# Dynamic Programming | Set 36 (Maximum Product Cutting)

Given a rope of length n meters, cut the rope in different parts of integer lengths in a way that maximizes product of lengths of all parts. You must make at least one cut. Assume that the length of rope is more than 2 meters.

Examples:

```
Input: n = 2
Output: 1 (Maximum obtainable product is 1*1)

Input: n = 3
Output: 2 (Maximum obtainable product is 1*2)

Input: n = 4
```

```
Output: 4 (Maximum obtainable product is 2*2)

Input: n = 5
Output: 6 (Maximum obtainable product is 2*3)

Input: n = 10
Output: 36 (Maximum obtainable product is 3*3*4)
```

**1) Optimal Substructure:**
This problem is similar to Rod Cutting Problem. We can get the maximum product by making a cut at different positions and comparing the values obtained after a cut. We can recursively call the same function for a piece obtained after a cut.

Let maxProd(n) be the maximum product for a rope of length n. maxProd(n) can be written as following.

maxProd(n) = max(i*(n-i), maxProdRec(n-i)*i) for all i in {1, 2, 3 .. n}

**2) Overlapping Subproblems**
Following is simple recursive C++ implementation of the problem. The implementation simply follows the recursive structure mentioned above.

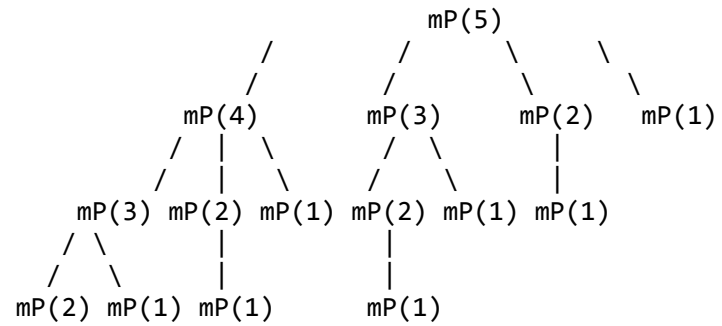
```
// A Naive Recursive method to find maxium product
#include <iostream>
using namespace std;

// Utility function to get the maximum of two and three integers
int max(int a, int b) { return (a > b)? a : b;}
int max(int a, int b, int c) { return max(a, max(b, c));}

// The main function that returns maximum product obtainable
// from a rope of length n
int maxProd(int n)
{
    // Base cases
    if (n == 0 || n == 1) return 0;

    // Make a cut at different places and take the maximum of all
    int max_val = 0;
    for (int i = 1; i < n; i++)
      max_val = max(max_val, i*(n-i), maxProd(n-i)*i);

    // Return the maximum of all values
    return max_val;
}

/* Driver program to test above functions */
int main()
{
    cout << "Maximum Product is " << maxProd(10);
    return 0;
}
```

Output:

```
Maximum Product is 36
```

Considering the above implementation, following is recursion tree for a Rope of length 5.

```
mP() ---> maxProd()
```




In the above partial recursion tree, mP(3) is being solved twice. We can see that there are many subproblems which are solved again and again. Since same suproblems are called again, this problem has Overlapping Subprolems property. So the problem has both properties (see this and this) of a dynamic programming problem. Like other typical Dynamic Programming(DP) problems, recomputations of same subproblems can be avoided by constructing a temporary array val[] in bottom up manner.

```
// A Dynamic Programming solution for Max Product Problem
int maxProd(int n)
{
   int val[n+1];
   val[0] = val[1] = 0;

   // Build the table val[] in bottom up manner and return
   // the last entry from the table
   for (int i = 1; i <= n; i++)
   {
      int max_val = 0;
      for (int j = 1; j <= i/2; j++)
         max_val = max(max_val, (i-j)*j, j*val[i-j]);
      val[i] = max_val;
   }
   return val[n];
}
```

Time Complexity of the Dynamic Programming solution is O(n^2) and it requires O(n) extra space.

