Recursion and backtracking

16 June 2024 13:59

Recursion :- When a f^n calls itself, it is called recursion.

```
main()
{
    A();
    print("main");
}

A()
{
    print("A");
}
```

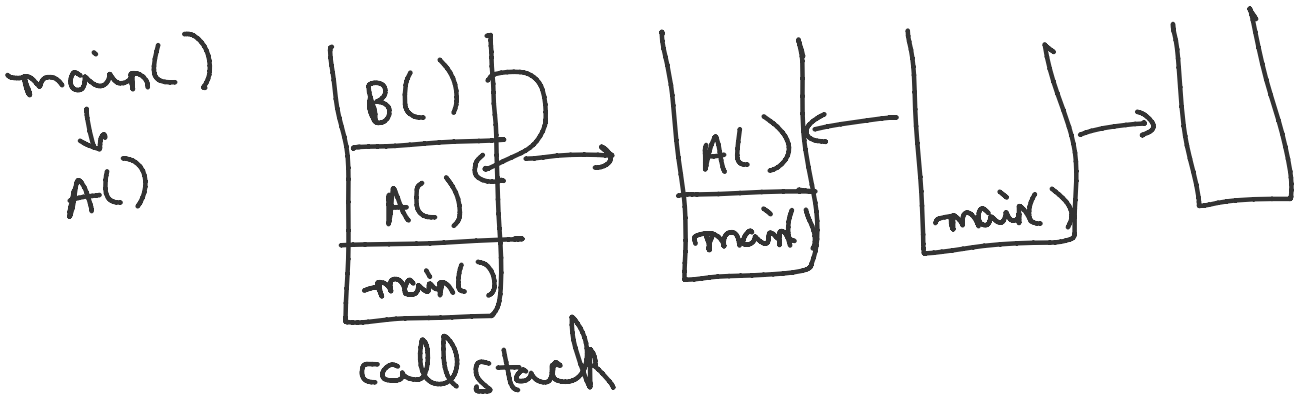
A
main

```
main()
{
    A();
    print("main");
}

A()
{
    print("A");
    B();
}

B()
{
    print("B");
}
```

O/P:-

A
B
A
main

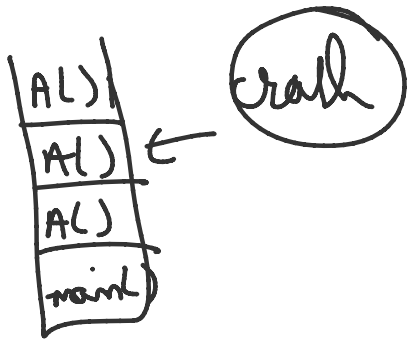
Call stack



SC = space req. to store I/P + AS
↓
call stack.

```
main()
{
  A();
}
```

```
A(){
  print("A");
  A();
}
```



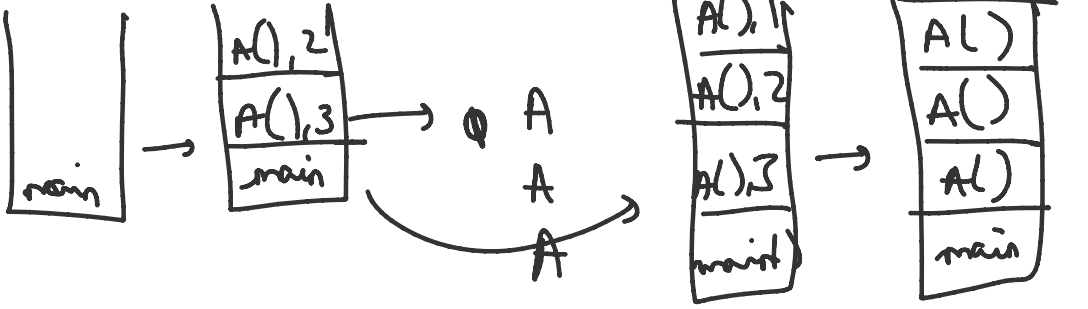
In order to stop recursion in finite amount of time, we add a base condition.

Q. Print "A" 10 times.

```
main()
{
  A(10);
}
```

```
A(n){
  if(n==0) return; } → base condition
  print("A");
  A(n-1);
}
```

n=3



Q. Print A n times using recursion

TC = O(n), AS = O(n).

