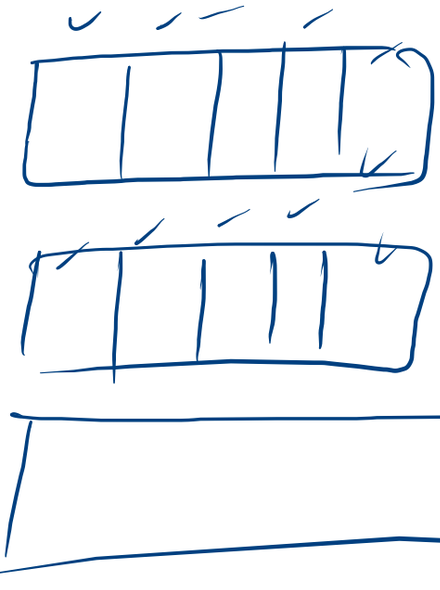
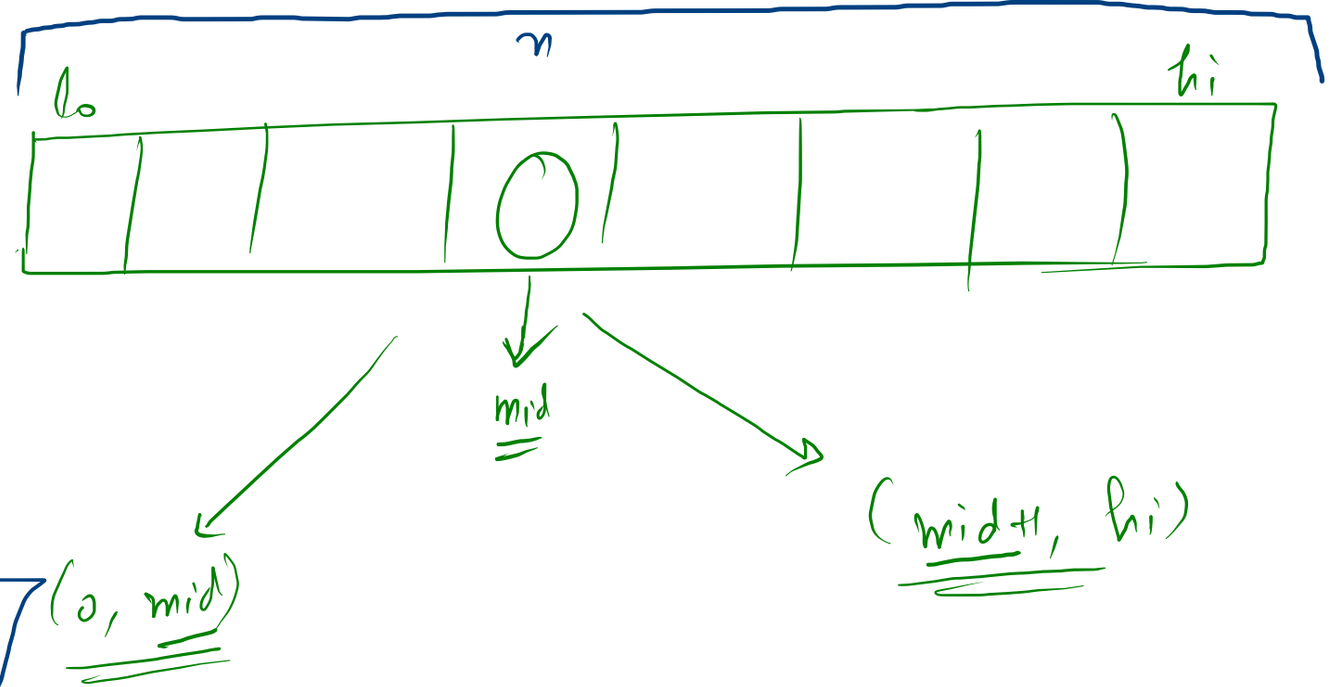
→ merge Sort

→ Quick Sort

→ Quick Select






```java
public static int[] mergeSort(int[] arr, int lo, int hi) {
  if(lo == hi){
    int b[] = new int[1];
    b[0] = arr[lo];
    return b;
  }
  int mid = (lo+hi)/2;
  int lp[] = mergeSort(arr,lo,mid);
  int rp[] = mergeSort(arr,mid+1,hi);

  return mergeTwoSortedArrays(lp,rp);
}
```

O(1) → O(1) Constant time

O(1)

n/2

n/2

→ O(n)

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n$$

MERGE SORT

$n = \frac{n}{2^0}$

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

$n/4 = \frac{n}{2^2}$

$n/8 = \frac{n}{2^3}$

K times

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

$0 \to 10$ ↳ 11

$0 \to K$ ↳ K+1

$0 \to (K+1)$ ↳ K+2

$0 \to (K-1)$ ↳ K terms

$T(n) = 2 \cdot T(\frac{n}{2}) + n$

$2 \cdot T(\frac{n}{2}) = 4 \cdot T(\frac{n}{4}) + n$

$4 \cdot T(\frac{n}{4}) = 8 \cdot T(\frac{n}{8}) + n$

$T(1) \Rightarrow$

$n_1 + n_2 = n_1 + n_2$

$n_1 = n_2$

$T(n) = n + n + n + \ldots\ldots$ (K times)

$T(n) = n.K$

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

$2^{K-1} = n$

taking $\log_2$

$K-1 = \log_2 n$

$K = \log_2 n + 1$

$T(n) = n[\log_2 n + 1]$

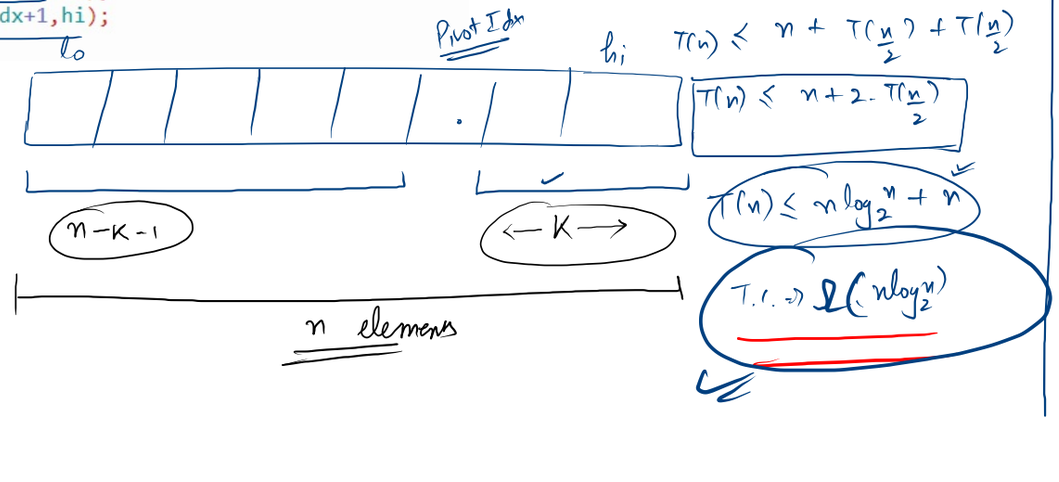
$T(n) = n \log_2 n + n$

T.C. $\Rightarrow O(n \log_2 n)$

```java
public static void quickSort(int[] arr, int lo, int hi) {
  if(lo > hi){
      return;
  }
  int pivot = arr[hi];

  int pivotIdx = partition(arr,pivot,lo,hi);
  quickSort(arr,lo,pivotIdx-1);
  quickSort(arr,pivotIdx+1,hi);
}
```