**A Tricky Solution:**
If we see some examples of this problems, we can easily observe following pattern.
The maximum product can be obtained be repeatedly cutting parts of size 3 while size is greater than 4, keeping the last part as size of 2 or 3 or 4. For example, n = 10, the maximum product is obtained by 3, 3, 4. For n = 11, the maximum product is obtained by 3, 3, 3, 2. Following is C++ implementation of this approach.

```
#include <iostream>
using namespace std;

/* The main function that teturns the max possible product */
int maxProd(int n)
```

```
{
   // n equals to 2 or 3 must be handled explicitly
   if (n == 2 || n == 3) return (n-1);

   // Keep removing parts of size 3 while n is greater than 4
   int res = 1;
   while (n > 4)
   {
       n -= 3;
       res *= 3; // Keep multiplying 3 to res
   }
   return (n * res); // The last part multiplied by previous parts
}

/* Driver program to test above functions */
int main()
{
    cout << "Maximum Product is " << maxProd(10);
    return 0;
}
```

Output:

```
Maximum Product is 36
```

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

## Related Topics:

- Linearity of Expectation
- Iterative Tower of Hanoi
- Count possible ways to construct buildings
- Build Lowest Number by Removing n digits from a given number
- Set Cover Problem | Set 1 (Greedy Approximate Algorithm)
- Find number of days between two given dates
- How to print maximum number of A's using given four keys
- Write an iterative O(Log y) function for pow(x, y)

Tags: Dynamic Programming



**Writing code in comment?** Please use **ideone.com** and share the link here.

**Rahul Singh** • 11 days ago

tricky soln and dynamic soln produce different output for n = 59

• Reply • Share ›

**Chao Zhou** • 6 months ago

If the power() function is implemented as O(1) in the language, there will be an O(1) solution.

The idea is based on the observation that since there isn't a component > 3 appear in the final production, we can calculate the number of 3 in n and deal with different remainder.

Here is the code in python:
http://ideone.com/E2PDK6

1 • Reply • Share ›

**manoj** • 6 months ago

http://ideone.com/EVXbad
please check this code using DP in O(n)

• Reply • Share ›

**aa1992** • 7 months ago

check out this code for max product.

let me know if any case fails.

http://ideone.com/oqCVUs

• Reply • Share ›

**Neel** • 8 months ago

This is the modified version of tricky solution and it works .

```
public static int maxProductAfterCutting_solution2(int length) {
if(length < 2) {
return 0;
}
if(length == 2) {
return 1;
}
if(length == 3) {
return 2;
}

int timesOf3 = length / 3;
if((length - timesOf3 * 3) % 2 == 1) {
timesOf3 -= 1;
}
```

}

int timesOf2 = (length - timesOf3 * 3) / 2;

return (int)(Math.pow(3, timesOf3)) * (int)(Math.pow(2, timesOf2));
}

• Reply • Share ›

**dev21** • 9 months ago

tabulated method for this problem

http://ideone.com/67qoLp

• Reply • Share ›

**caot** • 9 months ago

long maxProd(int n) throws Exception {
..if (n < 2)
....throw new Exception("invalid: " + n);

..if (n == 2 || n == 3)
....return n - 1;

..int p = n / 3;
..int m = n % 3;
..if (m < 2) {
....p = p - 1;
....m += 3;
..}
..long res = (long) Math.pow(3, p);

..return res * m;
}

Complexity: O(1)

• Reply • Share ›

**AlienOnEarth** • a year ago

Can someone please provide proof of correctness for the last "tricky" method. For me somehow, its not intuitive.

4 • Reply • Share ›

**vitus udemba** ➚ AlienOnEarth • 6 months ago

Please Alien can you help me with this question i only have 10 hours.

Write a
program that uses pointer in functions min

and max to find the minimum and
maximum respectively of a 2-D array elements and display their array locations.

Hint:

4

6

3

0

7

2

---

see more

• Reply • Share ›

**ThrustVectoring** AlienOnEarth • 9 months ago

It's a little more clear if you cut the rope of length 4 into two ropes of length 2.

If you have any rope of length 5 or above, maxCut(N) > N, so you want to cut it into pieces.

If you can have three ropes of length 2, or two ropes of length 3, you'd rather have two ropes of length 3, becuase 2^3 < 3^2.

