
REDEFINING
LIMITATIONS
SOFTWARE INCUBATOR
Presents
SINTAX
A Competitive Programming Workshop

BIG FACTORIAL

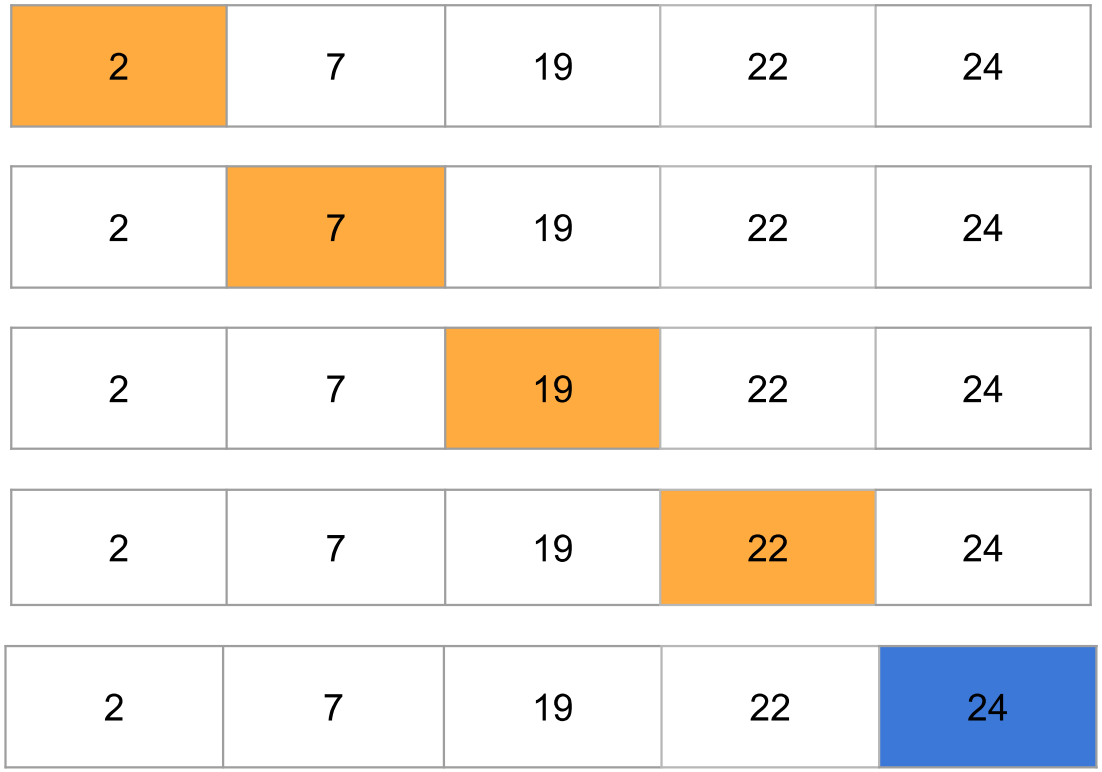
Q.) Find the Factorial of a number .

Constraint : 0 <= N <= 10^5

Linear Search

A linear search, also known as a sequential search, is an algorithm used in computer science to locate a specified value (key) within an array. It sequentially checks each element of the array until a match is found or the whole array has been traversed.

To find element 24

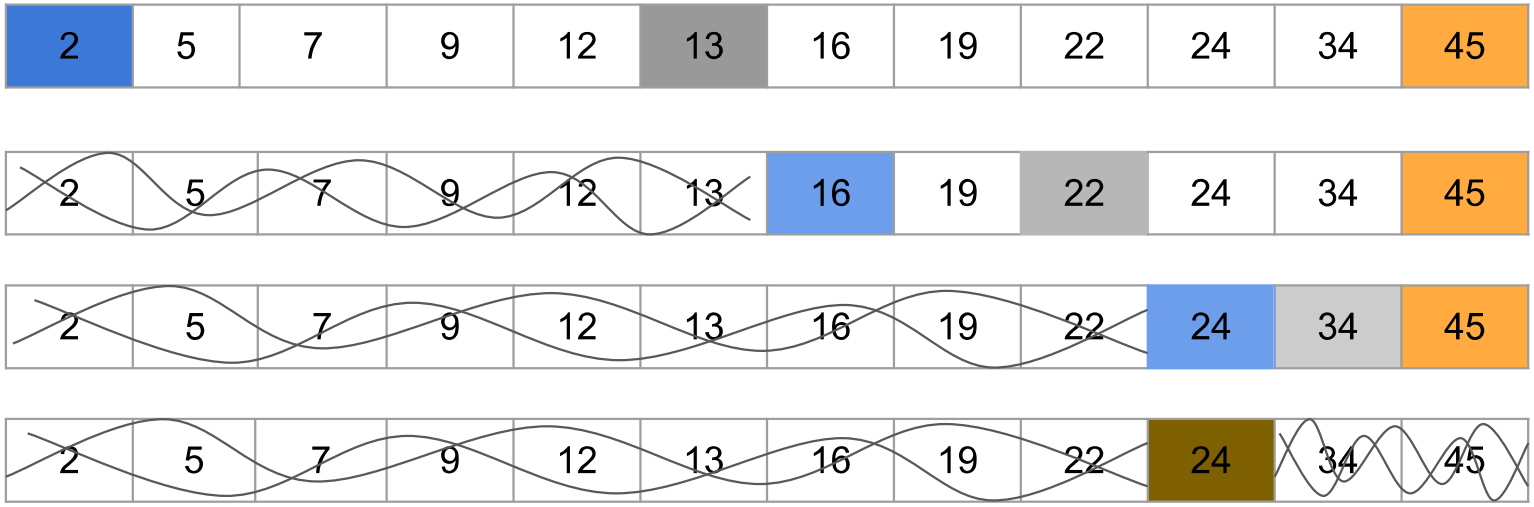



WHICH SEARCH DID YOU USE
LINEAR SERACH
WHAT DID IT COST?
TIME COMPLEXITY

Binary Search

A binary search, also known as a half-interval search, is an algorithm used in computer science to locate a specified value (key) within an array. For the search to be binary, the array must be sorted in either ascending or descending order.

