Today's content

- Recursion
- How to write a recursive code / tracing
- TC & S.C of recursive codes → [next session]

Why recursion ?

- Merge Sort / Quick Sort / Many other algorithms.
- Binary Tree / BST / BBST / Tries
- Dynamic Programming
- Backtracking
- Graph.

Recursion → function calling itself.

↳ solving a problem, using smaller instance of the same problem.

⇓

sub-problem

$Sum(N) = \underbrace{1 + 2 + 3 + 4 + \cdots (N-1)}_{} + N$

$Sum(N) = Sum(N-1) + N$

$Sum(4) = Sum(3) + 4$ // sum(3) is a sub-problem

How to write Recursive codes ?

Assumption : Fix what should your function do.

Main Logic : Solving assumption using sub-problems

Base condition : Input values, for which we want to stop the recursion.

$N > 0$

```
int Sum ( N) { //Ass → Given N , calculate & return sum of 1st N natural no's.
    if ( N == 1) { return 1 }
    return (sum(N-1) + N )
}
```

1+2+3+ — (N-1) + N : Sum of first N Natural no's.

sum(3) = 6.

sum(4) = 10

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

$N \geq 1$

```
int fact ( N) { // Ass → Given N, calculate & return N!
    if (N == 1) return 1
    return (fact(N-1) * N )
}
```

fact(N) = 1*2*3* — (N-1) * N

fact(N) = fact(N-1) * N

Function Call Tracing.

```
int add (N, m) {
    return N+m
}

int mul ( x, y) {
    return x*y
}

int sub ( x, y) {
    return x-y
}
```

```
main () {
    x=10, y=20
    print ( sub (mul (add (x,y), 30),75))
}
```

sub (mul (add (x,y), 30) , 75) : return 825

900

mul (add (x,y), 30) : return 900

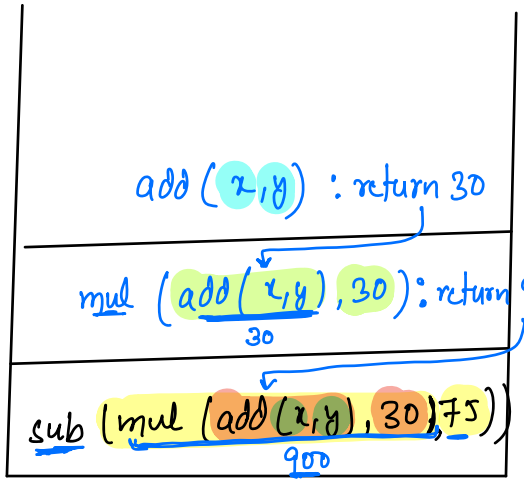
30

add (x,y) : return 30

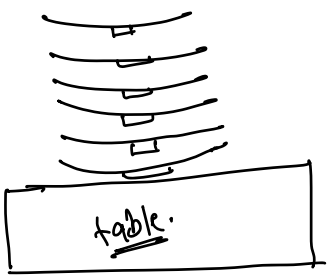
// Data Structure.

obs1 : Whenever a function call is made, we need to add that function on the top.

obs2. : When function return, we remove it from the top.



