Binary search

No. of elements = n

$2^{level} = n$

$level = \frac{\log n}{\log 2}$

$comparisions = \frac{\log n}{\log 2}$

$Time \propto \frac{\log n}{\log 2}$

$T(n) = O(\log n)$

Best case — if key is found at first few middle position — O(1)

Avg case — if key is found at middle level

Worst case — if key is found at last few middle positions

O(log n)

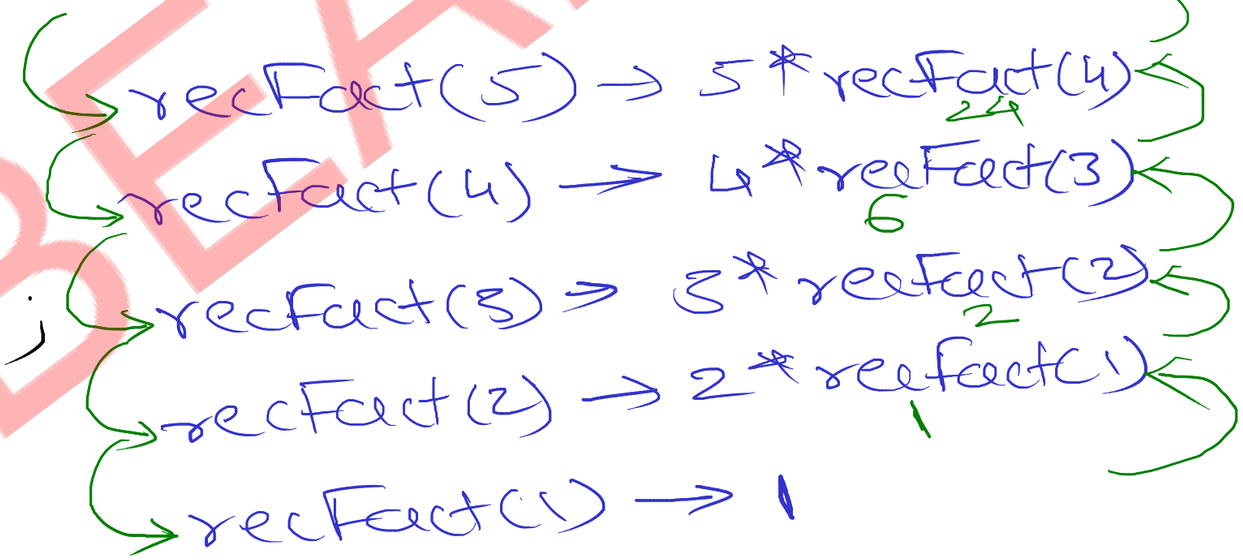
Recursion

- calling same function within function itself
- when we can define mathematical formula/process in terms of itself
- you should have some terminating condition

$$n! = n * (n-1)! \quad | \quad 1 \text{ or } 0! = 1$$

```
int recfact(int n){
    if(n==1)
        return 1;
    return n * recfact(n-1);
}

int main(){
    int n=5;
    recfact(n);
    return 0;
}
```



Algorithm Implementation

Iterative approch

loops are used

```
int fact(int n) {
    int fact = 1;
    for(i=1; i<=n; i++)
        fact *= i;
    return fact;
}
```

- count number of iterations

$T(n) = O(n)$

Recursive approch

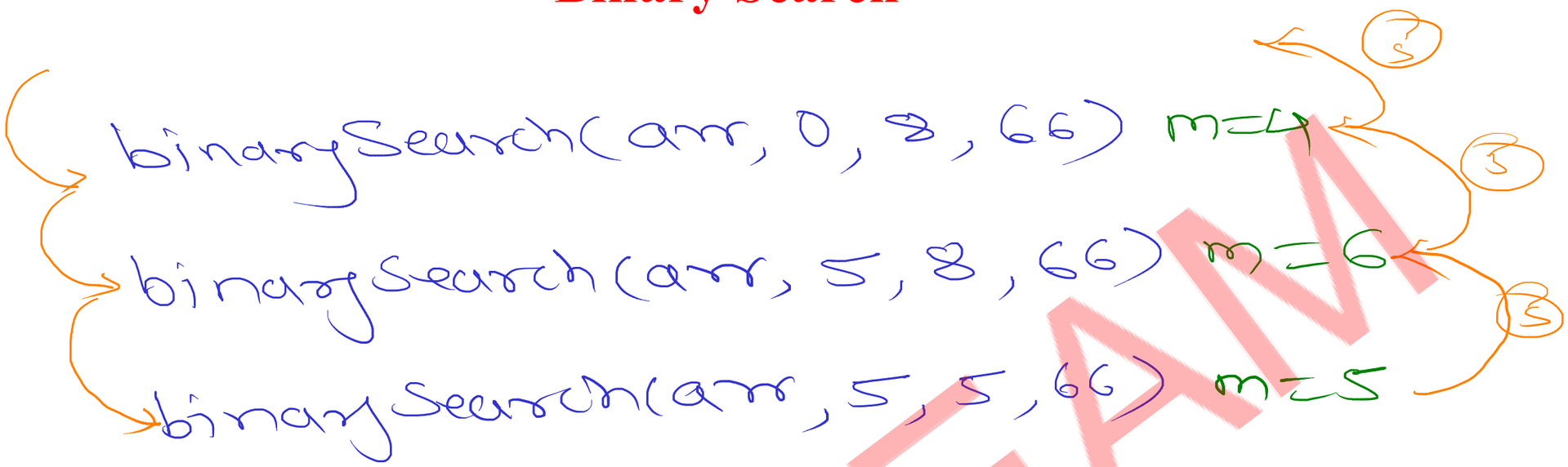
recursion

```
int recfact(int n) {
    if (n==1)
        return 1;
    return n * recfact(n-1);
}
```