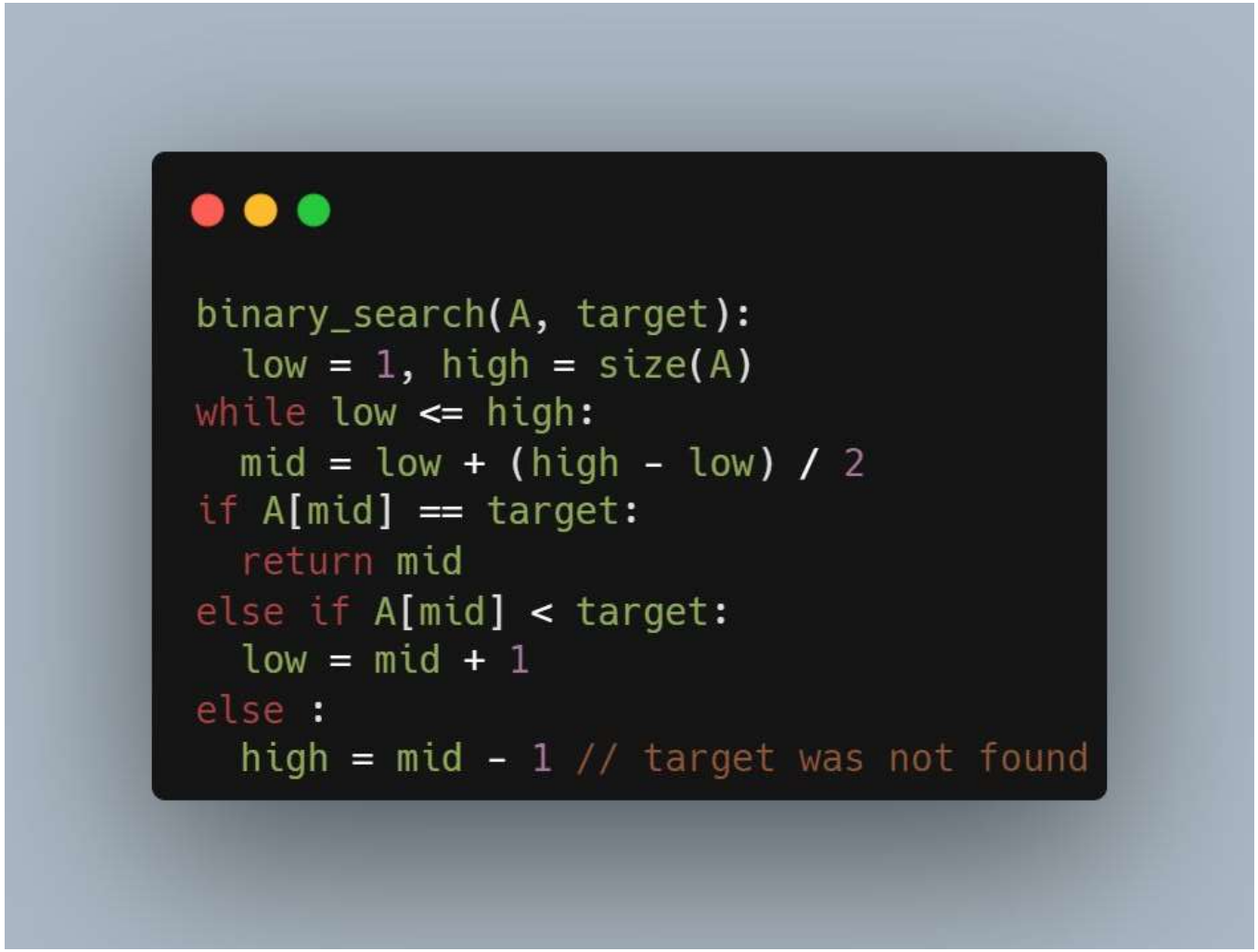
To find element 24



Element is found at index 9

https://ideone.com/1f5ocT

Linear Search in Sorted Array


binary_search(A, target):
low = 1, high = size(A)
while low <= high:
mid = low + (high - low) / 2
if A[mid] == target:
return mid
else if A[mid] < target:
low = mid + 1
else :
high = mid - 1 // target was not found

To find square root of a number

If perfect square:

    Then find the square root

Else:

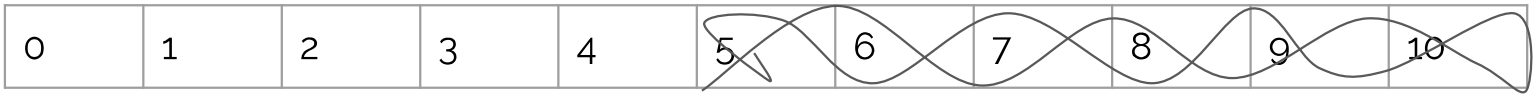
    Find the integer part of square root.

s mid e

while(s<=e)