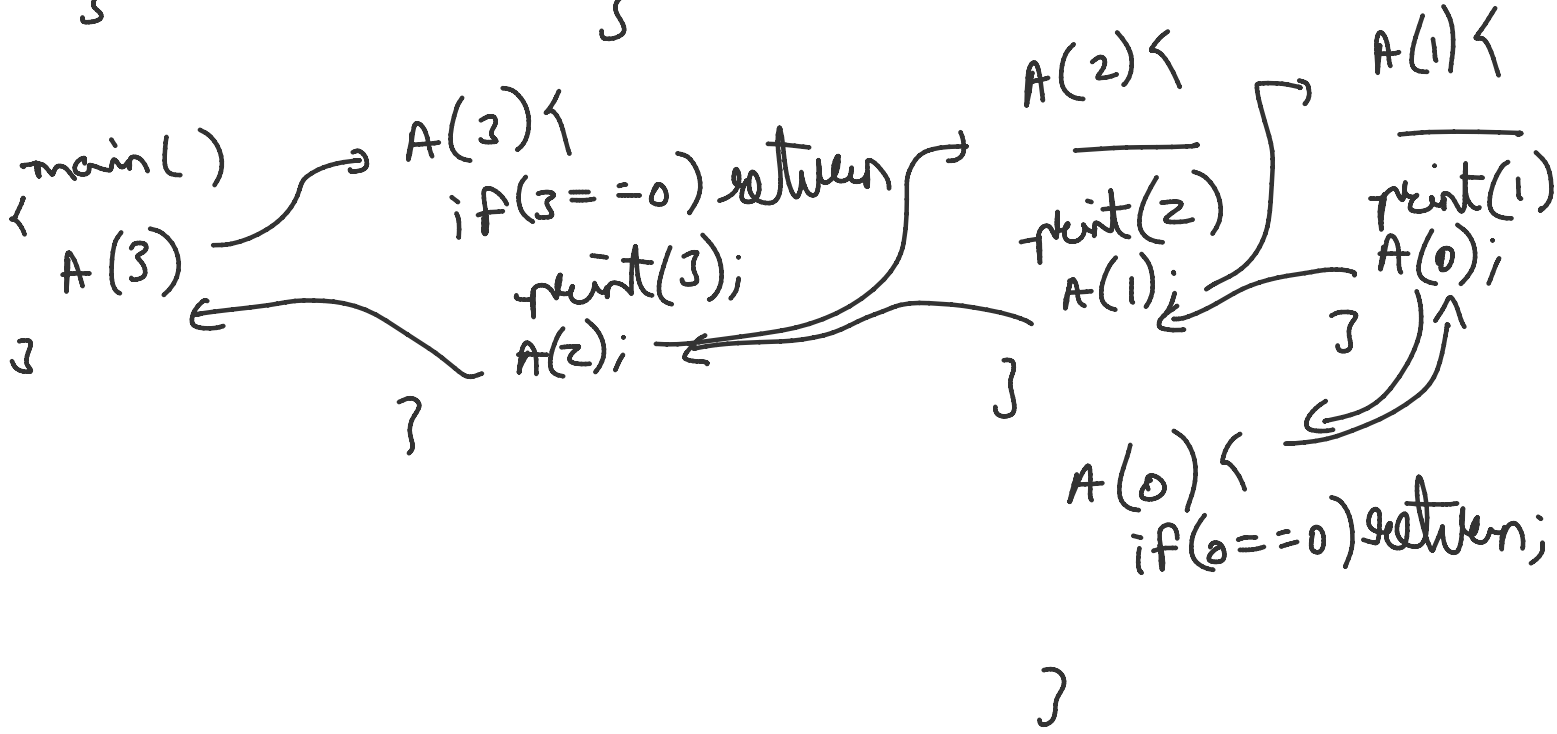
Q. Print numbers from n to 1.

```
main()
```

```
A(n){
  if(n==0) return;
```

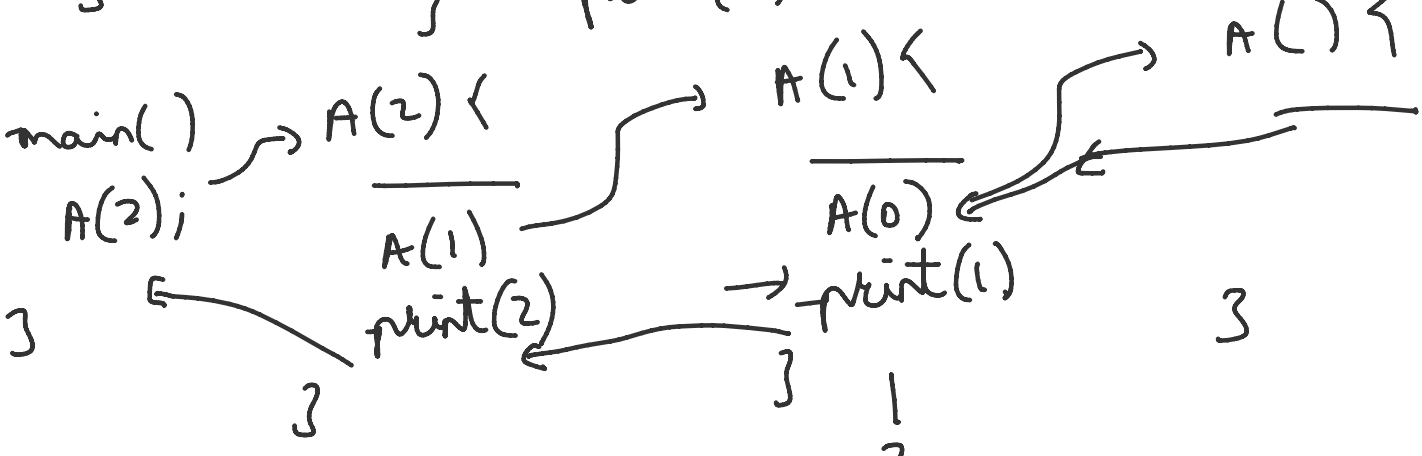
```
main()
{
  A(6);
}
```

```
A(n){
  if(n==0) return;
  print(n);
  A(n-1);
}
```