$O(n)$

```
while(i < len) {
  if(arr[i] <= val){
    Swap
    i++;
    j++;
  } else {
    i++;
  }
}
```



$T(n) \Rightarrow n + T(n-k-1) + T(k)$

B.C. $\rightarrow$ $k = n/2$

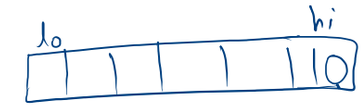
$T(n) = n + T(\frac{n}{2}-1) + T(\frac{n}{2})$

$T(n) \leq n + T(\frac{n}{2}) + T(\frac{n}{2})$

$T(n) \leq n + 2 \cdot T(\frac{n}{2})$

$T(n) \leq n \log_2 n + n$

T.C. $\Rightarrow \Omega(n \log_2 n)$



Sorted Array

W.C. $k = 0$

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

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

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

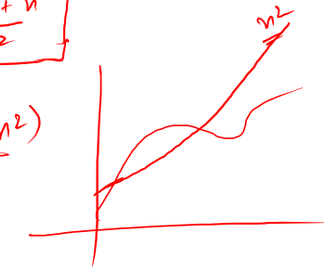
$T(1) =$

$T(n) = n + (n-1) + (n-2) + \ldots 3 + 2 + 1 + 0$

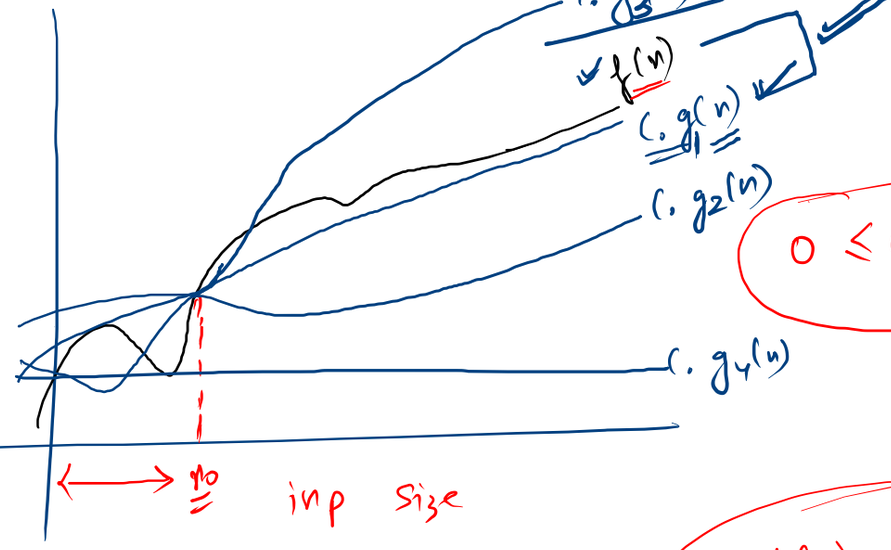
$T(n) \Rightarrow \frac{n(n+1)}{2}$

