Todays Content:

→ Recursion Intro

→ How to write Recursive Codes

→ TC/SC of Recursion in Recursion-3

→ Recursion:

a) Sorting

b) Trees

c) Back Tracking

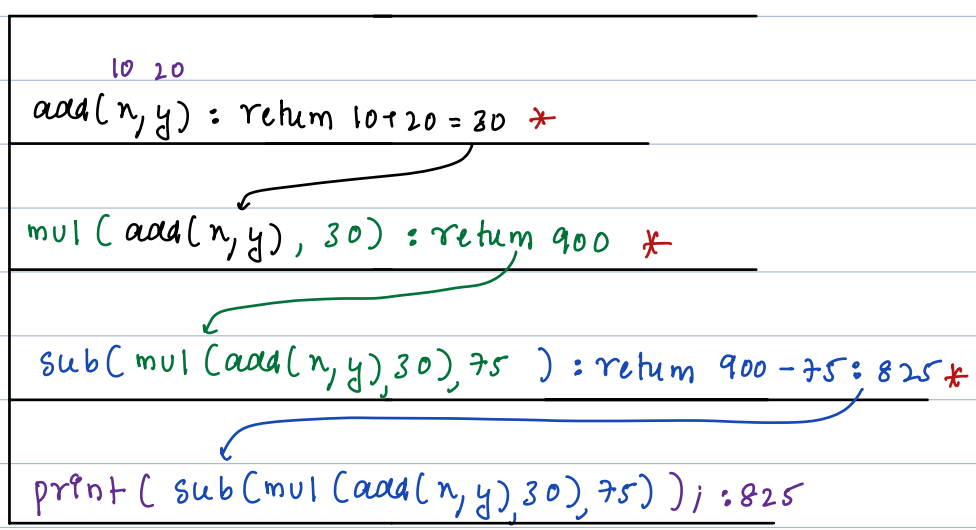
d) Dynamic Programming

# Function Calls

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

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

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

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

Note:

1. Every time a function call is called, it is stored at top of Container
2. When function returns or function completes execution, it will exit Container
3. Above Container is called Stack.



Recursion: Function calling itself

Solving a problem using Smaller Input of Same Problem: SubProblem

Sum(5) = 1 + 2 + 3 + 4 + 5

Sum(5) = Sum(4) + 5

↳ SubProblem

Sum(N) = 1 + 2 + 3 + .. N-1 + N

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

Steps to Recursive Code:

Assumption : Decide what your function does & assume it works

Hint: Question itself Assumption

Main Logic : Solving asumption using SubProblems

Base Conditions : Input, for which we want to stop recursion

↳ Why? In Stopping Recursion

Note: Don't use post increment operators, while writing recursion.

```
int sum(int N){
    if(N==1){return 1} : Base
    return sum(N-1) + N : Main
}
```

Asst: Given N, calculate & return sum of N Natural numbers

sum(4) = return 10  sum(5) = return 15

## Tracing with line Numbers

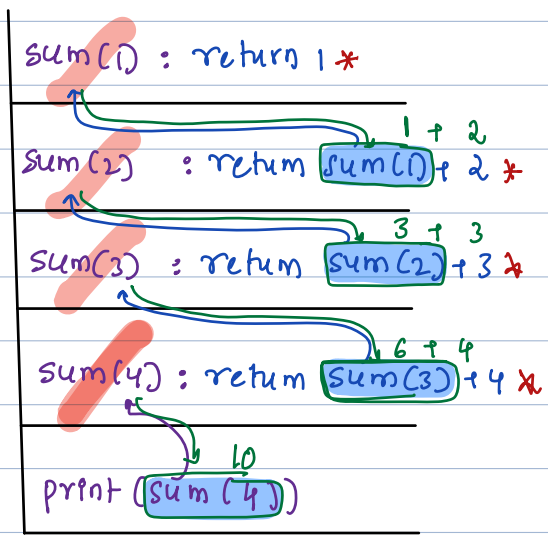
```
int main(){
    print(sum(4))   // 10
}
```

```
int sum(int N=4) : a
    1. if(N==1){return 1}
    2. return sum(N-1) + N;
}
```

```
int sum(int N=3) : b
    1. if(N==1){return 1}
    2. return sum(N-1) + N;
}
```

```
int sum(int N=2) : c
    1. if(N==1){return 1}
    2. return sum(N-1) + N;
}
```

```
int sum(int N=1) : d
    1. if(N==1){return 1}
    2. return sum(N-1) + N;
}
```