Q.
```
main()
{
  A(3);
}
```

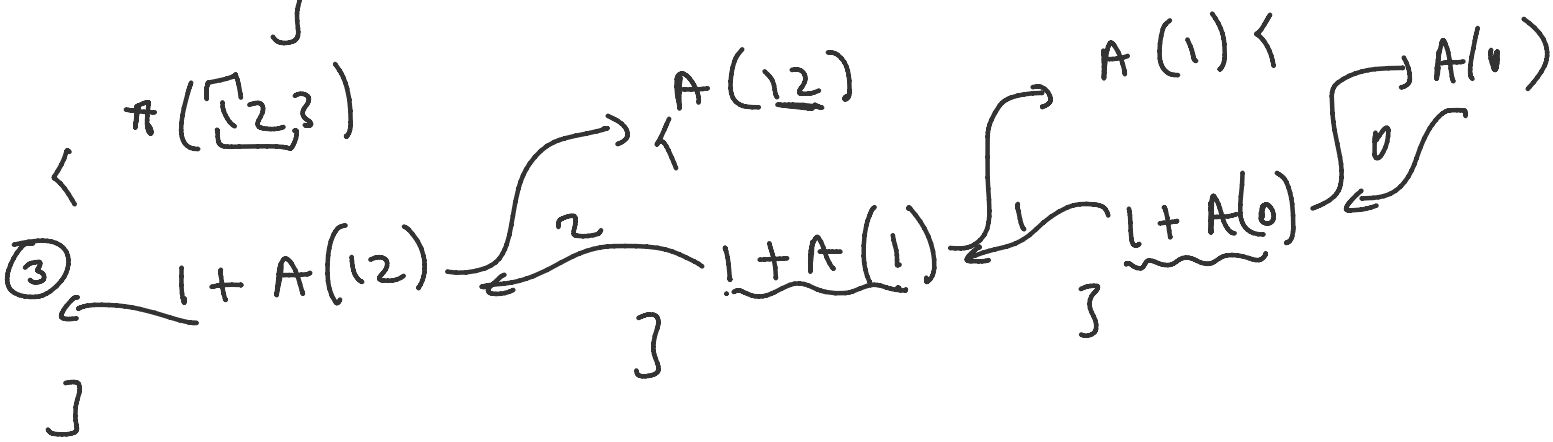
```
A(n){
  if(n==0) return;
  A(n-1);
  print(n);
}
```



General structure of recursion:-

General structure of recursion:-

```
A( ) {
    } → base condn
  ----- → execution
  A() → recursive call
  ----- → execution
}
```



Q/ $5^4 \rightarrow 5 \times 5^3$

pc(5,4) → 5 × pc(5,3) → 5 × pc(5,2)

Properties of %:-

$(x+y)\%10 \rightarrow 0$ to 9

$(3^x)\%m$

```
ans=1;
for(i=1; i<=x; i++)
{   ans=ans*3;
    ans=ans%m;
}
```

X

```
ans = ans%m;
for(i=1; i<=x; i++)
{
    ans=(ans*3)%m;
}
return ans;
```

$(x+y+z)\%m = ((x+y)\%m+z)\%m;$

$(abc)\%m = (((ab)\%m)c)\%m;$

$(a-b)\%m$ ✗ $(a-b+m)\%m$

$n^p \rightarrow$

```
ans=1;
for(i=1; i<=p; i++)
{
    ans=ans*n;
}
```

→ O(p)

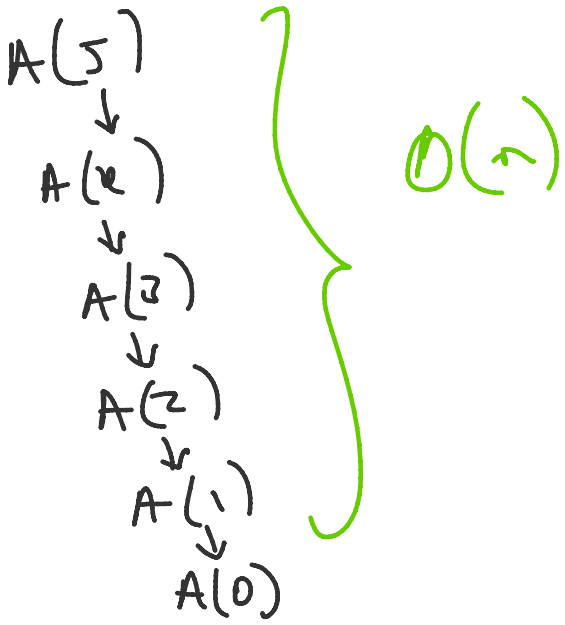
→

```
ans=1;
while(p>0)
{
    if(p%2==1)
    {
        ans = ans*n;
    }
    n=(n*n);
    p=p/2;
}
```

→ O(log p)

Recursion

Recursion Tree :-



Q/ Fibonacci Series :-

1, 1, 2, 3, (5), 8, 13, 21, 34, 55 ------

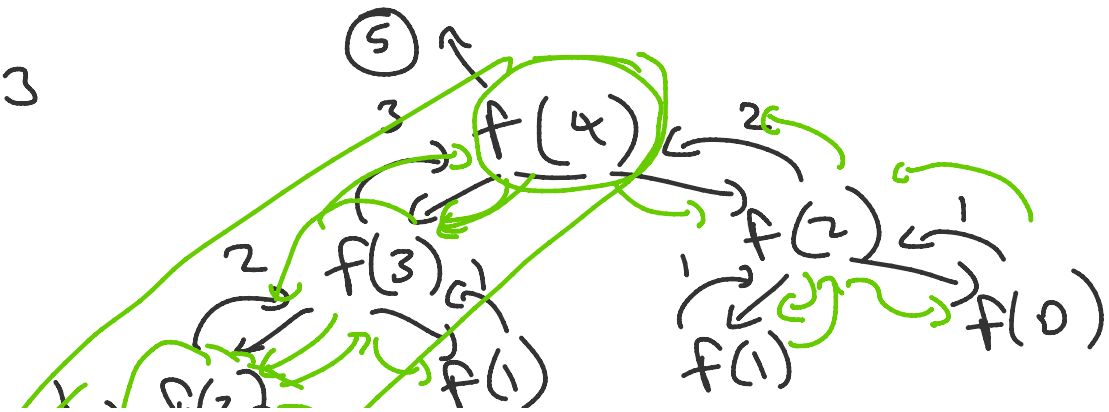
n=0 n=1

fib(n) = fib(n-1) + fib(n-2)