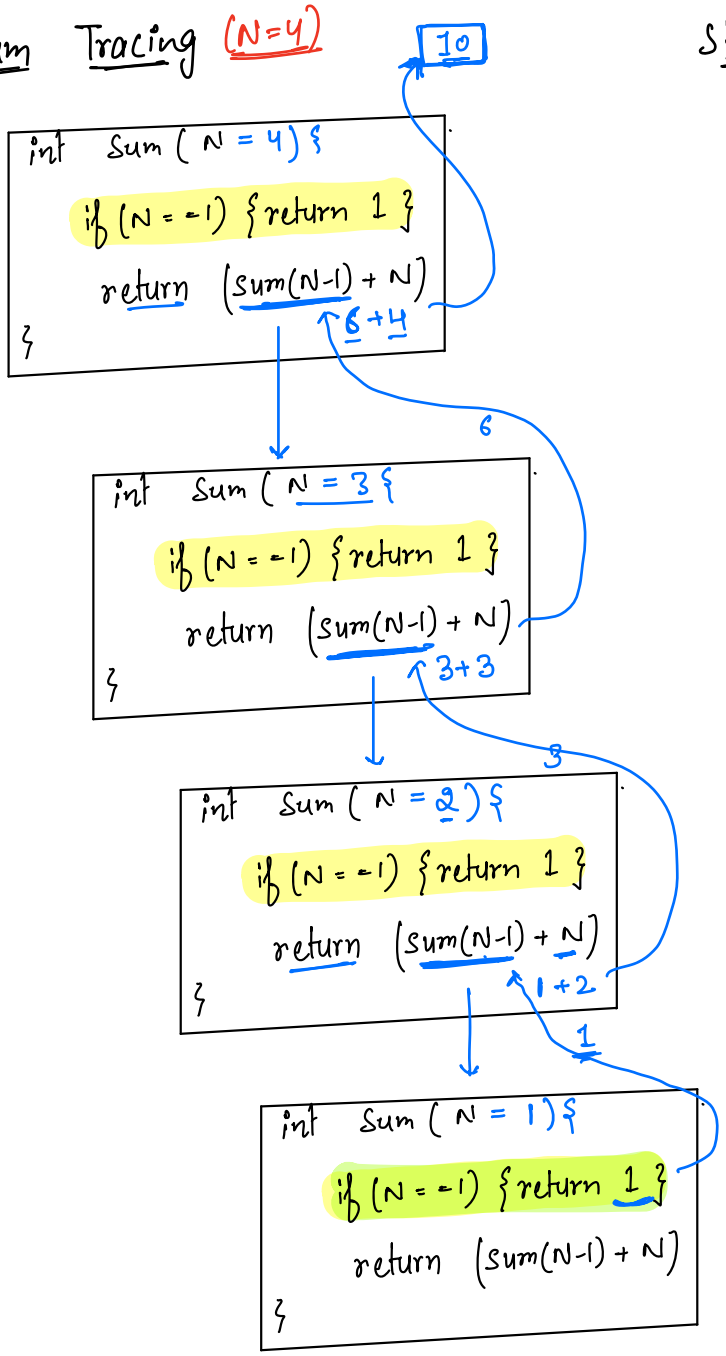
Note : All function calls are stored in stack



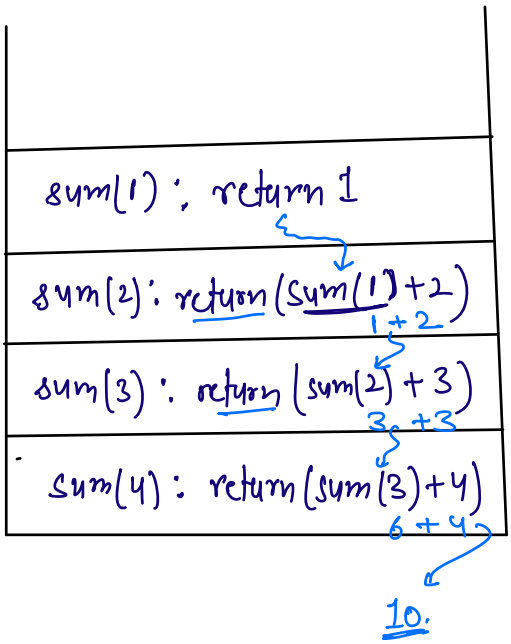
→ Stack property

LIFO → Last In First Out.

//Sum Tracing (N=4)



Stack trace.



todo → trace factorial.

Without Base Condition Recursion won't stop, memory limit exceeds first.

- TLE
- Memory Limit Exceeded / Stack overflow

Note → In recursion, if your idea give Memory Limit Exceeded that means, code is not properly stopped, verify base conditions.

$N \geq 0$

# input :	0	1	2	3	4	5	6	7	8	9	10
fib() :	0	1	1	2	3	5	8	13	21	34	55

```
int fib (N) {  //Ass → Calculate & return Nth fibonacci number.
    if (N <= 1) return N
    return { fib(N-1) + fib(N-2) }
}
```

fib(5) → 5
fib(6) → 8
fib(7) → 13

fib(N) = Sum of previous 2 fib. numbers.