$mid = \frac{0+10}{2} = 5$ $5^2 = 25$

if(mid^2 <10)
ans=mid---->5



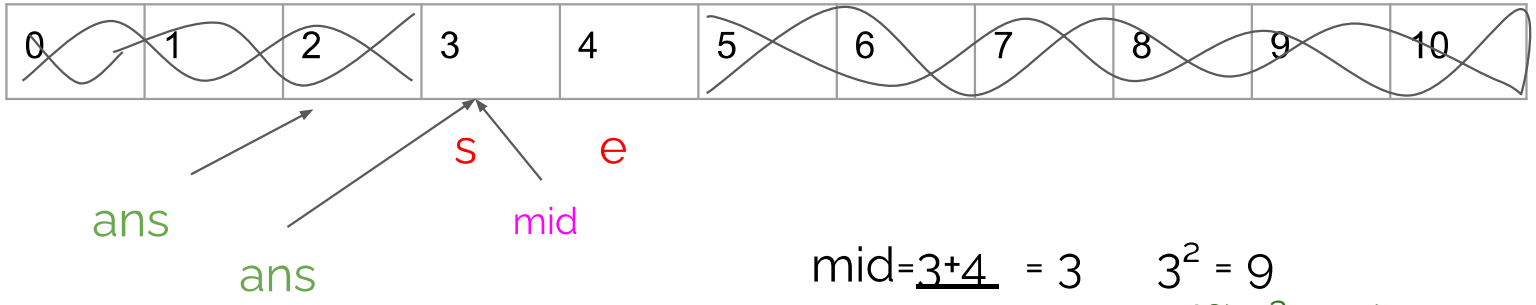
s mid e

ans

$mid = \frac{0+4}{2} = 2$ $2^2 = 4$

while(s<=e)

if(mid^2 <=10)
ans=mid----->2

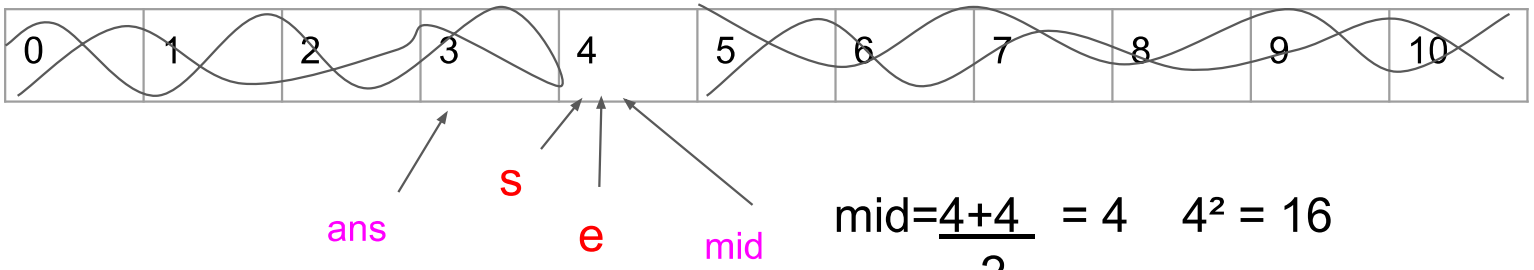


$mid = \frac{3+4}{2} = 3$ $3^2 = 9$

if(3^2 <10)

ans=3

while(s<=e)

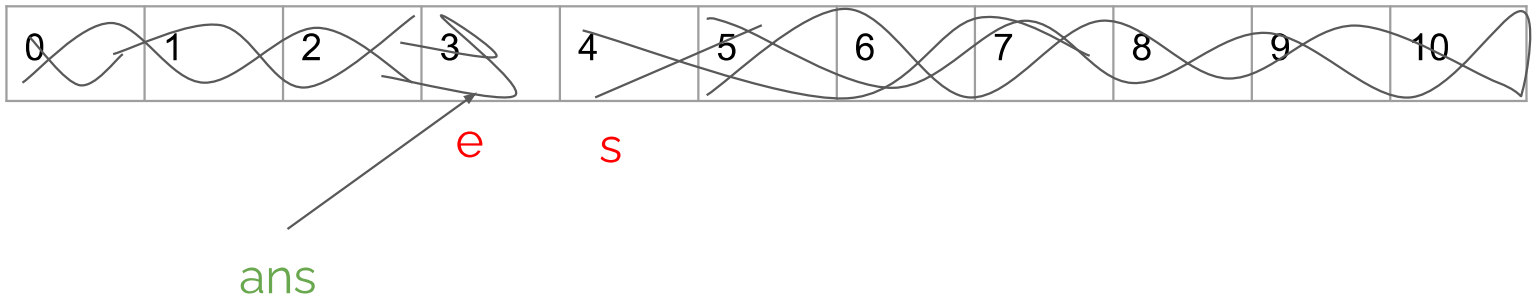


$mid = \frac{4+4}{2} = 4$ $4^2 = 16$

if(4^2 <10)

ans=4

while(s<=e)