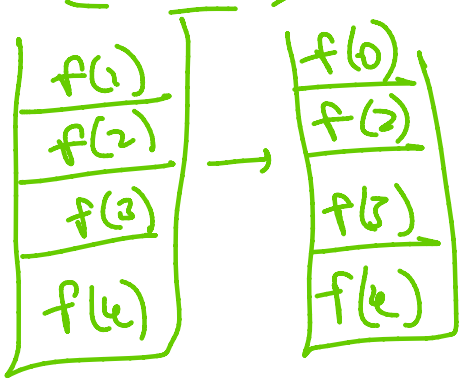
```
if(n==0 || n==-1)
    return 1;
```

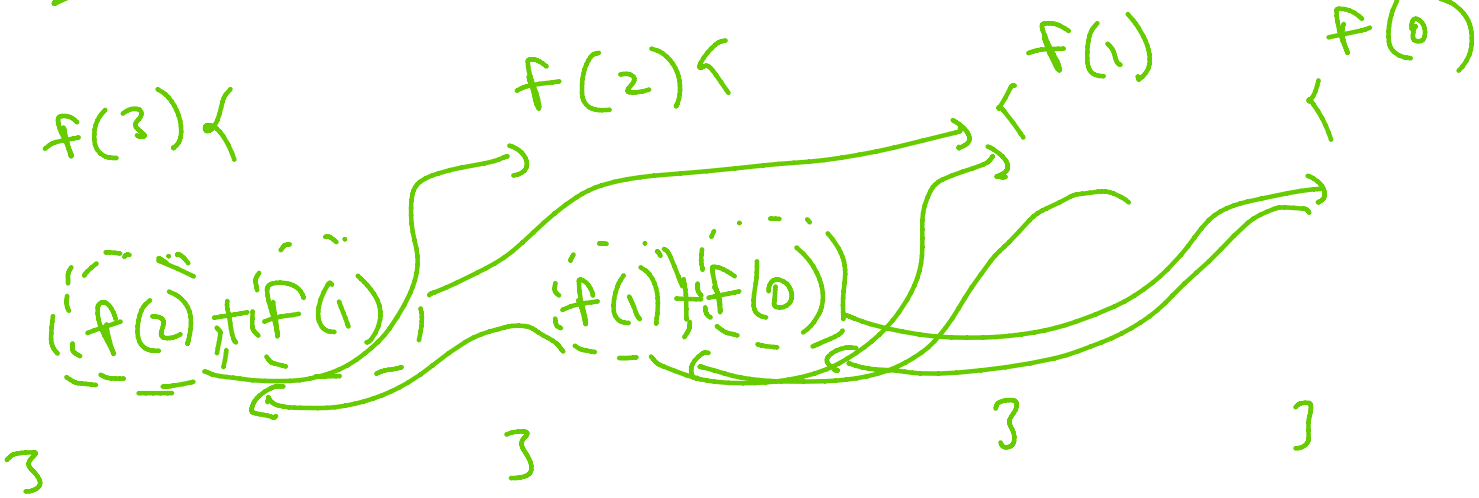
```
int f(int n)
{
  if(n==0 || n==-1)
      return 1;

  return f(n-1) + f(n-2);
```

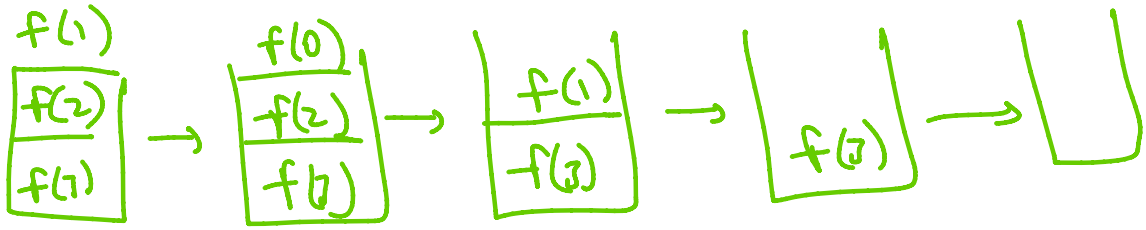
3



f(1) → f(0)








```
int f(int n)
{
    if(n==0 || n==1)
        return 1;
    int p1 = f(n-1);
    int p2 = f(n-2);
    return p1+p2;
}
```

]

max no of fn in call stack = ht of the tree

AS = ht of the tree × space occupied by each fn.