$T(n) = \frac{n^2 + n}{2}$

T.C. $\Rightarrow O(n^2)$

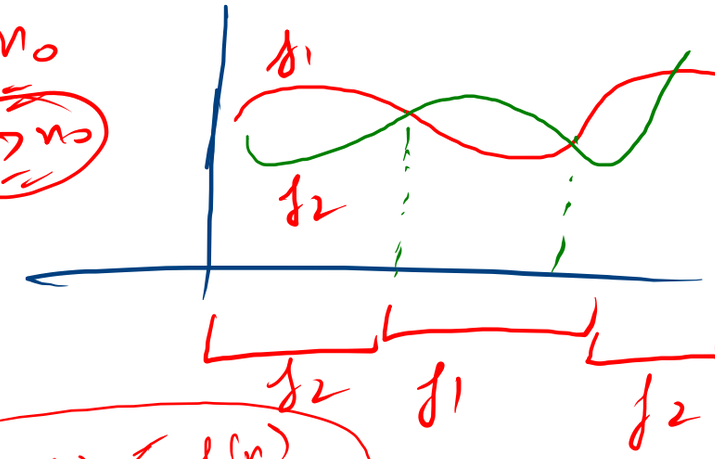


$\Omega \Rightarrow$ tightest lower bound



n_0

$n \geq n_0$



$0 \leq c \cdot g(n) \leq f(n)$

$n \geq n_0$

$f(n) = \Omega(g(n))$

$n > n_0$

$f(n) = 8n^2 + 24n + 3$

$g_1(n) = n^2$

$g_2(n) = n$

$g_3(n) = n^3$

$g_4(n) = 1$

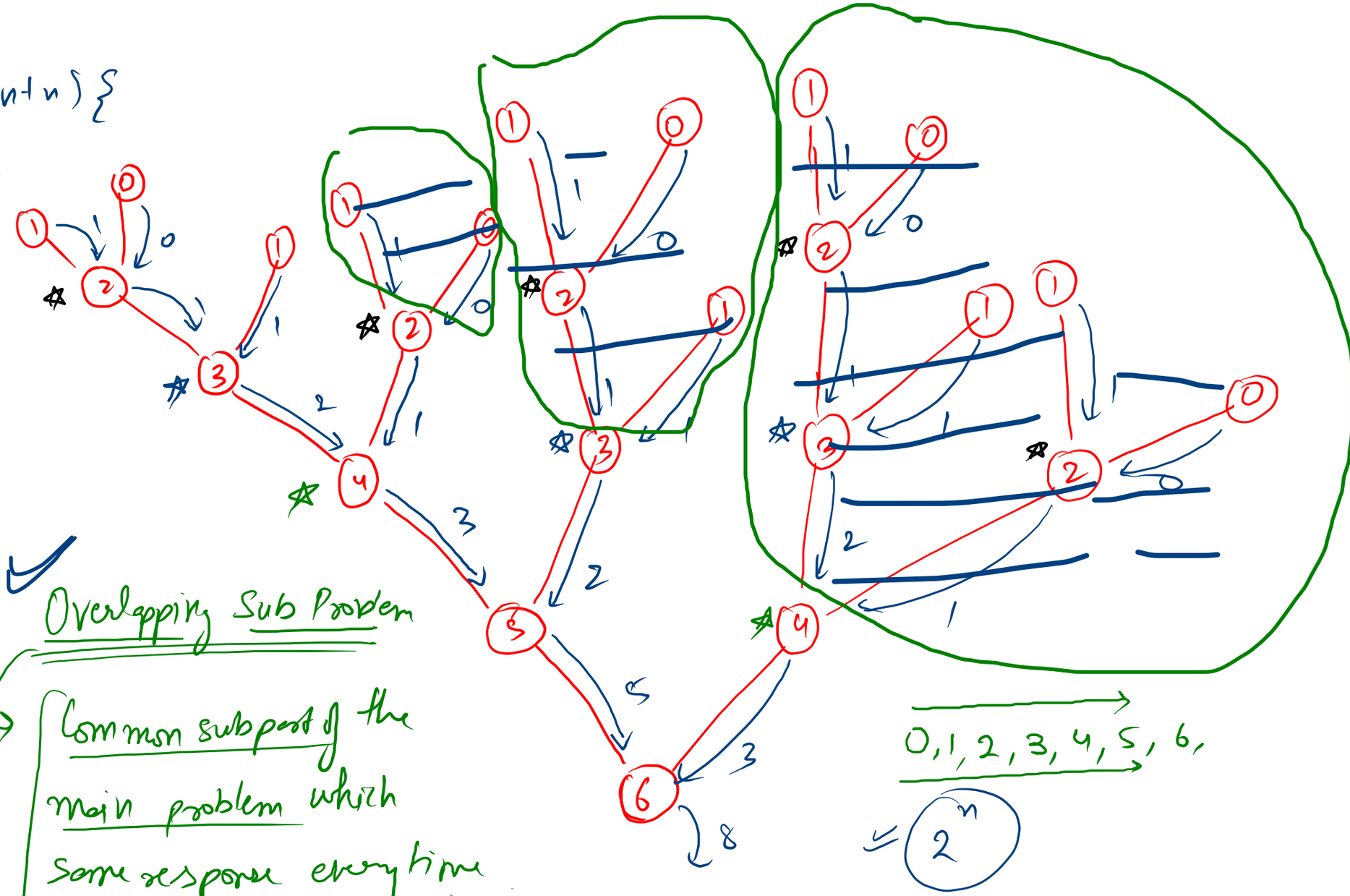
T 1. $f(n)$ is $\Omega(g_1(n))$ $\quad 8n^2 + 24n + 3 \geq 1 \cdot n^2$

T 2. $f(n)$ is $\Omega(g_2(n))$

F 3. $f(n)$ is $\Omega(g_3(n))$ $\quad 8n^2 + 24n + 3 \geq n^3$ ✗

T 4. $f(n)$ is $\Omega(g_4(n))$ $\quad 8n^2 + 24n + 3 \geq 1$

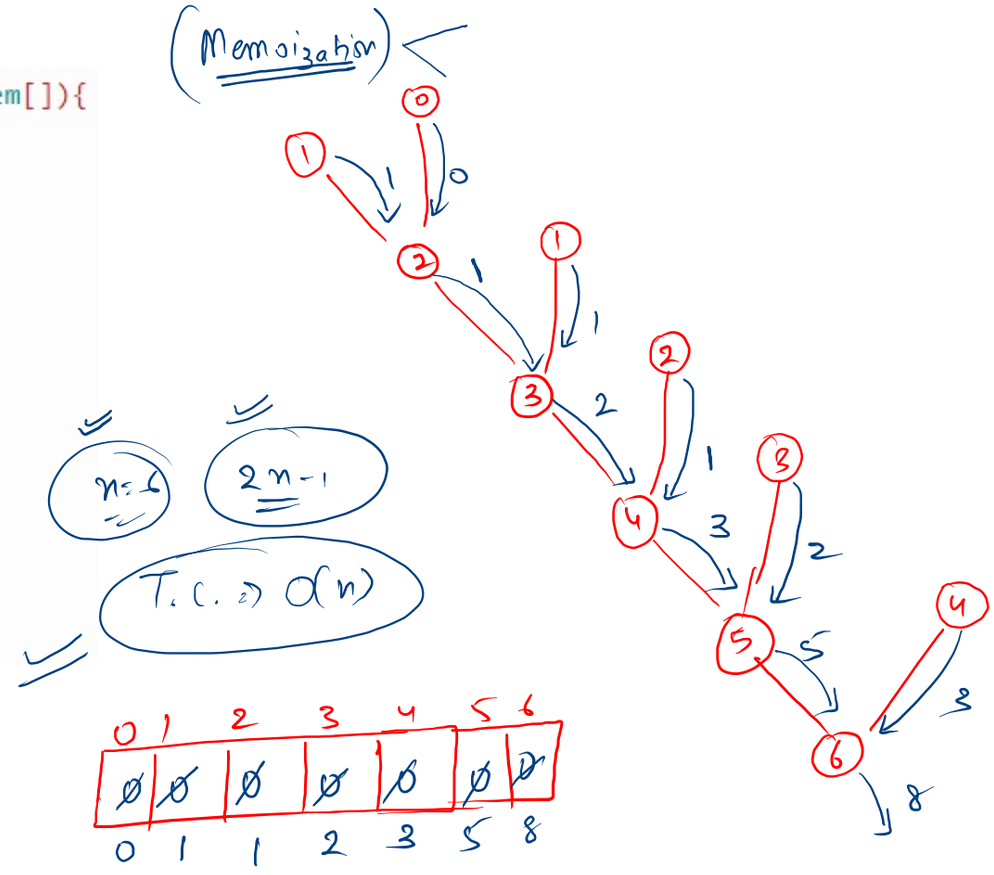
```
public static int fib(int n){
  if(n==0 || n==1){
    return n;
  }
  fibNm1 ← fib(n-1)
  fibNm2 ← fib(n-2)
  return fibNm1 + fibNm2
}
```