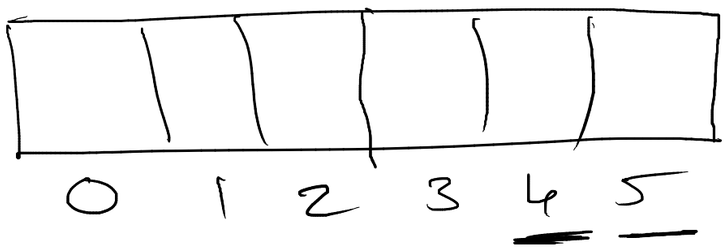
- count number of recursive calls

$T(n) = O(n)$

Binary Search



Selection Sort



$n = 6$

$i = 0, 1, 2, 3, 4$

`i=0; i<n-1; i++`

`j=i+1; j<n; j++`

No. of passes $= n-1$

No. of comparisions $= (n-1)+(n-2)+(n-3) \ldots +2+1$

$= 1+2+3+ \ldots \ldots n$

$= \frac{n(n+1)}{2}$

$= \frac{n^2+n}{2}$

$n^2+n \leftarrow$ mathematical polynomial/function

degree → highest power of variable

degree = 2

$Time \propto \frac{n^2+n}{2}$

n	n^2
1	1
10	100
100	10000

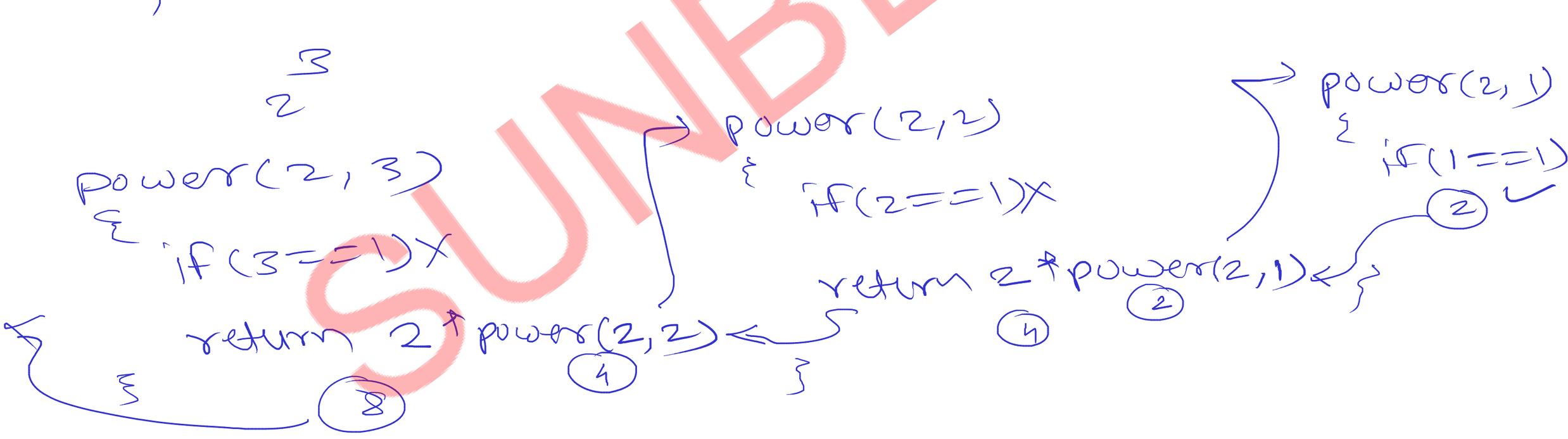
Best
Avg
Worst

$T(n) = O(n^2)$

$base^{index} = base * base^{index-1}$ index==1 → base

Power Using Recursion

```
power(base, index) {
    if(index==1)
        return base;
    return base * power(base, index-1);
}
```



Print in Binary using Recursion

```
printbinary(n) {
    if(n==0)
        return;
    printbinary(n/2);
    sysout(n%2);
}
```

9
4
2
1
0

1
0
0
1