Time & Space Complexity

① Substitution
② Recursion Tree
③ Masters theorem (H.W)

① $T(n) = T(n-1) + c$

$T(n-1) = T(n-2) + c$

$T(n-2) = T(n-3) + c$

$\vdots$

$T(0) = c$

$T(n) = T(n-2) + 2c$

$T(n) = T(n-3) + 3c$

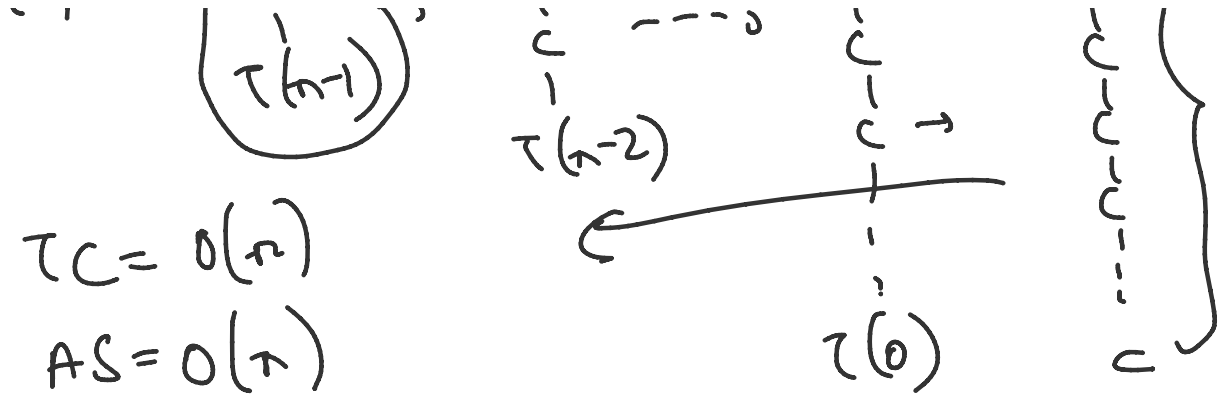
$\vdots$

$T(n) = T(0) + nc$

$= (n+1)c$

$= O(n)$

$T(n) \rightarrow$ 



$TC = O(n)$

$AS = O(n)$

(2) $T(n) = T(n/2) + C \quad T(1) = C$

$T(n/2) = T(n/4) + C$

$T(n/4) = T(n/8) + C$

$\vdots$

$T(0) = C$

$T(n) = T(n/4) + 2C$

$T(n) = T(n/8) + 3C$

$T(n) = T(n/2^K) + KC$

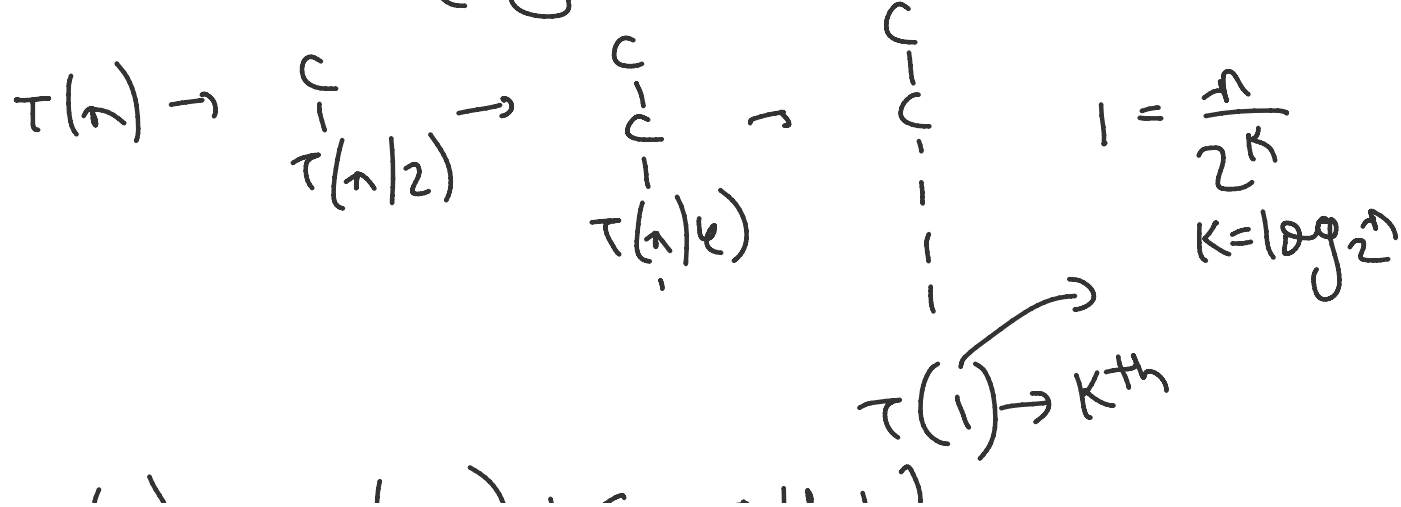
$\frac{n}{2^K} = 1 \quad 2^K = n$

$K = \log_2 n$

$T(n) = T(1) + \log_2 n\, C$