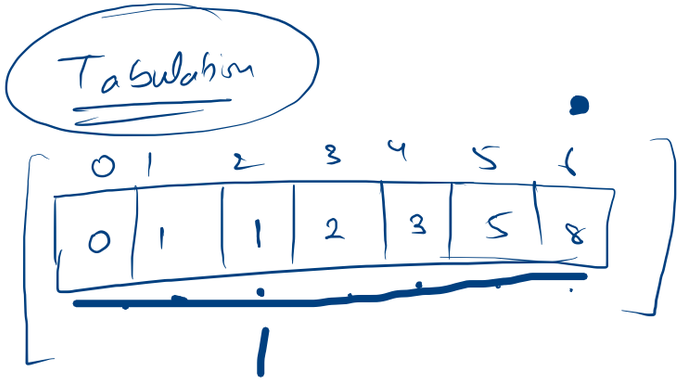


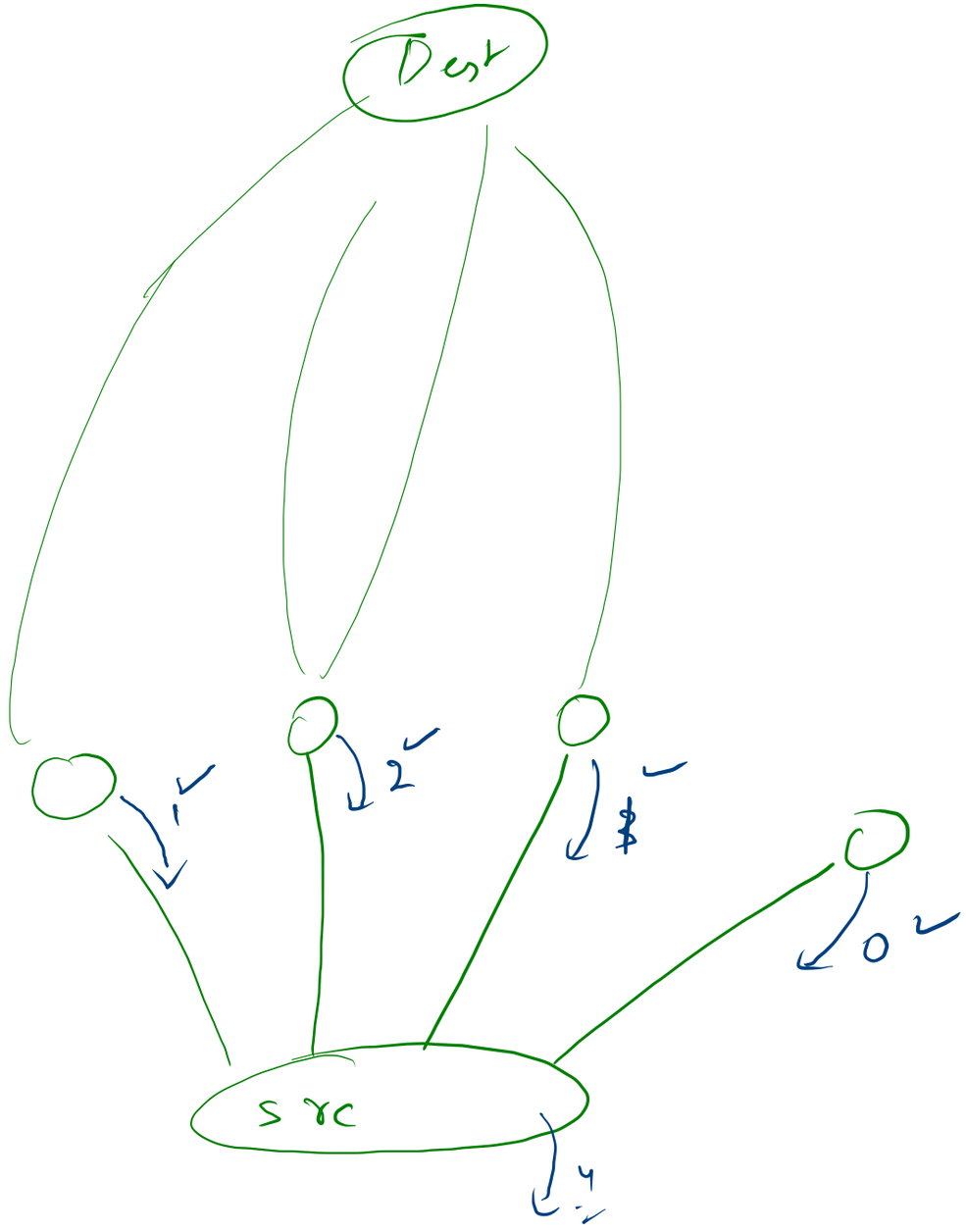
Overlapping Sub Problem

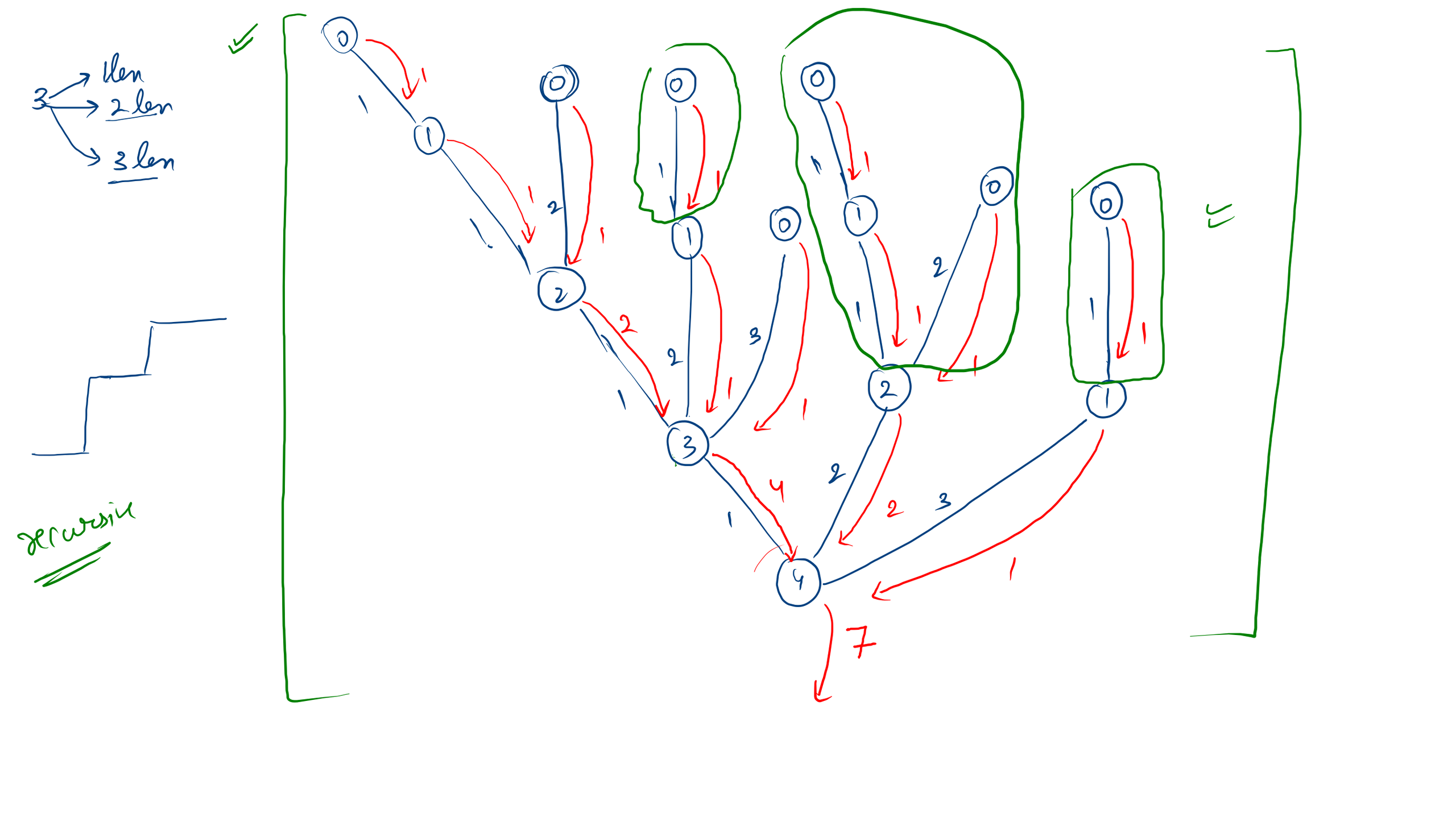
Common subpart of the main problem which same response everytime

```java
public static int fib1(int n,int mem[]){
    if(n == 1 || n == 0){
        return mem[n] = n;
    }

    if(mem[n] != 0){
        return mem[n];
    }

    int fibNm1 = fib1(n-1,mem);
    int fibNm2 = fib1(n-2,mem);

    int fibN = fibNm1+fibNm2;

    return mem[n] = fibN;
}
```