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

//Note → How to figure out base conditions

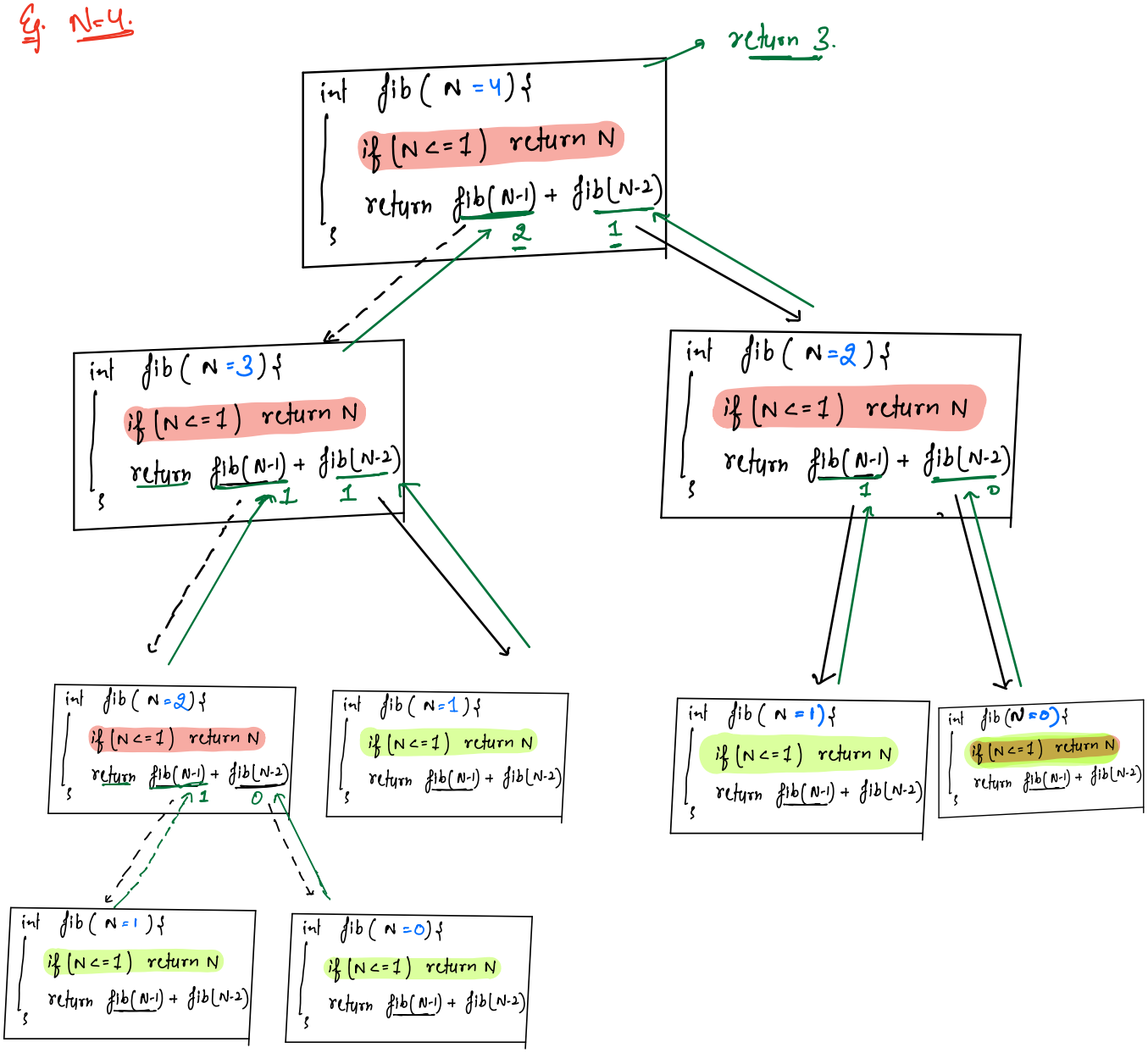
→ valid values where expression will be invalid.

fib(0) = fib(-1) + fib(-2) ✗

fib(1) = fib(0) + fib(-1) ✗

fib(2) = fib(1) + fib(0) ✓

Eg. N=4.