$= O(\log_2 n)$

$TC = AS = O(\log_2 n)$



③ $T(n) = 2T(n-1) + C \rightarrow$ H.W

④ $T(n) = 2T(n/2) + C$, $T(1) = C$

$T(n/2) = 2T(n/4) + C$

$T(n/4) = 2T(n/8) + C$

$T(n) = 2\left[2T(n/4) + C\right] + C$

$= 2^2 T(n/4) + 3C$

$T(n) = 2^2\left[2T(n/8) + C\right] + 3C$

$= 2^3 T(n/8) + 7C$

$T(n) = 2^K T\left(n/2^K\right) + (2^K - 1)C$

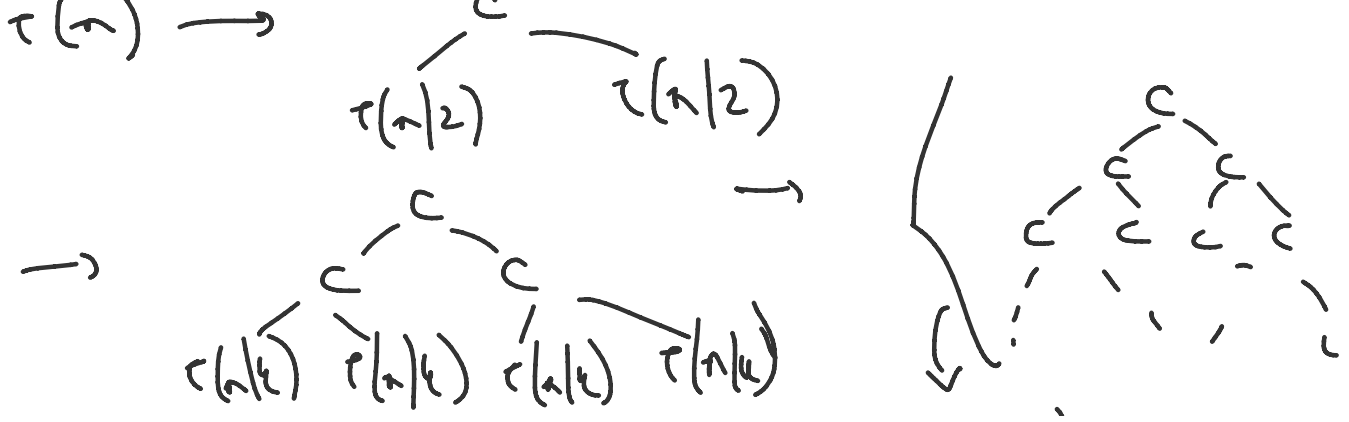
$\frac{n}{2^K} = 1$

$2^K = n$

$T(n) = nT(1) + (n-1)C$

$= nC + (n-1)C$

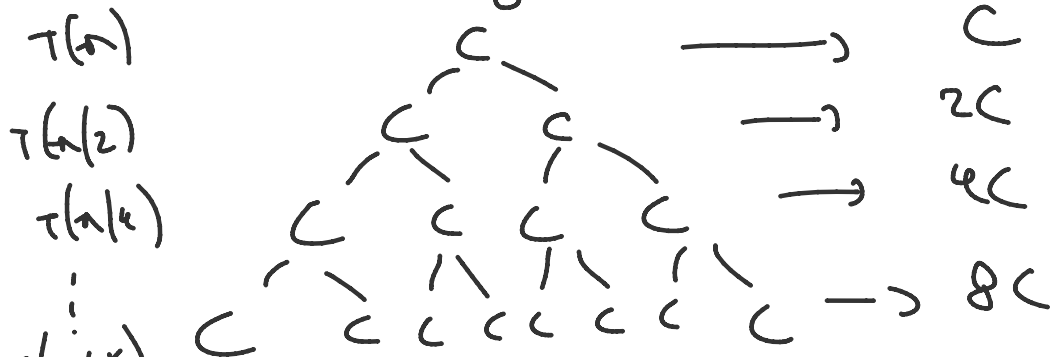
$= (2n-1)C = \underline{O(n)}$



$T(n/4)$ $T(n/4)$ $T(n/4)$ $T(n/4)$

$AS = O(\log n)$

$Lt = \log_2 n$



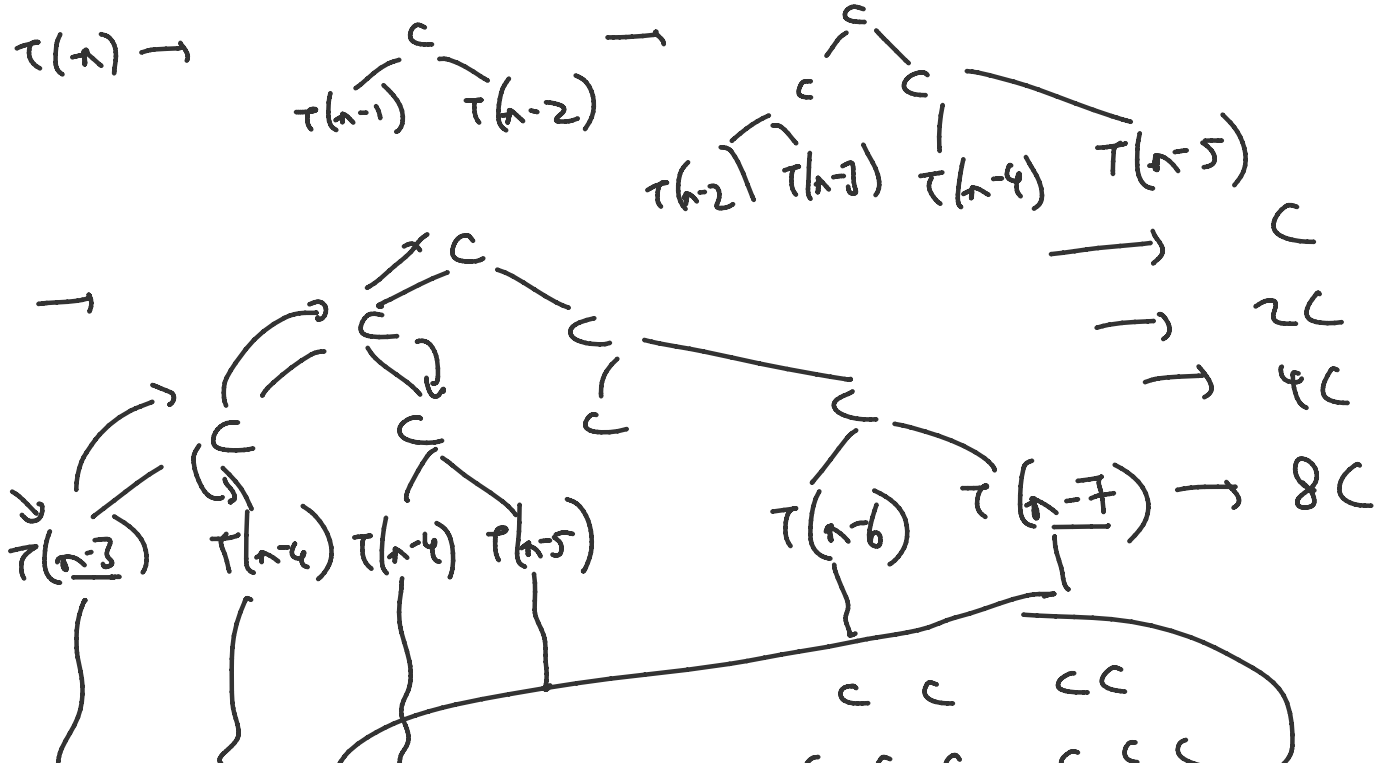