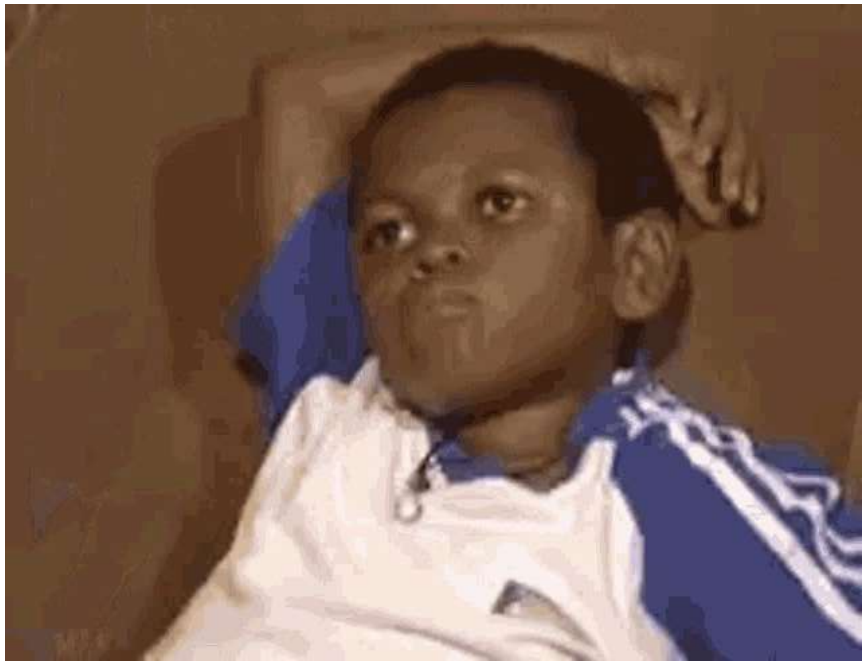


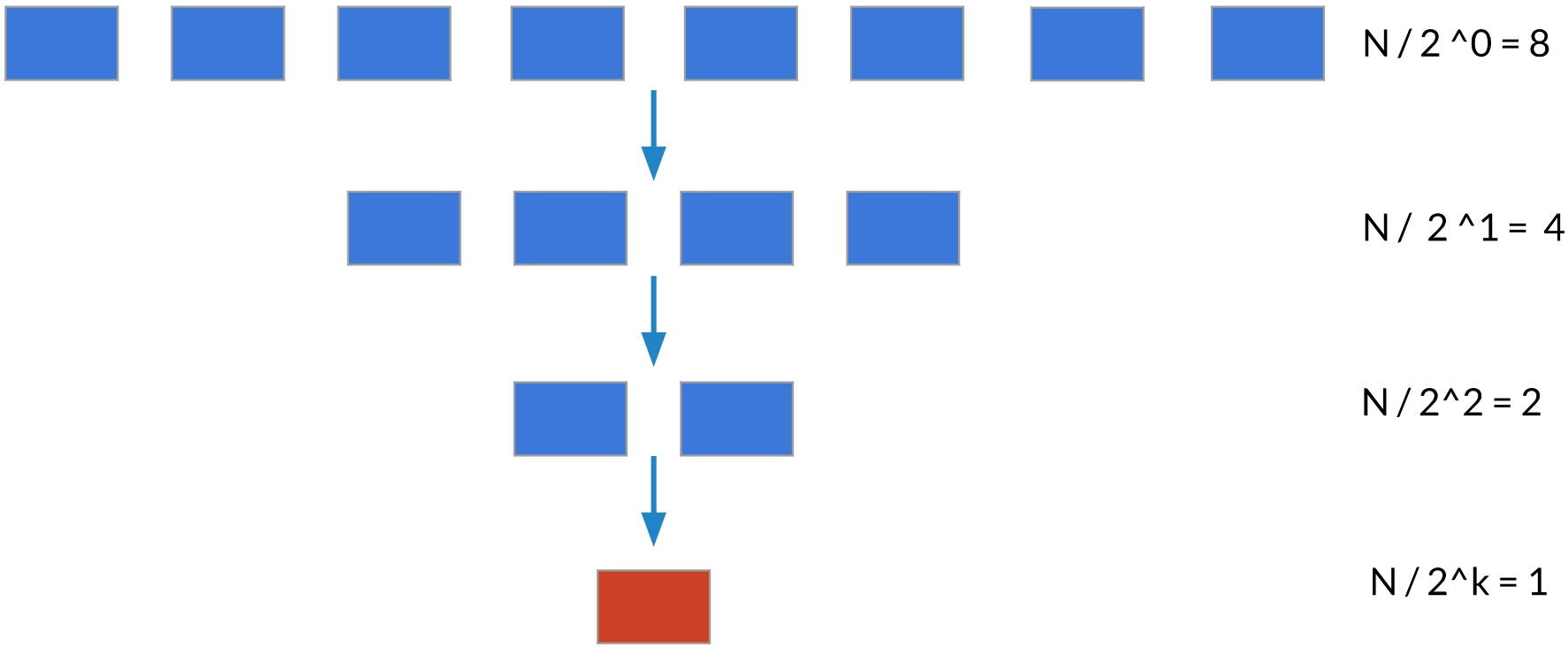
while(s<=e)

Final answer= 3

https://ideone.com/EAP2Lg

Complexity of Binary Search ????