So, we know that the solution can't have any ropes of length 4 or more, because we cut them. We also know that the solution can't have more than two ropes of length 2, because we'd rather have 6 units of length in 3-length chunks. So whatever the solution is, it has either 0, 1, or 2 pieces of rope of length 2, and any number of pieces of rope of length 3.

1 • Reply • Share ›

**Rupesh gautam** • a year ago

```
if (n == 0 ) return 0;
if (n == 2 ) return 2;

// Make a cut at different places and take the maximum of all
int max_val = 0;
for (int i = 1; i < n; i++)
max_val = max(max_val, maxProd(n-i)*i);
```

this will do the trick. we don't need to get the max of i*(n-i) etc..

just because Max_product will always be equal to i * Max_product_of_remaining_length by the

property of Maximum.

• Reply • Share ›

**gaurav mutreja** • a year ago

max_val = max(max_val, i*(n-i), maxProd(n-i)*i);
Why not max_val = max(max_val, i*(n-i), maxProd(n-i)*maxProd(i));

2 • Reply • Share ›

**sujeet singh** • a year ago

this problem can be solved by maximaize & minimize of calculus = for any n the max product can be easily achieved of formula => if n is even it will be n/2 if n is odd n+1/2 n-1/2..we can see number is odd and even and decide accordingly. this is for one cut we have to do .. missed that we can have multiple cut too ..
it can be further extended - let say for n we get max product at n/2 or n+1/2 depending upon even and odd .again we have to do the same thing for n/2 or n+1/2 until we have 1 .

2 • Reply • Share ›

**Abhinav Choudhury** • a year ago

If we need to make atleast one cut, shouldn't the recursive logic be:
maxProd(n) = max(i*(n-i), maxProdRec(n-i)*i) for all i in {1, 2, 3 .. n-1}

3 • Reply • Share ›

**geek** • a year ago

In second solution
max_val = max(max_val, i*(n-i), maxProd(n-i)*i);

If we cut at any point then maxproduct(to cut at that point) will be from below 4 cases :
1) i*(n-i)
2)maxProd(n-i)*i
3)maxProd(i)*(n-i)
4) maxProd(i)*maxProd(n-i)

Here we know 3rd case will be covered when i iterates till end to take value as n-i. But we are not considering 4th case.

Please someone explain me this logic.

10 • Reply • Share ›

**v2v4** • a year ago

O(1) solution
if something is wrong please correct

```
#include <iostream>
#include <cmath>
```

```
#define ll long long

using namespace std;

ll maxproduct(ll n)
{
if(n==0 || n==1)
return 0;

if(n==2)
return 1;

if(n==3)
return 2;
```

see more

• Reply • Share ›

**krishna** v2v4 • 10 months ago

could please explain.. how you derived these formulas or could you give reference

• Reply • Share ›

**v2v4** krishna • 10 months ago

as jeffa told..we optimize to reach at finding maximum no. of 3 length cuts.
now as above tricky solution is given..i just suggest an alternative way of doing that -> instead of each time subtracting and multiplying by 3 till it is greater than 4, i found that no. directly by taking modulus.
for eg. if len=14 .. after subtracting it till it is greater than 4, we reach res=4, similarly i m getting this no. directly by taking (14-2)/3.
if u compare my solution to above solution mentioned..its almost same.

• Reply • Share ›

**jeffa** krishna • 10 months ago

the proof of method is using basic calculus to
optimize the function f(m) = (n/m)^m, where m is the number of pieces. A simple symmetry argument shows that each piece should be the same length. Using logarithmic differentiation you find that f is maximized when m=n/e. This means that the length of each piece is should be n/m=e~2.71. Since we must use integral lengths, monotonicity of the function f then implies that the optimal solution occurs when n/m =3 as much as possible.

• Reply • Share ›

**Ronny** v2v4 • a year ago

**@v2v4** I guess the solution is correct. But your analysis of complexity isn't. power

function doesn't run in O(1). However by implementing it optimally we can do it in O( lg n ).