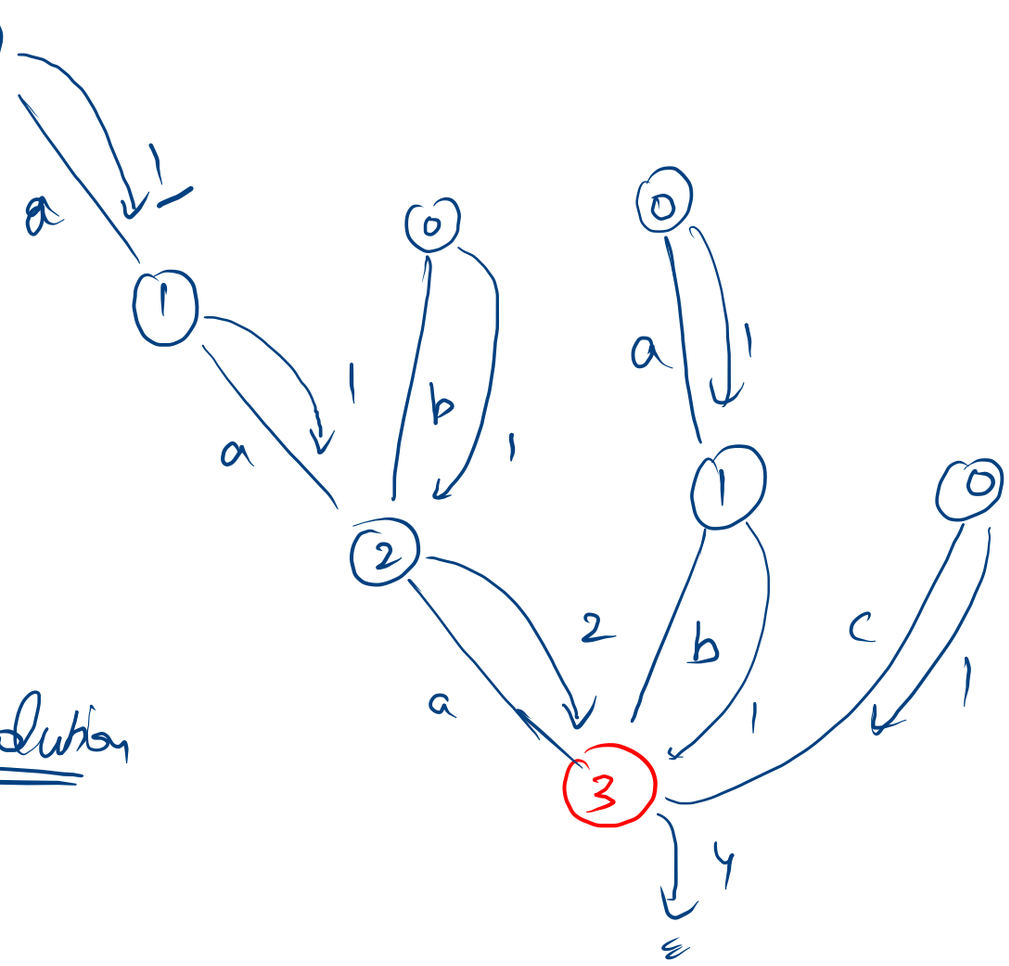

Dest
1
2
3
4
5
src


1 len
2 len
3 len
рекурсия
7

n = 3

```java
public static int climbStairsRec(int n){
    if(n == 0){
        return 1;
    }

    int tways = 0;

    if(n-1 >= 0){
        tways += climbStairsRec(n-1); → a
    }
    if(n-2 >= 0){
        tways += climbStairsRec(n-2); → b
    }
    if(n-3 >= 0){
        tways += climbStairsRec(n-3); → c
    }

    return tways;
}
```