N / 2 ^0 = 8
N / 2 ^1 = 4
N / 2^2 = 2
N / 2^k = 1

So, In worst case we take K steps to reduce array size to 1

$$N / 2^k = 1$$

$$N = 2^k$$

$$\log_2(N) = \log_2(2^k)$$

$$\log_2(N) = k$$

Aggressive cows

Problem Statement:
https://www.spoj.com/problems/AGGRCOW/

Solution:
https://ideone.com/IsMSls

Bit Magic

Bit Manipulation

Bit manipulation is the process of manipulating bit of a data.
Eg : 7 can be represented as 111 (in binary)

$$1\ 1\ 1 = 2^2 + 2^1 + 2^0 = 7$$

AND

5 & 6 = 4

101 & 110 = 100

Tricks

- a & a = a
- a & 1 = a % 2
- a & 0 = 0

X	Y	AND
0	0	0
0	1	0
1	0	0
1	1	1

OR

5 | 6 = 7
101 | 110 = 111

Tricks

- b | b = b
- a | 0 = a

X	Y	OR
0	0	0
0	1	1
1	0	1
1	1	1

NOT

~ 5 = 2
~ 101 = 010

~ 2 = 5
~ 010 = 101

X	~X
0	1
1	0

Tricks

- ~(~a) = a

XOR

5 ^ 6

101 ^ 110 = 011 => $(011)_2 = 3$

5 ^ 6 = 3

Tricks

- Exist in triplet
  - 5 ^ 6 = 3
  - 5 ^ 3 = 6
  - 3^ 6 = 5
- b ^ b = 0
- b ^ 0 = b

X	Y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

QUESTION

Given an array, every element in the array occurs twice except one. Find that number.

https://ideone.com/TT9akS

Left shift

7 << 2

$7 = (111)_2$

11100

$(11100)_2 = 28$

7 << 2 = 28

Tricks

Multiplication with powers of 2

$a << 1 = 2 * a$

$a << 4 = 16 * a$

$a << k = 2^k * a$

Right shift

7 >> 2

$7 = (111)_2$

~~$(00111)_2 = 7$~~

001

$(001)_2 = 1$

7 >> 2 = 1

Tricks

Integer division with powers of 2

a >>1 = a / 2

a >> 4 = a / 16

a >> k = a / 2^k

QUESTION

Given a number represent it as the sum of power of 2.

Eg : 8 = 2 + 2 + 2 + 2
or 4 + 4
or 4 + 2 + 2
or 8

https://ideone.com/xrxYLR

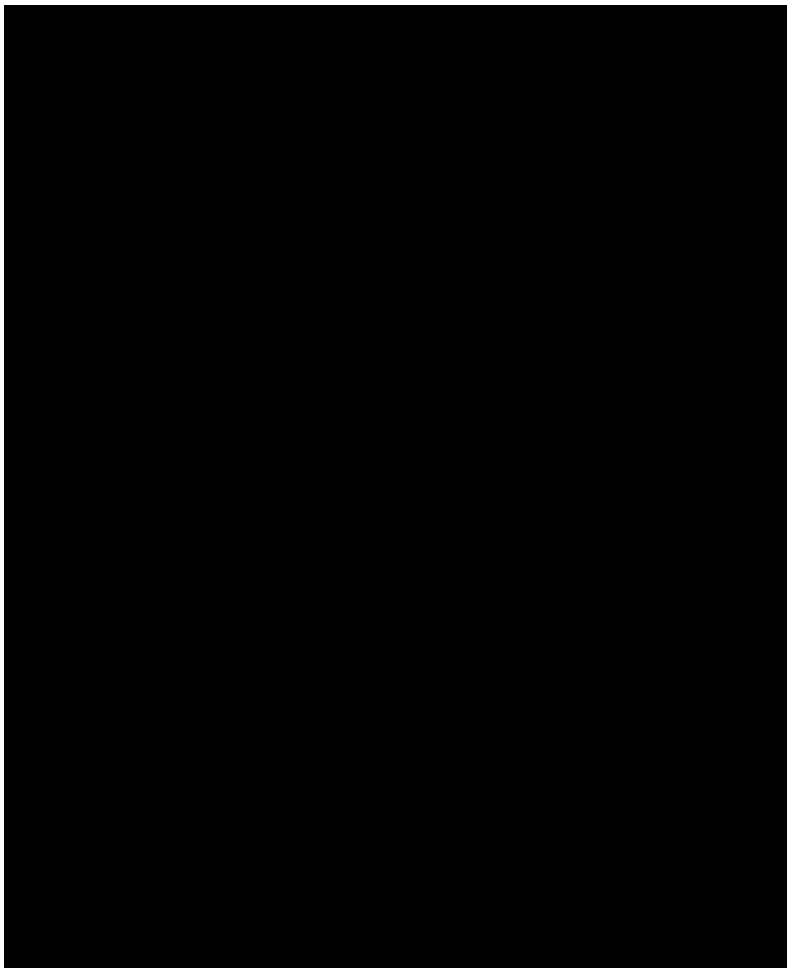
RECURSION

Recursion

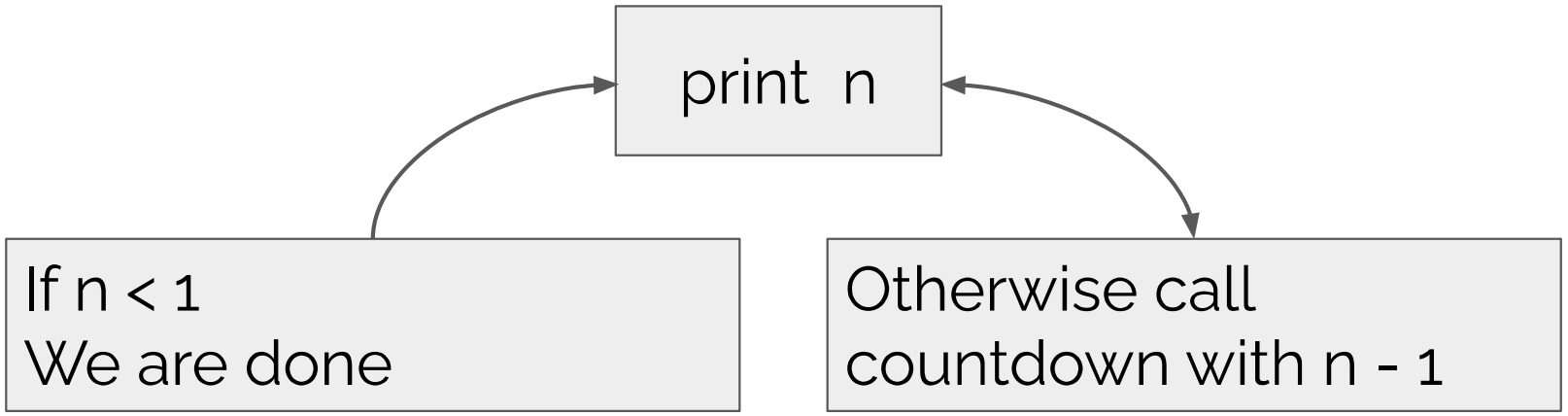
The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called a recursive function.