# Importance of Base Conditions

1. Recursion wont stop : Error TLE

   a) Infinite calls : a) TLE, at max iterations $\approx 10^8$ iteration

   : Say 1 Function $\approx$ 1 iteration

   : at $10^8$ function calls we TLE

   b) Every function stored in stack :

   Note: Limit of a stack $\approx$ 1MB $\approx 10^5 - 10^6$ functions

   : at $10^5 - 10^6$ we reach stack limit : MLE

   Memory limit Exceeded

| ↑ |
|---|
| fun(N=-1) |
| fun(N=0) |
| fun(N=1) |

2. Base always at top

   a) If we write at bottom, they are never called

Fact(N) = 1 * 2 * 3 * .. N-1 * N

Fact(5) = 1 * 2 * 3 * 4 * 5

Fact(5) = fact(4) * 5 // Problem: fact(5) ⇒ SubProblem: fact(4)

Fact(N) = 1 * 2 * 3 * .. N-1 * N

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

Input: 0 <= N <= 10

```
int fact(int N){                    Ass: Given N : Calculate & return factorial of N
  1. if(N==1 || N==0){  ⇒ // if(N<=1)
  2. }   return 1
  3. return fact(N-1) * N
```

Tracing:

```
int fact(int N==4){                 → 24:
  1. if(N==1 || N==0){
  2. }   return 1
  3. return fact(N-1) * N           6
```

```
int fact(int N==3){
  1. if(N==1 || N==0){
  2. }   return 1
  3. return fact(N-1) * N           2
```

```
int fact(int N==2){
  1. if(N==1 || N==0){
  2. }   return 1
  3. return fact(N-1) * N           1
```

```
int fact(int N==1){
  1. if(N==1 || N==0){
  2. }   return 1
  3. return fact(N-1) * N
```

```
int fact(int N==0){                 → 1
  1. if(N==1 || N==0){
  2. }   return 1
  3. return fact(N-1) * N
```

```
int fact(int N==0){
  1. if(N==1){
  2. }   return 1
  3. return fact(N-1) * N
```

```
int fact(int N==-1){
  1. if(N==1){
  2. }   return 1
  3. return fact(N-1) * N
```

fact(-2) → fact(-3) → .. TLE

## Fibanacci:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | . . . |
|---|---|---|---|---|---|---|---|---|---|---|
| Series = | 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | . . . . |

Fib(10) = Fib(9) + Fib(8) // Sub Problems: fib(9) & fib(8)

Fib(N) = Fib(N-1) + Fib(N-2) // Sub Problems: fib(N-1) & fib(N-2)

Constraints: $0 <= N <= 20$

```
int Fib(int N){      Ass: Given N, Calculate & return Nth fib number
    if(N==0) {return 0}
    if(N==1) {return 1}
    return fib(N-1) + fib(N-2)
}
```

How to get baseConditions?

Inputs for which main logic fails, for those inputs write conditions

Eg:

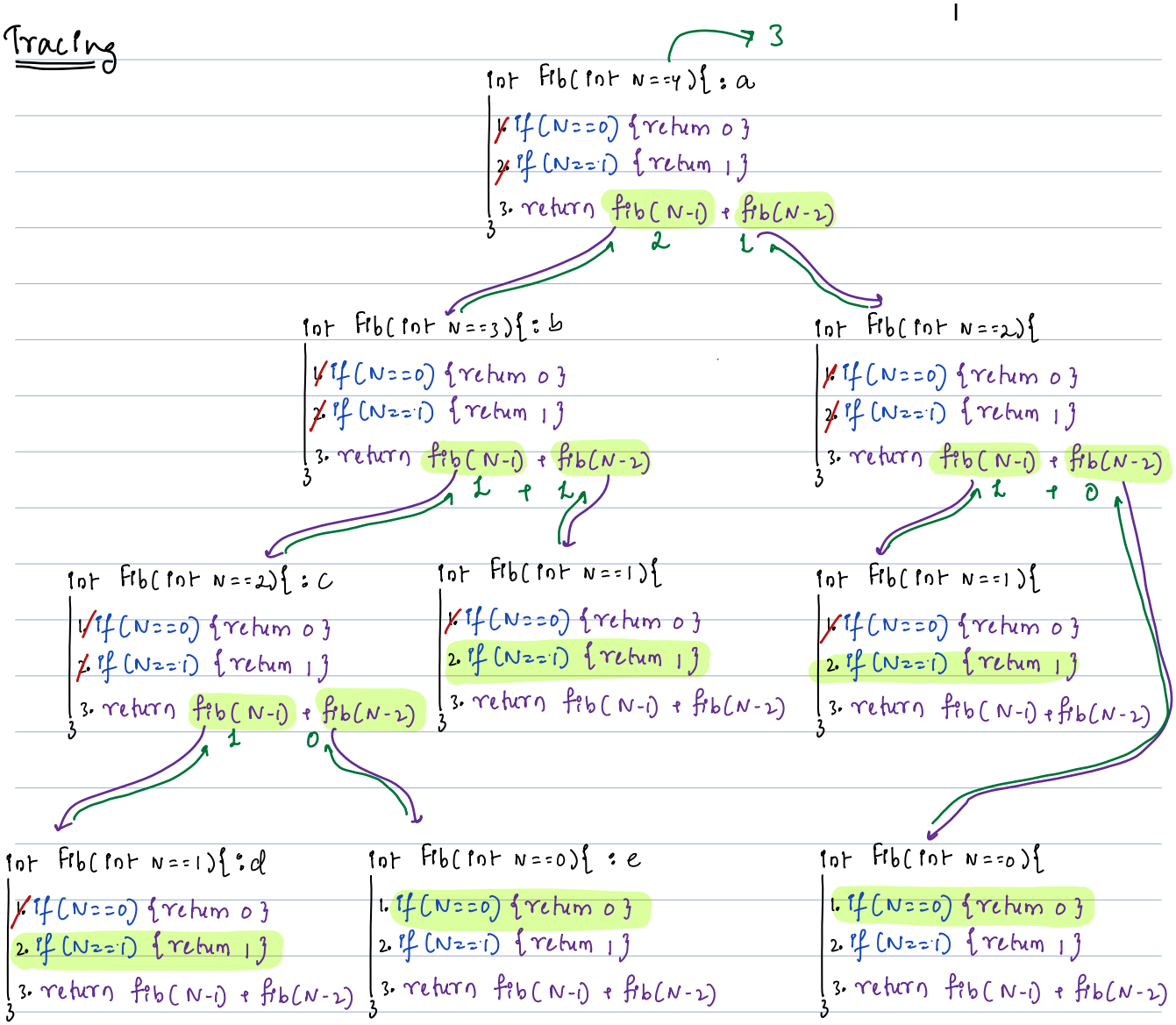
Fib(N) = Fib(N-1) + Fib(N-2)

Fib(0) = Fib(-1) + Fib(-2) * for input 0, it fails } BaseConditions

Fib(1) = Fib(0) + Fib(-1) * for input 1, it fails }

Fib(2) = Fib(1) + Fib(0) ✓ valid

# Tracing


int Fib(int N==4){ : a
1. if(N==0){return 0}
2. if(N==1){return 1}
3. return fib(N-1) + fib(N-2)
3
2
1
3
int Fib(int N==3){ : b
1. if(N==0){return 0}
2. if(N==1){return 1}
3. return fib(N-1) + fib(N-2)
1 + 1
int Fib(int N==2){
1. if(N==0){return 0}
2. if(N==1){return 1}
3. return fib(N-1) + fib(N-2)
1 + 0
int Fib(int N==2){ : c
1. if(N==0){return 0}
2. if(N==1){return 1}
3. return fib(N-1) + fib(N-2)
1
0
int Fib(int N==1){
1. if(N==0){return 0}
2. if(N==1){return 1}
3. return fib(N-1) + fib(N-2)
int Fib(int N==1){
1. if(N==0){return 0}
2. if(N==1){return 1}
3. return fib(N-1) + fib(N-2)
int Fib(int N==1){ : d
1. if(N==0){return 0}
2. if(N==1){return 1}
3. return fib(N-1) + fib(N-2)
int Fib(int N==0){ : e
1. if(N==0){return 0}
2. if(N==1){return 1}
3. return fib(N-1) + fib(N-2)
int Fib(int N==0){
1. if(N==0){return 0}
2. if(N==1){return 1}
3. return fib(N-1) + fib(N-2)

5Q8: Given N, print all numbers from 1...N in Inc order

Inc(5) = 1 2 3 4 5

Inc(5) = Inc(4) Print(5) // Sub Problem: Inc(4)