1) Recursion
2) Memoization
3) Tabulation



Order of solution

0, 1, 2, 3

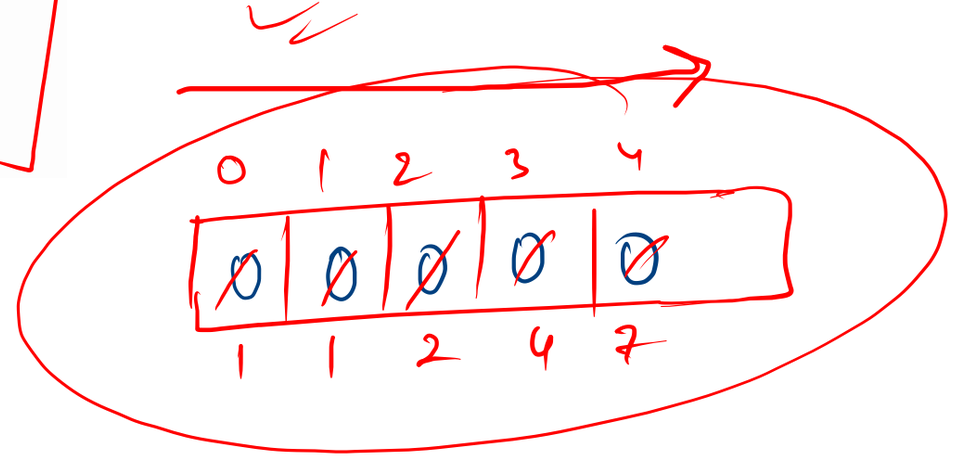
Mem = 4

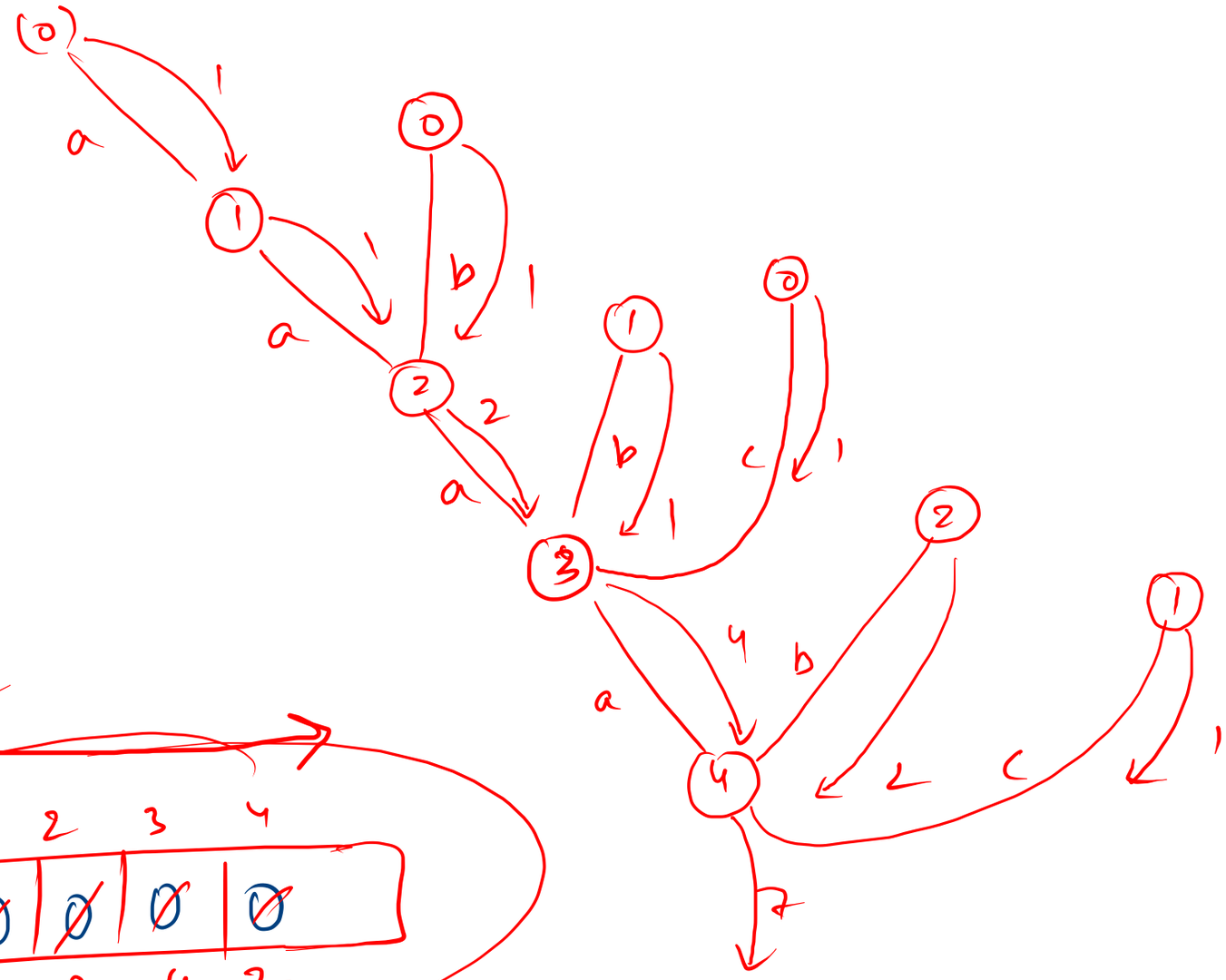
```java
public static int climbStairsM(int n,int mem[]){
    if(n == 0){
        return mem[n] = 1;
    }

    if(mem[n] != 0){
        return mem[n];
    }

    int tways = 0;

    if(n-1 >= 0){
        tways += climbStairsM(n-1,mem);
    }
    if(n-2 >= 0){
        tways += climbStairsM(n-2,mem);
    }
    if(n-3 >= 0){
        tways += climbStairsM(n-3,mem);
    }

    return mem[n] = tways;
}
```