→ Try tracing it out with Stack (#todo)

Break → 11:42 → 11:50

// Given N, print all numbers from 1 → N in increasing order.

```
void pI (N) {  // Assm → Give N, print all no's from [1,N] in increasing order.
    if (N == 1) { print 1 }
    pI(N-1)
    print(N)
}
```

pI(3) = 1 2 3

pI(4) = 1 2 3 4

pI(N) = 1 2 3 ---(N-1) N

pI(N-1)

N=3.

```
void pI (int N) {3
    if (N == 1) print N
    if (N > 1) {
        pI(N-1)
        print(N)
    }
}
```

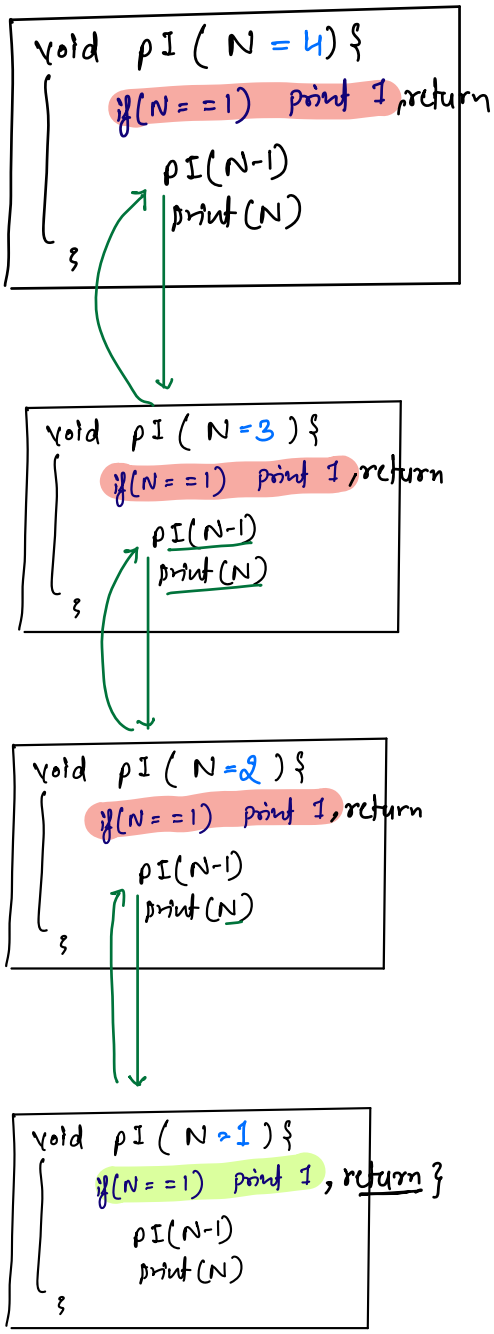
```
void pI (int N) {2
    if (N == 1) print N
    if (N > 1) {
        pI(N-1)
        print(N)
    }
}
```

```
void pI (int N) {1
    if (N == 1) print N
    if (N > 1) {
        pI(N-1)
        print(N)
    }
}
```

o/p

1
2
3

N=4.



o/p:

1
2
3
4

Note → Even for void functions, you can use return
→ function call will be removed from the stack.

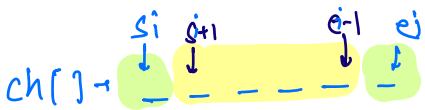
Q) Given a substring. Check if it is palindrome or not ?
[s <= e]

Eg1 : g o o d d a d (0 1 2 3 4 5 6) s=4, e=6 : YES

Eg2 : g o o d d a d (0 1 2 3 4 5 6) s=2, e=5 : NO

Assumption: Return, if substring from [s, e] is palindrome or not.

```
boolean isPal(char[] ch, int si, int ei){
    if(si > ei){ return true }
    if(ch[si] == ch[ei] && isPal(ch, si+1, ei-1)){
        return true
    }
    return false
}
```