Inc(N) = 1 2 3 4 ... N-1 N

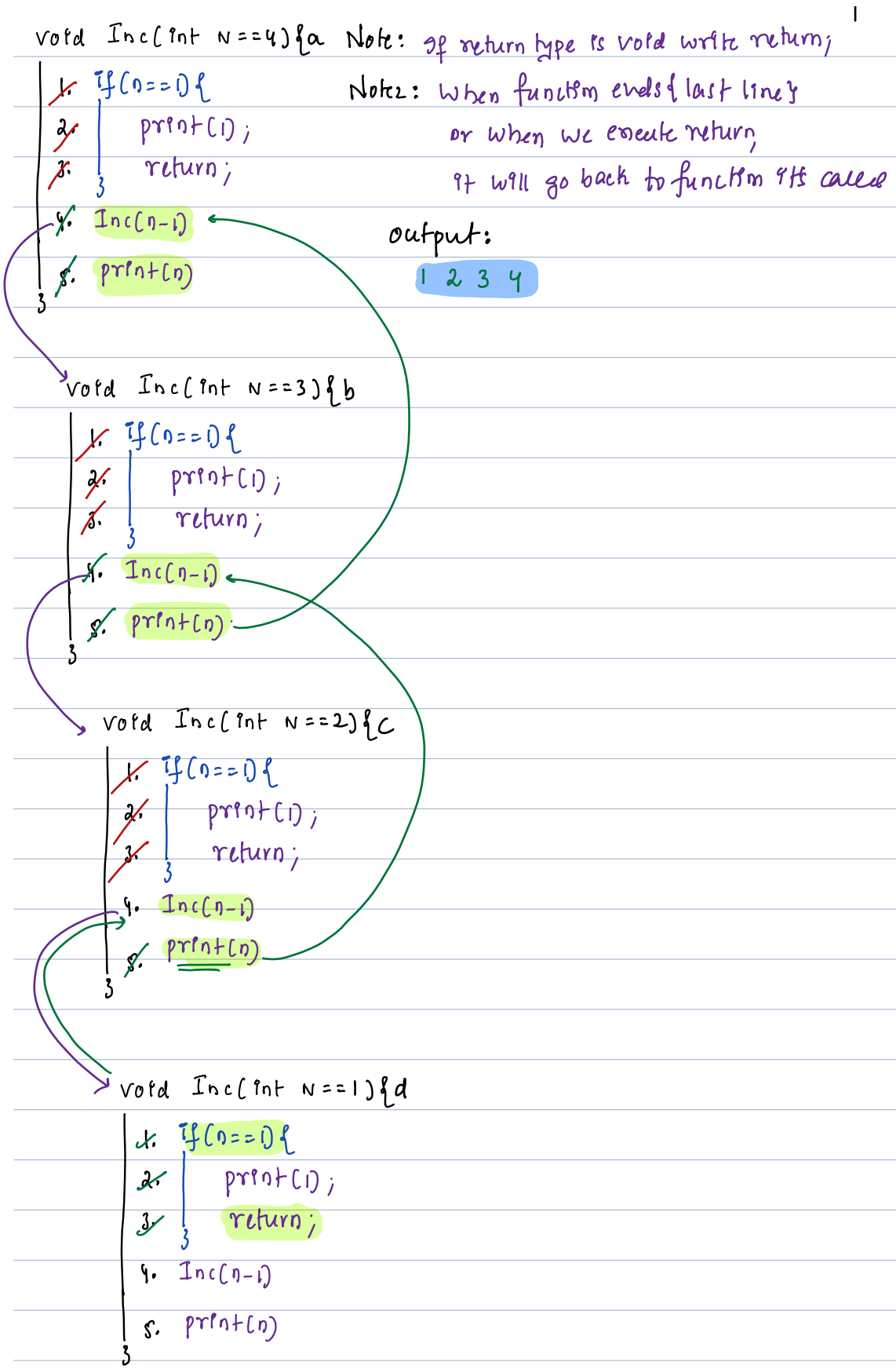
Inc(N) = Inc(N-1) Print(N) // Sub Problem: Inc(N-1)

Input: $1 <= N <= 50$;

```
void Inc(int N){ Ass: Given N print all numbers from 1..N
1. if(N==1){
2.     print(1)
3. }   return;
4. Inc(N-1)
5. print(N)
}
```

Base: Inc(int N=1){ ✗ Inc(int N=2){ ✓

```
Inc(0)          Inc(1)
print(N)        print(N)
}               }
```




Note: If return type is void write return;

Note2: When function ends { last line }
or when we execute return,
it will go back to function it's called

output:

1 2 3 4

GQ: Given N, print all numbers from N...1 in desc order. TODO

Dec(5) = 5 4 3 2 1

Dec(5) = print(5) dec(4)  Dec(5) = Dec(4) print(1) ?