a

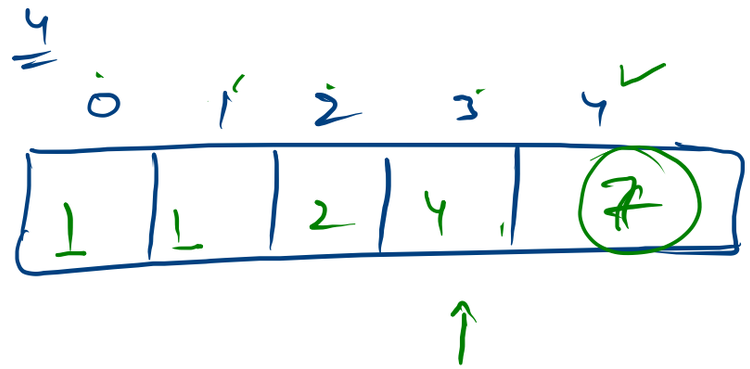
b

c



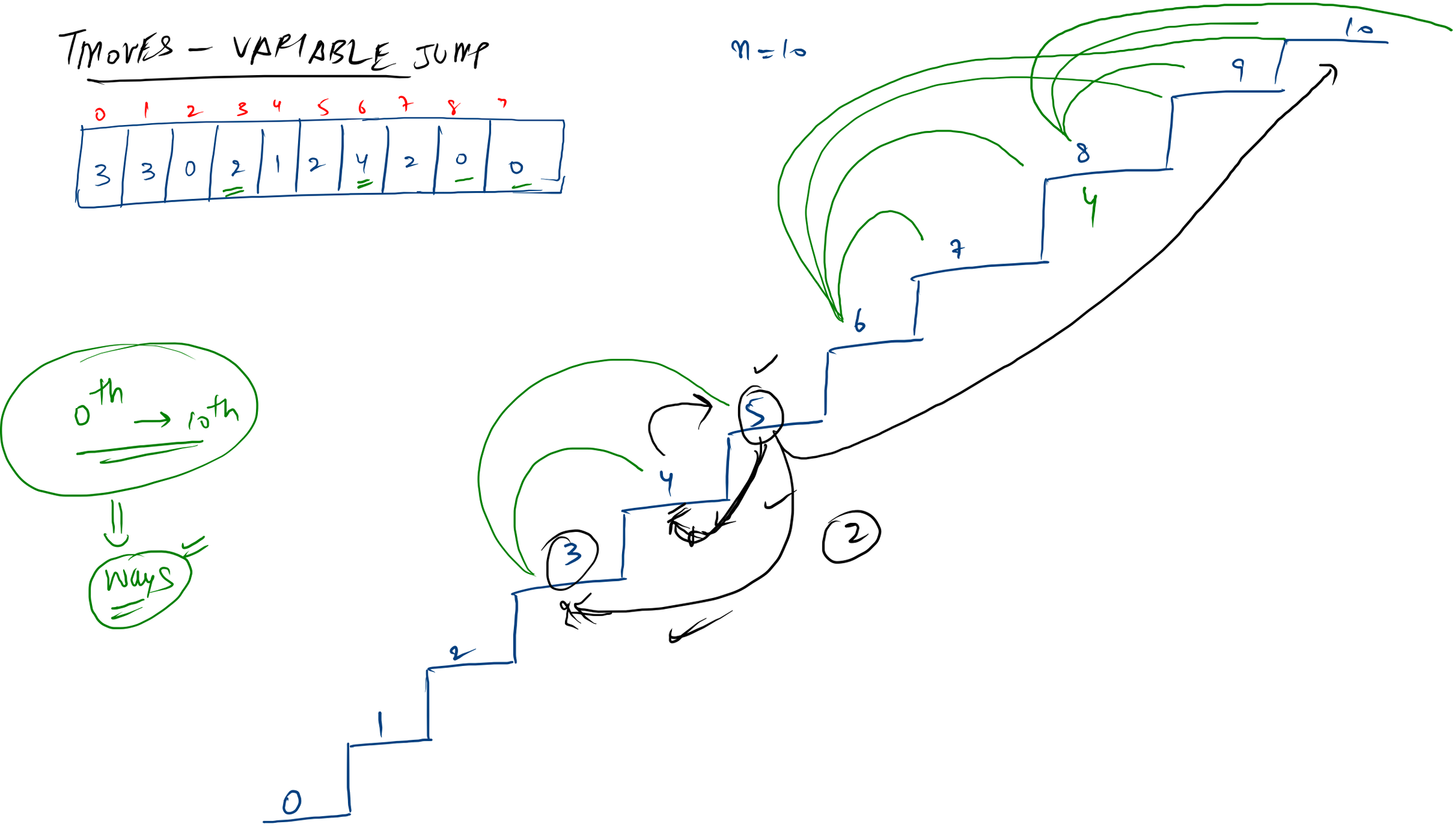


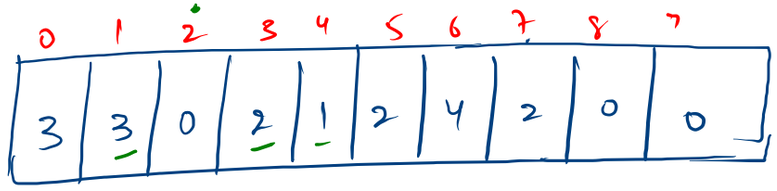
```java
public static int climbStairsT(int n){
    int mem[] = new int[n+1];

    for(int i = 0 ; i <= n ; i++){
        if(i == 0){
            mem[i] = 1;
            continue;
        }

        int tways = 0;

        if(i-1 >= 0){
            tways += mem[i-1];
        }
        if(i-2 >= 0){
            tways += mem[i-2];
        }
        if(i-3 >= 0){
            tways += mem[i-3];
        }

        mem[i] = tways;
    }

    return mem[n];
}
```