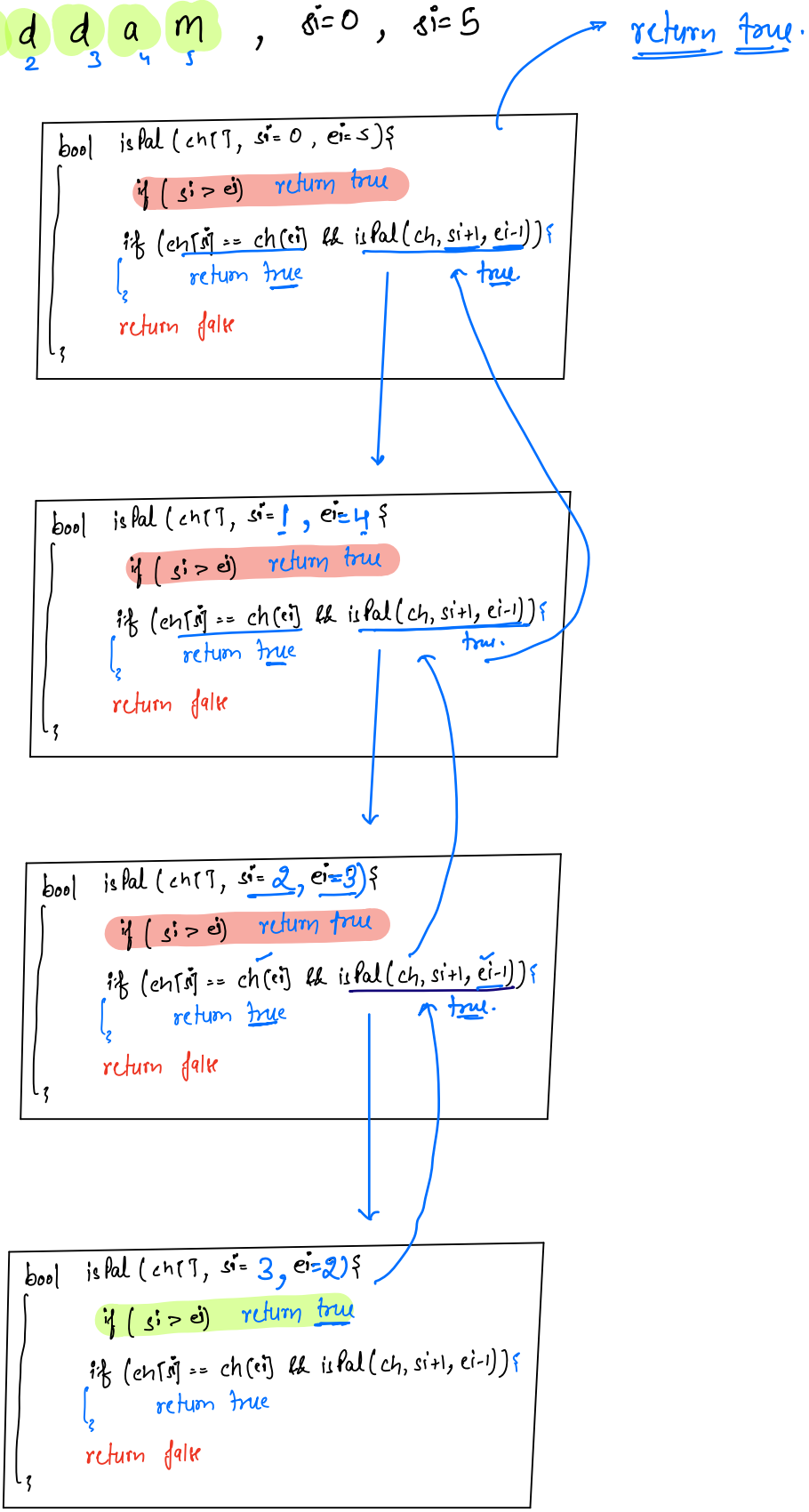


Eg: m a d a m (0 1 2 3 4) (0,4) → (1,3) sub-problem.

m a l a y a l a m (0 1 2 3 4 5 6 7 8) (0,7) → (1,6) sub-problem

a b b a (0 1 2 3)