Dec(N) = N N-1 N-2 ... 2 1

Dec(N) = print(N) dec(N-1)

Input: $1 <= N <= 50$;

```
void Dec(int N){ Ass:
|
}
```

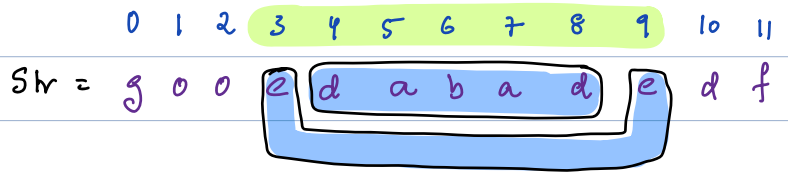
Given N _____

Dec(4):

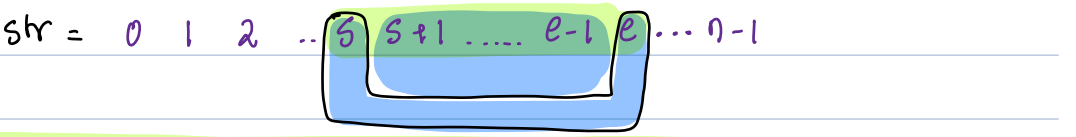
Dec(6):

Q8. Given a substring, check if its palindrome or not?

↳ continous part of string ≈ subarray. : True / False




ispal(str 3 9) = str[3] == str[9] && ispal(str, 4, 8) // sub: ispal(str, 4, 8)




s ≤ e

ispal(str, s, e) = str[s] == str[e] && ispal(str, s+1, e-1) // sub: ispal(str, s+1, e-1)

Ass: Given str, s, e check if substring from [s..e] is palindrome or Not?

```
boolean ispal(char str[], int s, int e){
    1. if(s > e){return true}
    2. if(str[s] == str[e] &&
                ispal(str, s+1, e-1))
    3.     return true
       }
    4. else{return false}
}
```

```
          0 1 2 3 4
str[] :   m a d a m     s = 0   e = 4
```

True:

```
boolean ispal (char str[], int s=0, int e=4) {a
    1. if (s > e) { return true }
    2. if (str[s] == str[e] && ispal(str, s+1, e-1))
    3.     return true
       }
    4. else { return false }
}
```

True

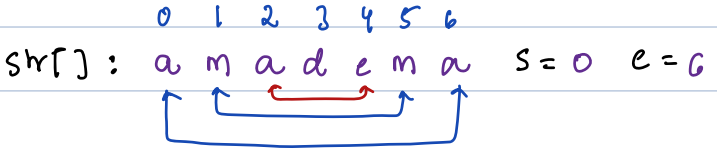
```
boolean ispal (char str[], int s=1, int e=3) {b
    1. if (s > e) { return true }
    2. if (str[s] == str[e] && ispal(str, s+1, e-1))
    3.     return true
       }
    4. else { return false }
}
```

True

```
boolean ispal (char str[], int s=2, int e=2) {c
    1. if (s > e) { return true }
    2. if (str[s] == str[e] && ispal(str, s+1, e-1))
    3.     return true
       }
    4. else { return false }
}
```

True

```
boolean ispal (char str[], int s=3, int e=1) {d
    1. if (s > e) { return true }
    2. if (str[s] == str[e] &&
                ispal(str, s+1, e-1))
    3.     return true
       }
    4. else { return false }
}
```




→ False:

```
boolean ispal(char str[], int s=0, int e=6){a
    1. if(s > e){return true}
    2. if(str[s] == str[e] && ispal(str, s+1, e-1))
    |      return true
    3. }
    4. else{return false}
}
```

```
boolean ispal(char str[], int s=1, int e=5){b
    1. if(s > e){return true}
    2. if(str[s] == str[e] && ispal(str, s+1, e-1))
    |      return true
    3. }
    4. else{return false}
}
```

```
boolean ispal(char str[], int s=2, int e=4){c
    1. if(s > e){return true}
    2. if(str[s] == str[e] && ispal(str, s+1, e-1))
    |      return true
    3. }
    4. else{return false}
}
```