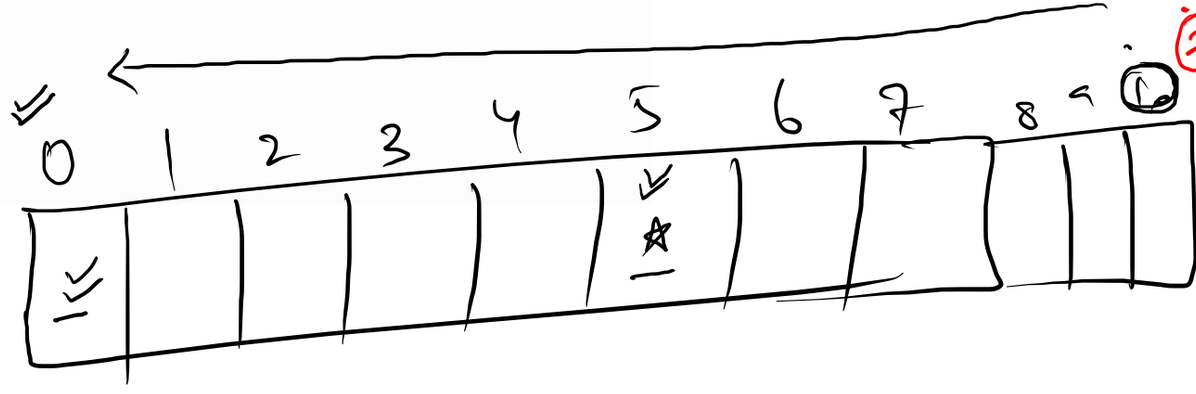
1 MOVES – VARIABLE JUMP

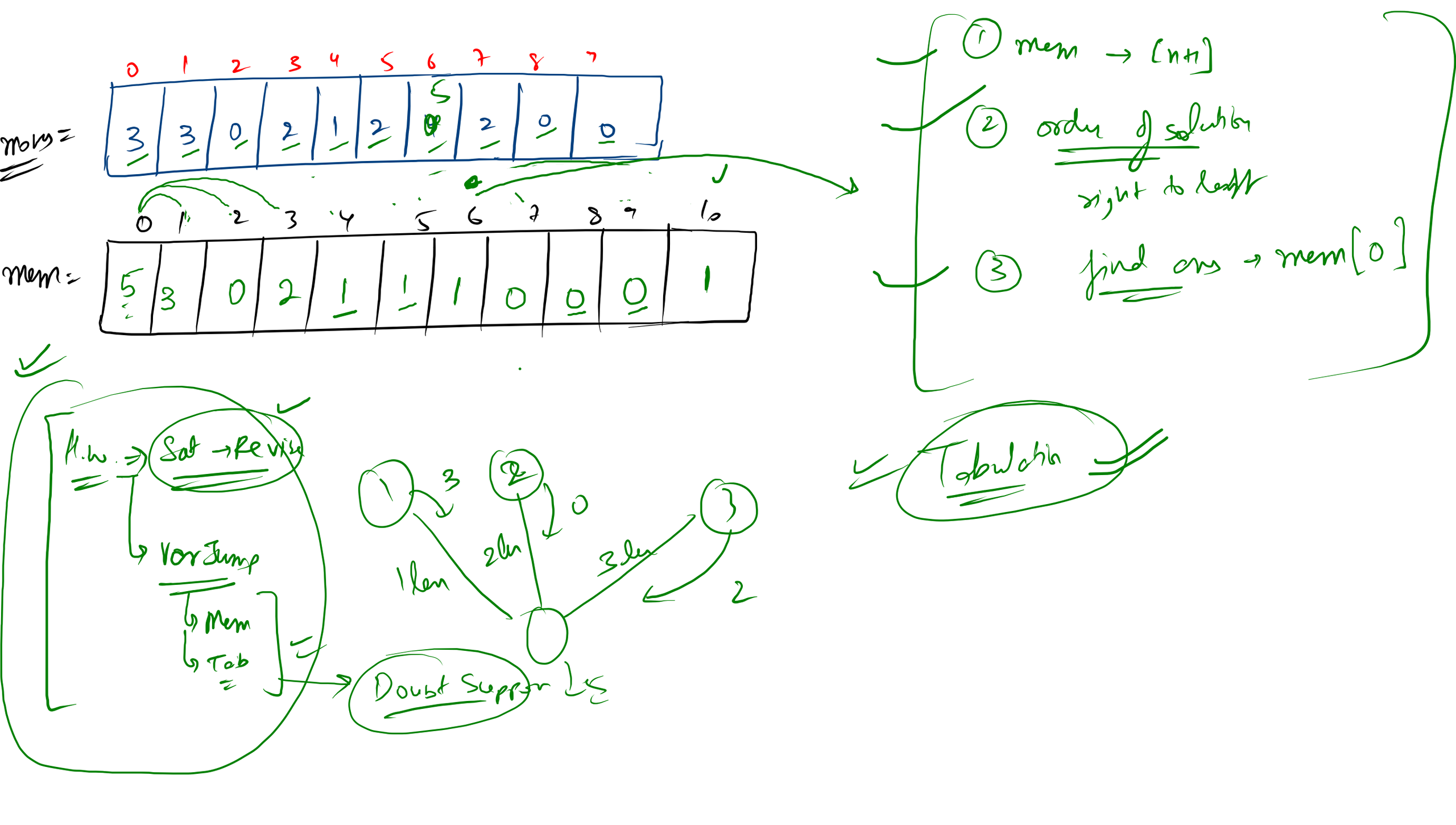
0	1	2	3	4	5	6	7	8	9
3	3	0	2	1	2	4	2	0	0

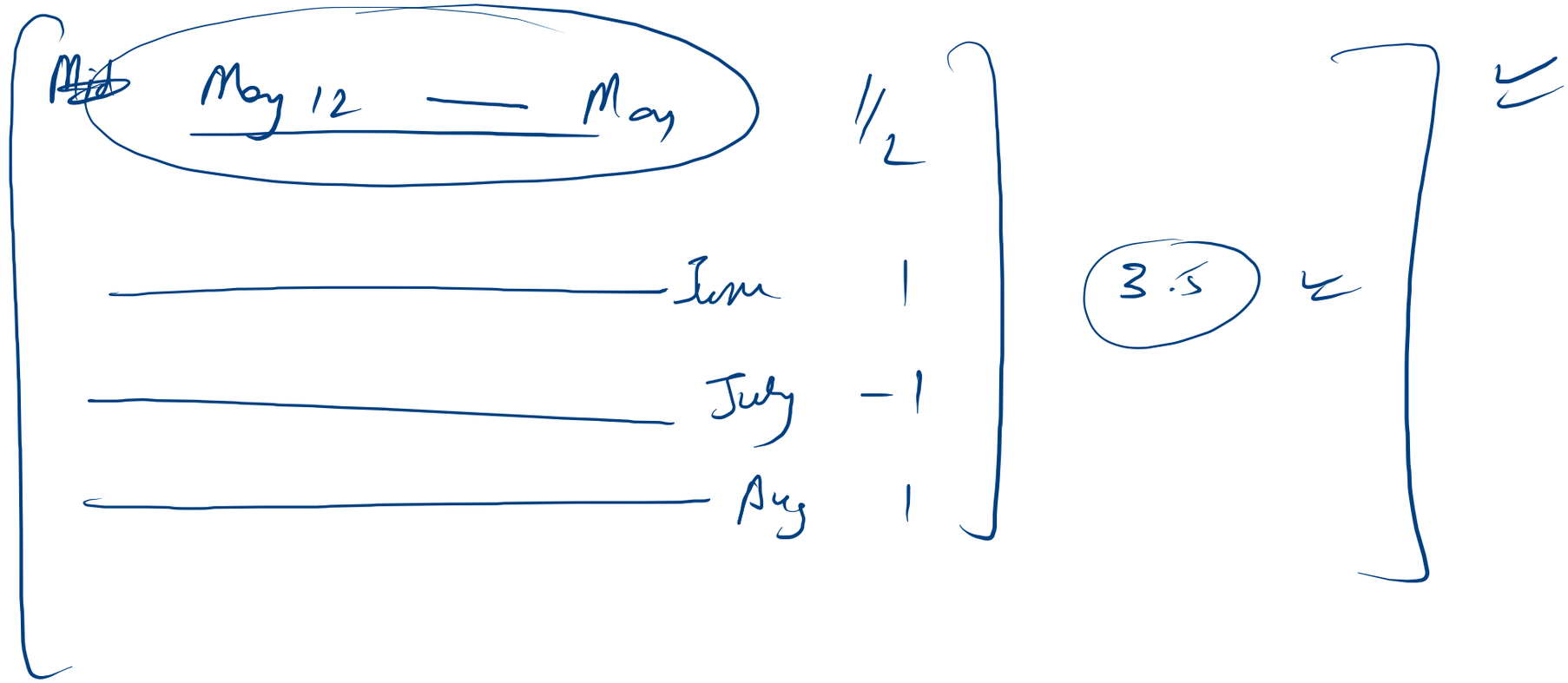





```java
public static int ClimbStairsVarJumpRec(int i,int moves[]){
    if(i == moves.length){
        return 1;
    }
    int tways = 0;
    int maxJump = moves[i];
    for(int len = 1 ; len <= maxJump ; len++){
        if(i+len <= moves.length){
            tways += ClimbStairsVarJumpRec(i+len,moves);
        }
    }

    return tways;
}
```




0 1 2 3 4 5 6 7 8 9
nums =
3 3 0 2 1 2 5 2 0 0
0 1 2 3 4 5 6 7 8 9 10
mem =
5 3 0 2 1 1 1 0 0 0 1
1) mem → [n+1]
2) order of solution
right to left
3) find ans → mem[0]
Tabulation
H.w
Sol → Revise
Vor Jump
Mem
Tab
Doubt Support
1
2
3
1 len
2 len
3 len
3
0
2


May 12 — May 1/2
June 1
July 1
Aug 1
3.5