to 100

N	Time
1	12205
16	130887
30	0
40	0
50	0
60	0
100	0



In above graph complexity first increases linearly with 'n', but as 'n' increases the isZero() function calculates the answer in $O(1)$ and graph gets merged with the x-axis.

E. TIME COMPLEXITY

Worst case : $O(\min(n, \log_2(MAX_M)) * m)$

Best case : $\Omega(1)$ when answer is 0

CONCLUSION

Our second algorithm which we solved by divide and conquer is more efficient because unlike recursive solution it is storing the result of every function call and hence is not making same function calls multiple times. It has a time complexity and space complexity of $O(\min(n, \log_2(M)) * m)$.

REFERENCES

- [1] CP-Algorithms : Divide and Conquer Algorithm.
- [2] GFG : Sequences of given length where every element is more than or equal to twice of previous.