$$T = C + 2C + 4C + 8C \cdots + 2^K C$$

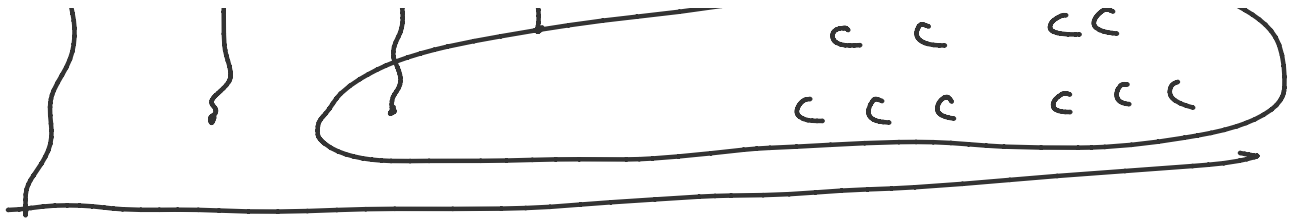
$$= \frac{C(2^K - 1)}{1} = C(2^{\log_2 n} - 1)$$

$$= C(n-1)$$

$$= O(n)$$

Q $T(n) = T(n-1) + T(n-2) + C$





$T = c + 2c + 4c \longrightarrow +2^n c$

$= \frac{c(2^n - 1)}{2-1} = O(2^n)$ ✓ $\quad AS = O(n)$

$T(n) = 2T(n-1) + c$

$T(n) = T(n-1) + T(n-2) + c \longrightarrow (1.61^n)$ ✓

Q. Given an array, print all its subsets.

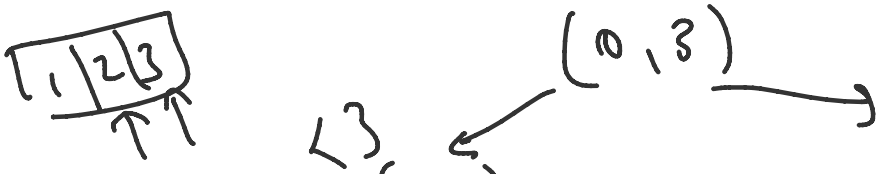
take / not take

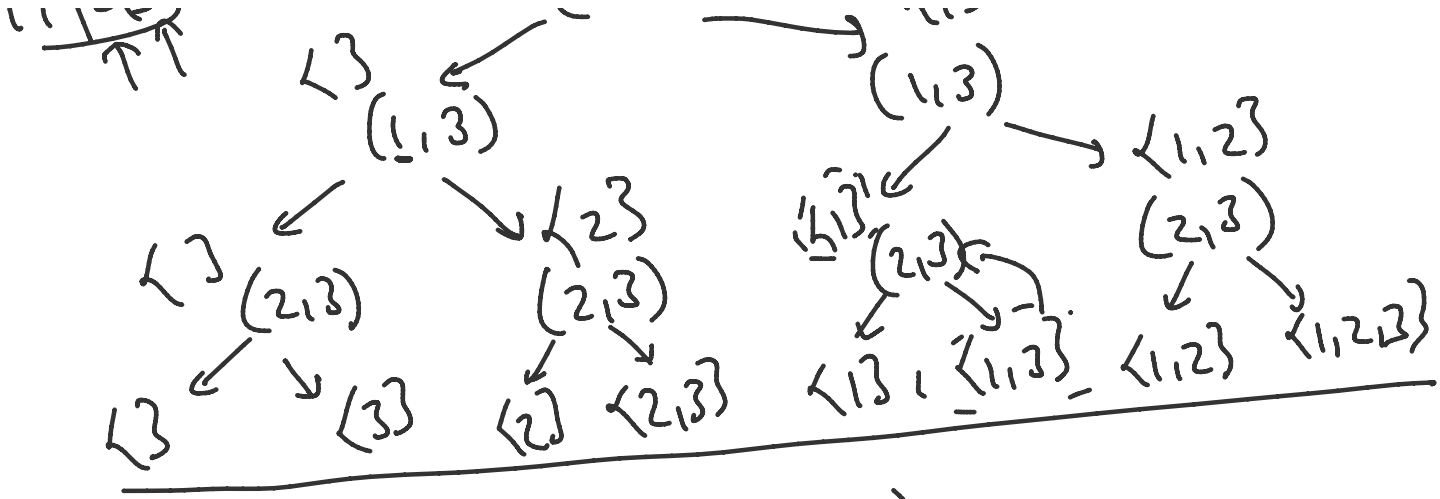
```
void subset (int i, int n)
{
    if(i==n)
        return;
    subset (i+1, n);
    s.add(a[i]);
    subset (i+1, n);
    s.remove(a[i]);
}
```