Example:-

```
void function()
{
    function();
}
```



```
void fun()
{
    // base condition
    if(condition)
        return;

    // recursive call
    fun();
}
```



https://ideone.com/MH9cnh

Factorial

$5! = 5 * 4 * 3 * 2 * 1$
$4! = 4 * 3 * 2 * 1$
$5! = 5 * 4!$

$n! = n * (n-1)!$

Print factorial in C++ :-

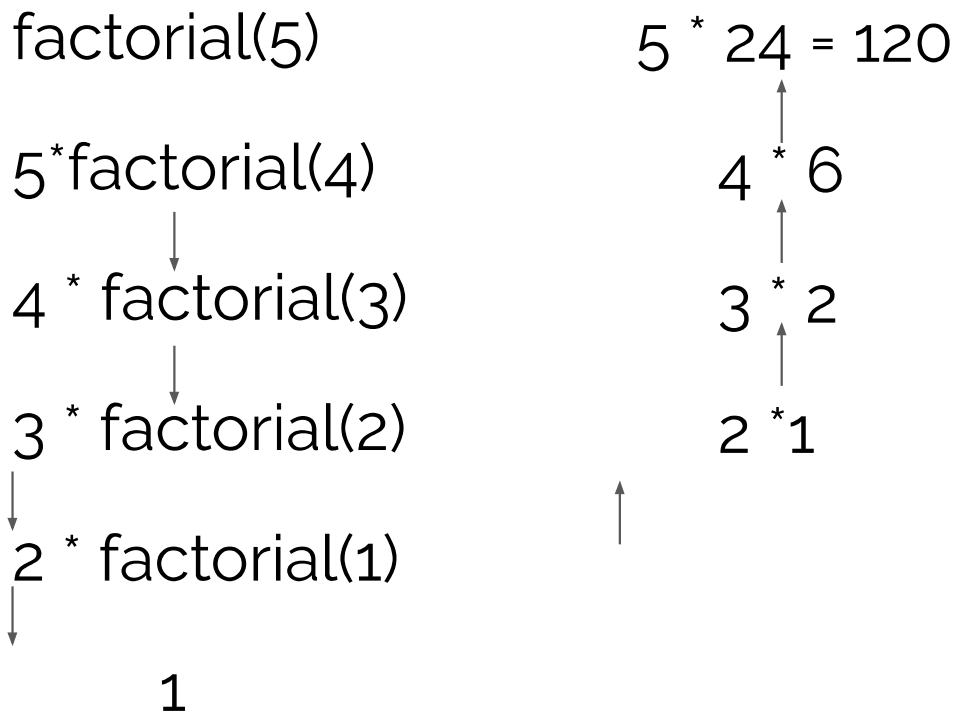
```
#include<bits/stdc++.h>
using namespace std;
int factorial(int n) {
   if (n==1)              // base condition
         return 1;
      else
         return n*factorial(n-1);   // recursive case
}
int main() {
   int n;
   cin>>n;
   int result = factorial(n);
   cout<<result<<"\n";
   return 0;
}
```