Input - m a d d a m , si=0 , si=5

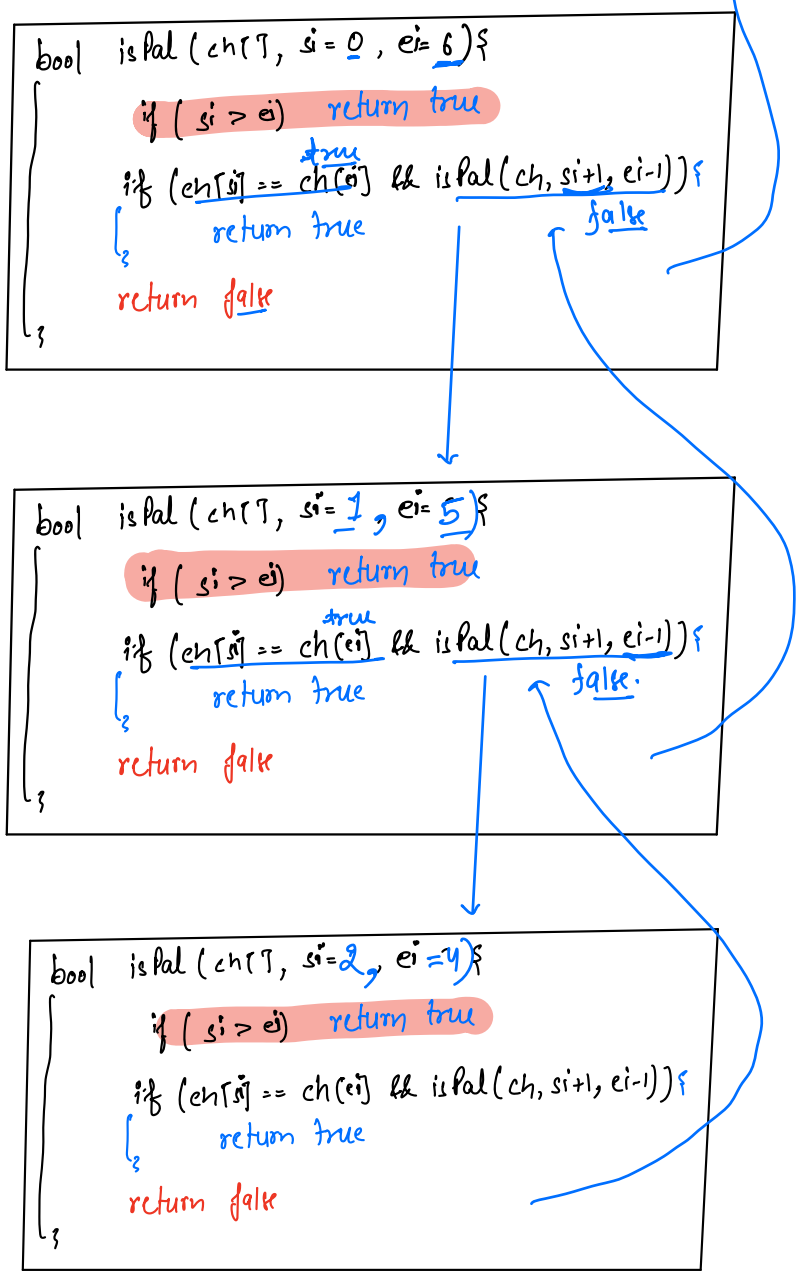


Input: a n m e t n a

0 1 2 3 4 5 6

s=0, e=6

return false.



Eg:- m a d a m
0 1 2 3 4

true.

```
bool isPal(ch[], si=0, ei=4){
    if (si > ei) return true
    if (ch[si] == ch[ei] && isPal(ch, si+1, ei-1)){
        return true
    }
    return false
}
```

(ch[si] == ch[ei] : true ; isPal(ch, si+1, ei-1) : true)

```
bool isPal(ch[], si=1, ei=3){
    if (si > ei) return true
    if (ch[si] == ch[ei] && isPal(ch, si+1, ei-1)){
        return true
    }
    return false
}
```

(ch[si] == ch[ei] : true ; isPal(ch, si+1, ei-1) : true)

```
bool isPal(ch[], si=2, ei=2){
    if (si >= ei) return true
    if (ch[si] == ch[ei] && isPal(ch, si+1, ei-1)){
        return true
    }
    return false
}
```

(ch[si] == ch[ei] : true ; isPal(ch, si+1, ei-1) : true)

```
bool isPal(ch[], si=3, ei=1){
    if (si > ei) return true
    if (ch[si] == ch[ei] && isPal(ch, si+1, ei-1)){
        return true
```

```
    return false
}
```

str = "aadda"

0 1 2 3 4

↓

adda