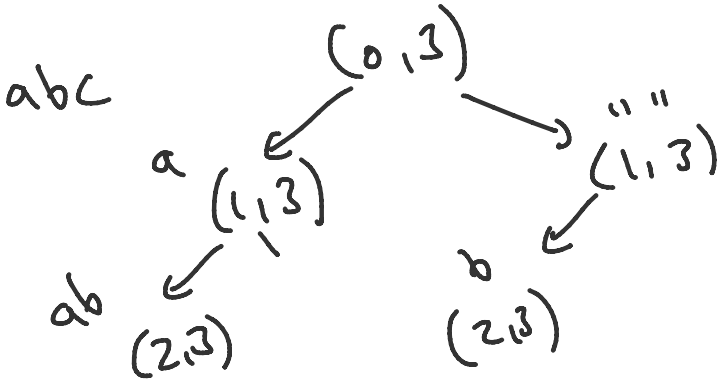
$T(n) = 2T(n-1) + c$

$\downarrow$

$O(2^n)$







⇒ Backtracking :- It is the technique to go back to the previous recursion call on finding a dead end.

Q. Given an array, check whether a subset exists with a given sum S.

true ← a = {1, 7, 5, 7, 10, 2} S = 12

a = {14, 1, 8, 13} → false

```
bool combination_sum(int i, int n, int s)
{
    if (s<0)
        return False;
```

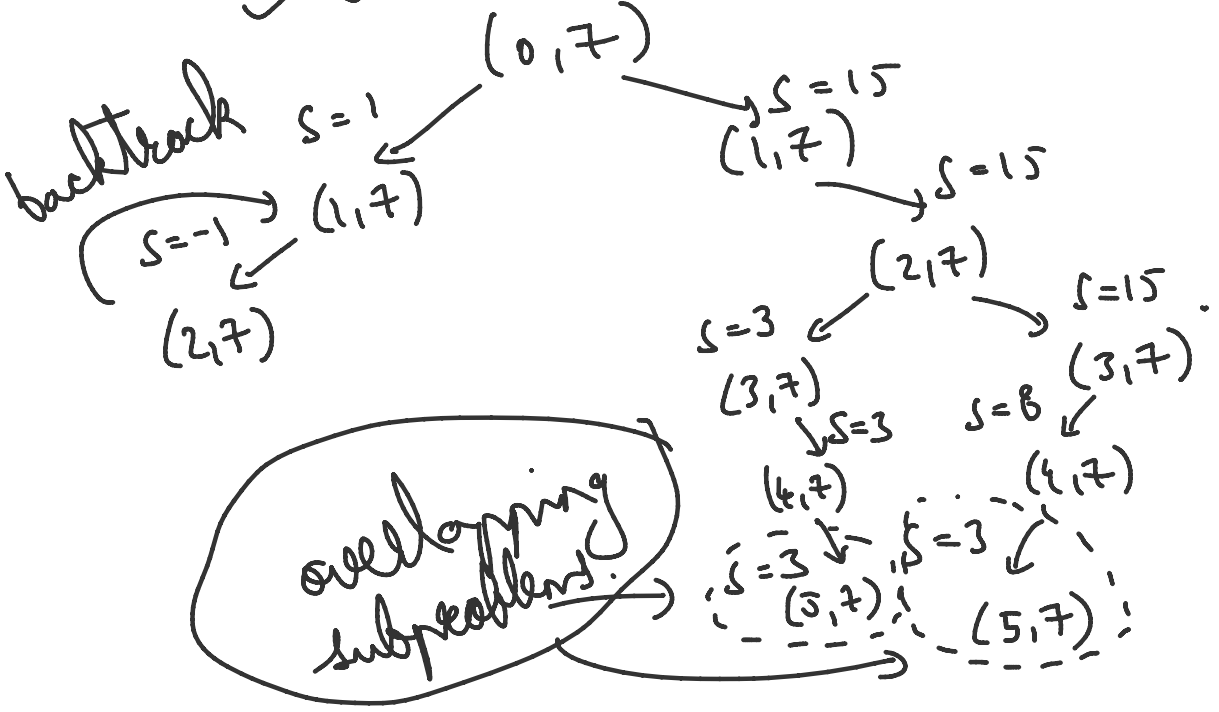
subset sum problem

```
    if (s==0)
        return true;
    if(i==n)
        return false;
    return cs(i+1,n,s) || cs(i+1,n,s-a[i]);
}
```

subset sum problem

{1, 3, 5, 7, 12, 2, 14} s = 15

{14, 2, 12, 7, 5, 3, 1} s = 15



{2, 5, 10, 7}