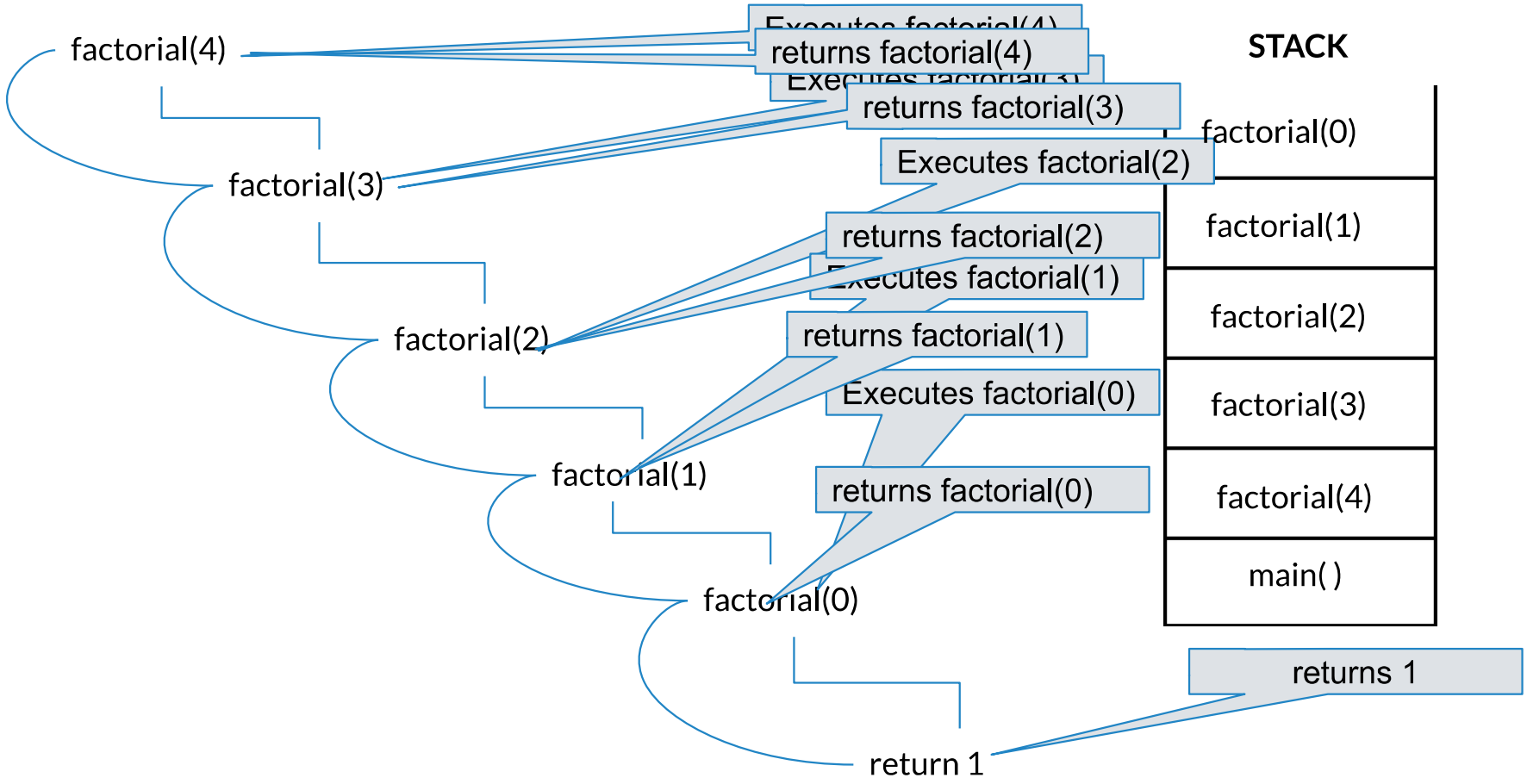
INPUT

5

OUTPUT

120


factorial(5)
5*factorial(4)
4 * factorial(3)
3 * factorial(2)
2 * factorial(1)
1
5 * 24 = 120
4 * 6
3 * 2
2 *1


factorial(4)
returns factorial(4)
returns factorial(3)
factorial(3)
Executes factorial(2)
returns factorial(2)
factorial(2)
returns factorial(1)
Executes factorial(0)
factorial(1)
returns factorial(0)
factorial(0)
return 1
returns 1
STACK
factorial(0)
factorial(1)
factorial(2)
factorial(3)
factorial(4)
main( )

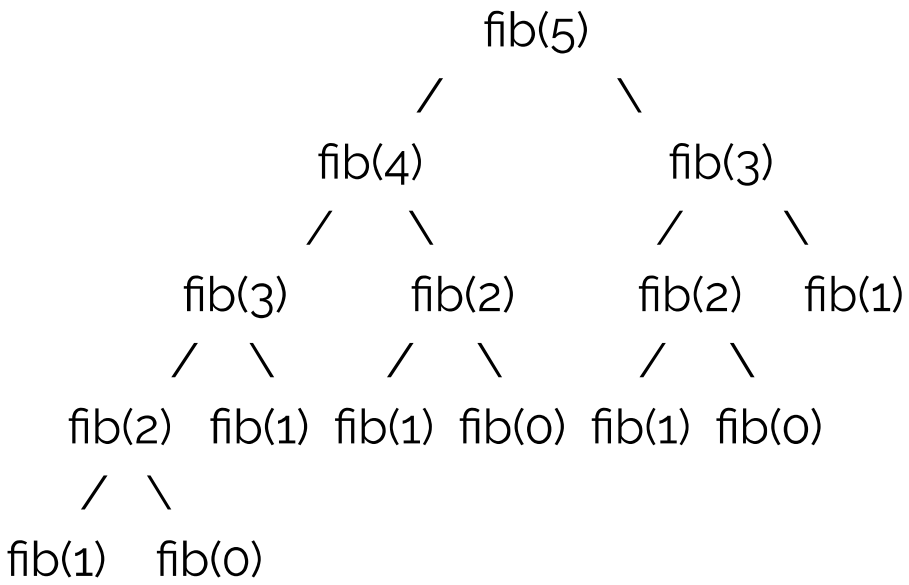
Fibonacci

0th = 0
1st = 1
2nd = 1st + 0th = 1 + 0 = 1

nth = (n-1)th + (n-2)th

=> 0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 . . . .