1 • Reply • Share ›

**v2v4** ➚ Ronny • a year ago

**@Ronny**

yes u r right
we can find power exponentially in O(lg n)
thanx for ur reply :)

• Reply • Share ›

**mojo** • a year ago

```
public int maxProduct3(int length) {

int[] memo = new int[length + 1];

Arrays.fill(memo, -1);

memo[0] = 0;

memo[1] = 0;

for (int i = 2; i <= length; ++i) {

int max = 1;

for (int k = 1; k < i; ++k) {

int localMax = max(k * (i - k), memo[k] * memo[i - k]);

if (localMax > max) max = localMax;

}

memo[i] = max;

}

return memo[length];

}
```

• Reply • Share ›

**Kaushik Srinivasan** • a year ago

Is there a proof for the tricky solution?

2 • Reply • Share ›

**Chetna Gupta** • a year ago

For the tricky solution
Answer returned for 8 would be 3*3*2 = 12 while 4*4 = 16. This would not work in general for many cases. Correct me if i am wrong

• Reply • Share ›

**foolish** ➚ Chetna Gupta • a year ago

3*3*2 is not 12... lol it's 18
and it is greater than 12 and 16 :D :D
at least run before u comment!

7 • Reply • Share ›

**edu** ➚ foolish • 8 months ago

Answer returned for 16 would be 3*3*3*3*3*1 = 243 while 4*4*4*4 = 256. This would not
work in general for many cases. Correct me if i am wrong

• Reply • Share ›

**Rfun** ➚ Chetna Gupta • a year ago

the answer for 8 in the tricky solution would be 2*2*2*2=16!

• Reply • Share ›

**zain** ➚ Rfun • a year ago

first understand the logic of the tricky solution

2 • Reply • Share ›

**Chetna Gupta is not my name** ➚ Chetna Gupta • a year ago

Can you please explain how "3*3*2=12" ? I mean in which world is that statement even true?

7 • Reply • Share ›

**Guest** • 2 years ago

The recursive solution fails for values as the product goes on increasing exponentially. for eg 10 will 9*1 then 8*2 then 7*3 that means we are recursively calling maxProd(16) then max_prod(21) which in turn will increase. Please correct me if I am mistaken,

• Reply • Share ›

**Kaushik Srinivasan** ➚ Guest • a year ago

Check the while loop. It is while(n > 4). Which means the algo will return 3*3*4=36.

• Reply • Share ›

**Guest2** ➚ Guest • a year ago

It's maxProd(n-i) * i, not maxProd((n-i)*i)

• Reply • Share ›

**racks** • 2 years ago
What is the Time complexity of the Tricky solution
• Reply • Share ›

**pavansrinivas** racks • a year ago
O(n/3) which is nothing but O(n)
• Reply • Share ›

**Guest** racks • a year ago
Theoretically , time complexity is O(n).....actually ,it will be O(n/3).
• Reply • Share ›

**mstepan** • 2 years ago
What is the result for N = 153. From your code I receive 2136675824, but should be 2145130834. Correct if I'm not right.
1 • Reply • Share ›

**Vikram Ojha** • 2 years ago
Another approach to solve dis prob with dynamic programming

```
#include<stdio.h>
#include<conio.h>

int max(int a,int b){ return(a>b?a:b);}
int max(int a,int b,int c){ return max(a,max(b,c));}

int *mem=new int[15];

int no_of_cal=1;

int maxProDY(int n,int mem[]);

void main()
{

for(int i=0;i<=15;i++)
{
mem[i]=-1;
```

see more

1 • Reply • Share ›

**Deepak** • 2 years ago
long int prod1;
prod1=pow(3,(n/3))*((n%3)?(n%3):1);

```
prod1=pow(3,(n/3)-1)*((n%3)?(n%3):1);
std::queue<int> myqueue;
int a= n/2;
int b= n-a;
myqueue.push(a);
myqueue.push(b);
long int prod=1;
while(myqueue.empty()==false)
{
a=myqueue.front();
myqueue.pop();
if(a<=3)
prod*=a;
else
{
n=a;
a=n/2;
```

see more

1 ▲ ▼ • Reply • Share ›

**sanjay** • 2 years ago

```
// A Dynamic Programming solution for Max Product Problem
int maxProd(int n)
{
int val[n+1];
val[0] = val[1] = 0;

// Build the table val[] in bottom up manner and return
// the last entry from the table
for (int i = 1; i <= n; i++)
{
int max_val = 0;
for (int j = 2; j <= i/2; j++)
max_val = max(max_val, (i-j)*j, j*val[i-j]);
val[i] = max_val;
}
return val[n];
}
```

In second for loop j should be intialize with 1 rather than 2, then only it will produce result for n=2, result is 1;

2 ▲ ▼ • Reply • Share ›

**GeeksforGeeks** ➦ sanjay • 2 years ago

Thanks for pointing this out. We have updated the code.

• Reply • Share ›

**Amit** ➚ GeeksforGeeks • 2 years ago

You Have not updated !!!

• Reply • Share ›

**DRAGONWARRIOR** • 2 years ago

In A Tricky Solution
wats the output if n==4

• Reply • Share ›

**sanjay** ➚ DRAGONWARRIOR • 2 years ago

ans is 4
return res*n;
here res=1 and n=4
so result is 4

• Reply • Share ›

**Sabelan** • 2 years ago

Proof of the Third Method:

Consider cutting a section of the rope with length greater than or equal to 2.
Let that section's length be N, where N>=2.
That section COULD be cut as N-2 and another section of length 2, which would contribute as follows, to the product sum:

(N-2)*2

(N-2)*2 >= N, as long as N>=4. That is to say ANY rope section of length greater than or equal to 4 will contribute more, or equal, to the product sum by cutting that section into 2 sections, one of length 2. This is true for all integers except 2 and 3.

Thus we know that ALL of the rope sections can be optimally written as a partition of 2's and 3's, so long as N>=4.

Now consider any rope of length 6, the LCM of 2 and 3.

This segment of length 6 can be either:
2*2*2 OR 3*2

see more

4 • Reply • Share ›

**Souradip** • 2 years ago

The proof:

http://math.stackexchange.com/...

2 • Reply • Share ›

**GeeksFollower** • 2 years ago

O(1) solution:

```
double maxProd_mySoln(int n)
{
    if(n==2)
    return 1;

    if(n==3)
    return 2;

    if(n%3==1)
    {
        return pow(3.0,(n-4)/3)*4;
    }

    return n%3==0? pow(3.0,n/3): pow(3.0,n/3)*2;

}
```

see more

• Reply • Share ›

**amit** ➚ GeeksFollower • 2 years ago

this is wrong .. check for 100??

• Reply • Share ›

**ERYOYO** ➚ GeeksFollower • 2 years ago

you have missed the case when n==0 or n==1. In both cases, answer should be zero but your code is gonna to give some another answer rather than this one....???

• Reply • Share ›

**eric** • 2 years ago

Is there any proof of last solution (repeatedly cutting parts of size 3)?

• Reply • Share ›

**rahul23** • 2 years ago

@GFG

Can you plz give some explanation of 3rd method.Although it is working for every case,but how can we think of this kind of approach?I mean give some explanation that what is the

mathematical logic behind this that the pattern is coming like this.

thnx

• Reply • Share ›

Load more comments

- Greedy Algorithms
- Backtracking
- Pattern Searching
- Divide & Conquer
- Mathematical Algorithms
- Recursion
- Geometric Algorithms

# Popular Posts

- All permutations of a given string
- Memory Layout of C Programs
- Understanding “extern” keyword in C
- Median of two sorted arrays
- Tree traversal without recursion and without stack!
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Sorted Linked List to Balanced BST

Follow @GeeksforGeeks

# Recent Comments

- lt_k

  i need help for coding this function in java...

  Java Programming Language · 2 hours ago

- Piyush

  What is the purpose of else if (recStack[*i])...

  Detect Cycle in a Directed Graph · 2 hours ago

- Andy Toh

  My compile-time solution, which agrees with the...

  Dynamic Programming | Set 16 (Floyd Warshall Algorithm) · 2 hours ago

- lucy

  because we first fill zero in first col and...

  Dynamic Programming | Set 29 (Longest Common Substring) · 2 hours ago

- lucy

@GeeksforGeeks i don't n know what is this long...

Dynamic Programming | Set 28 (Minimum insertions to form a palindrome) · 3 hours ago

- manish

  Because TAN is not a subsequence of RANT. ANT...

  Given two strings, find if first string is a subsequence of second · 3 hours ago