Recursion Tree



Print nth fibonacci number in C++ :-

```
#include<bits/stdc++.h>
using namespace std;
int fib(int n) {
   if(n==0)    return 0;
   else if(n==1)      return 1;
   else        return fib(n-1)+fib(n-2);
}

int main() {
int n;
cin >> n;
int result = fib(n);
cout<<result<<"\n";
return 0;
}
```

INPUT

5

OUTPUT

5

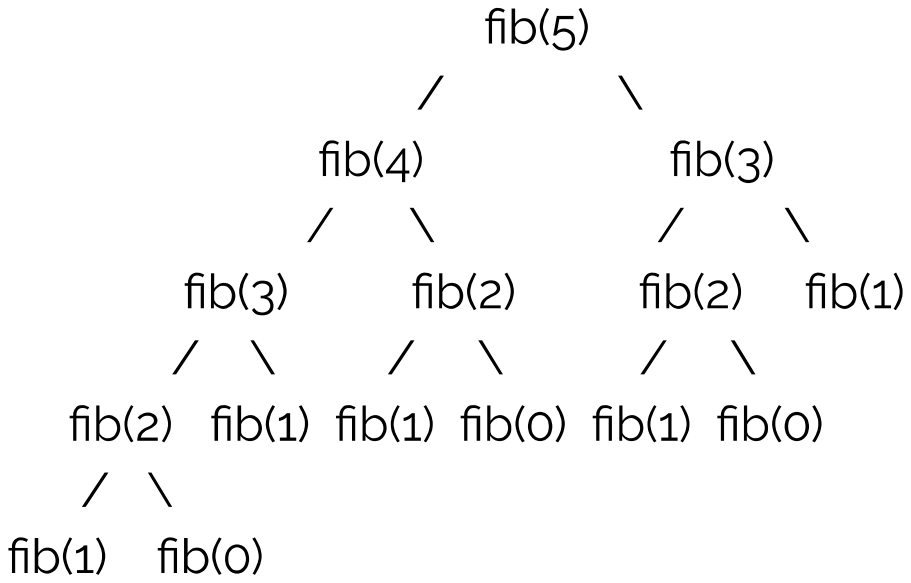
DYNAMIC PROGRAMMING

Those who cannot remember the past are condemned to repeat it.

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.

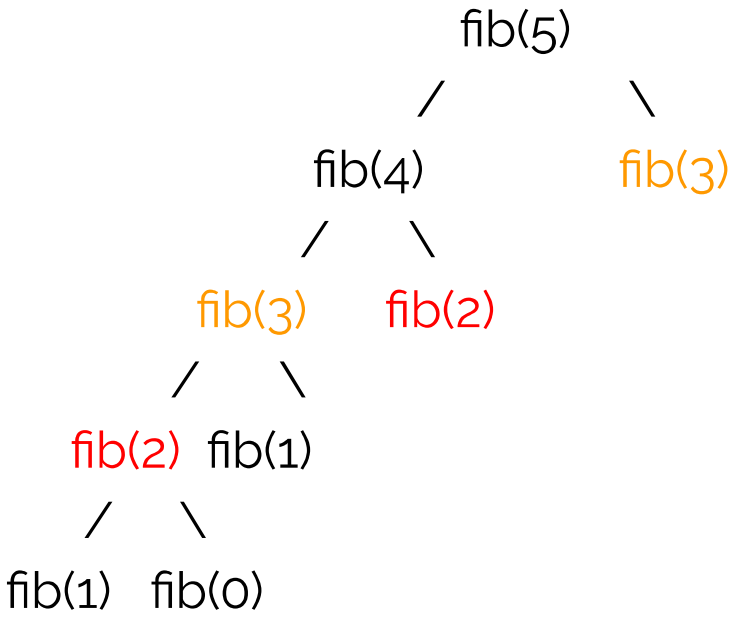
Recursion Tree

Overlapping subproblem



DP Fibonacci

Optimisation using DP :

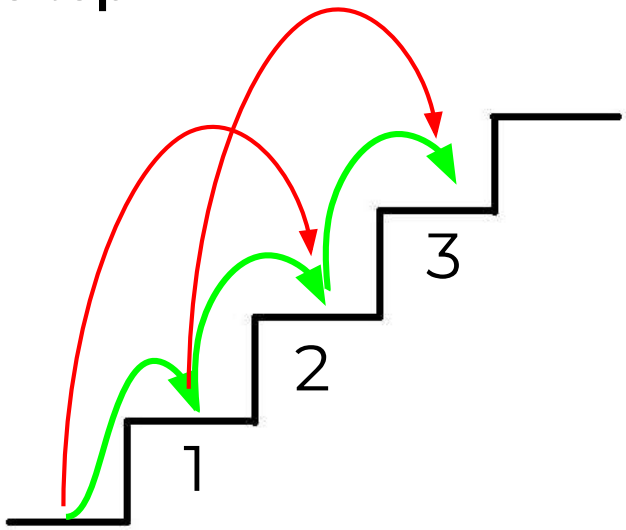


```
#include <bits/stdc++.h>
using namespace std;

int a[100];

int fib(int n)
{
    if(a[n]==-1) {
        a[n] = fib(n-1) + fib(n-2);
    }
return a[n];
}
```

```
int main()
{
    int n;
    cin>>n;
    memset(a,-1,sizeof(a));
    a[0]=0,a[1]=1;
    int res = fib(n);
    cout<<res;
    return 0;
}
```

Count ways to reach the n'th stair

There are n stairs, a person standing at the bottom wants to reach the top. The person can climb either 1 stair or 2 stairs at a time. Count the number of ways, the person can reach the top.



ways(1) = 1
ways(2) = 2
ways(3) = ways(2) + ways(1) = 3
ways(4) = ways(3) + ways(2) = 5
ways(n) = ways(n-1) + ways(n - 2)

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
    int n;
    cin >> n;
    int dp[n + 5];
    memset(dp, 0, sizeof(dp));
    dp[0] =0;
    dp[1] =1;
    dp[2] =2;
    for(int i = 3; i <= n; i++)
    {
        dp[i] = dp[i-1] + dp[i-2];
    }
    cout << dp[n];
}
```

INPUT

5

OUTPUT

8