Example 1:

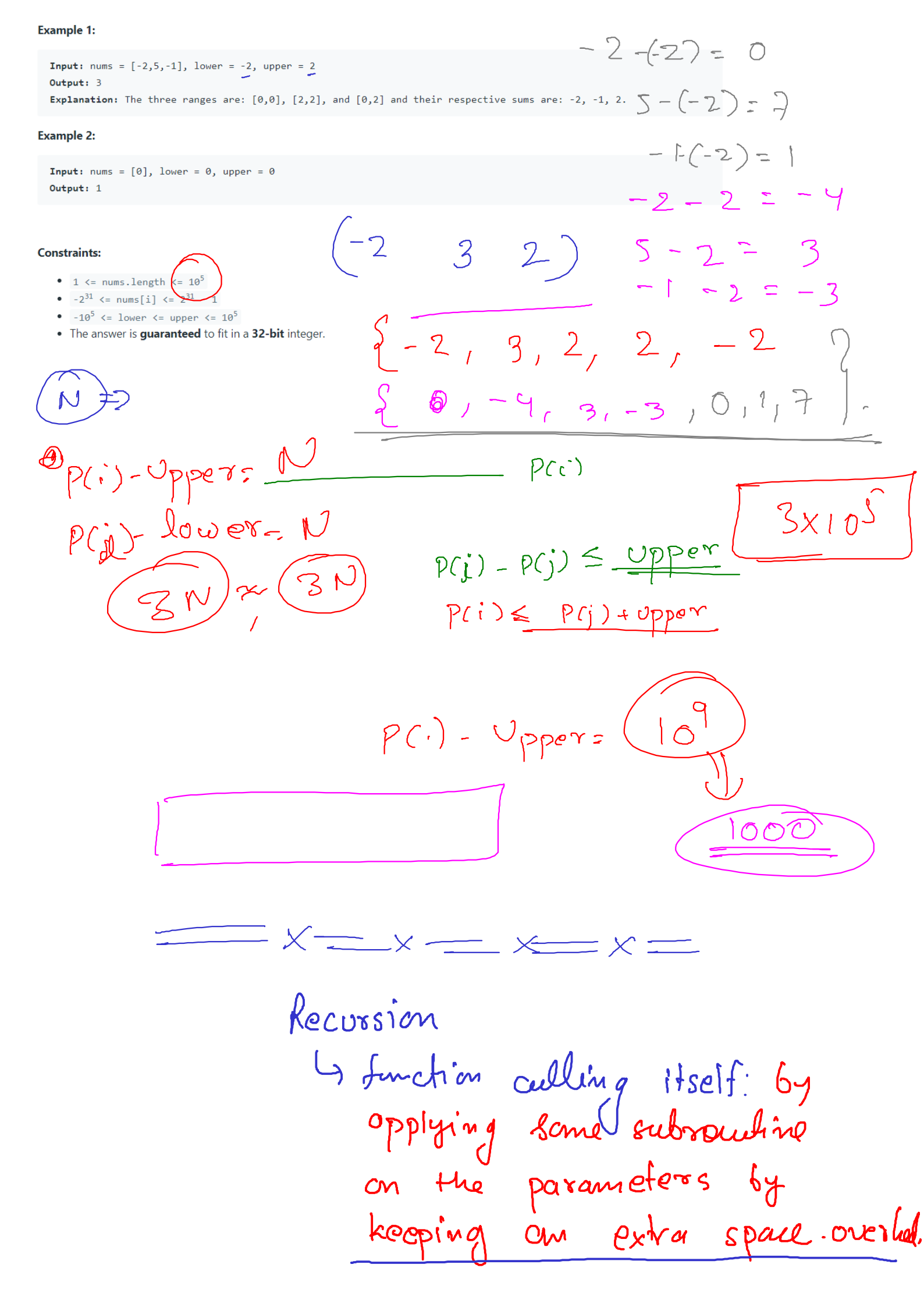
```
Input: nums = [-2,5,-1], lower = -2, upper = 2
Output: 3
Explanation: The three ranges are: [0,0], [2,2], and [0,2] and their respective sums are: -2, -1, 2.
```

Example 2:

```
Input: nums = [0], lower = 0, upper = 0
Output: 1
```

Constraints:

- `1 <= nums.length <= 10^5`
- `-2^31 <= nums[i] <= 2^31 - 1`
- `-10^5 <= lower <= upper <= 10^5`
- The answer is **guaranteed** to fit in a **32-bit** integer.



factorial: $f(n) = n!$

$f(n) = n \times f(n-1)$ ⟸ $(n-1)! \times n = n!$

$(n-1) \times f(n-2)$

$(n-2) \times f(n-3)$

$2 \times f(1)$

$1! = 1$

$f(5)$ — $5 \times 4 \times 3 \times 2 \times 1 = 120$

$5 \times f(4)$ — $4 \times 3 \times 2 \times 1$

$4 \times f(3)$ — $3 \times 2 \times 1$

$3 \times f(2)$ — 2×1

$2 \times f(1)$

bigger problem ⟹ Smaller Sub-problem.

⊙ Three Steps. $1 + 2 + 3 + \cdots N = \frac{N(N+1)}{2}$

① P(1) True. $1 = 1$ ✓

② Assum P(k) is true $1 + 2 + 3 + \cdots k = \frac{k(k+1)}{2}$

③ Proove P(k+1) is true.

LHS: $1 + 2 + 3 + 4 + \cdots k + (k+1)$

$\Downarrow$

$\frac{k(k+1)}{2} + (k+1)$

$\Rightarrow \frac{k(k+1) + 2(k+1)}{2}$

$= \frac{(k+1)(k+2)}{2} = \text{RHS}$

RHS $\frac{(k+1)(k+2)}{2}$

① Base Case: find out smallest subproblem for which we know the answer.

② Recursive task: Assume that for the given subproblem recursion will give correct answer.

③ Self work: With the help of answer of the subproblem build the answer for the current problem.

Q. $f(n) = n!$

Base Case: `if(n==1) return 1;`

Recursive task: `sub = f(n-1);`

Self work: `return n * sub;`

Time Complexity for recursive codes:-

$\sum$ No. of states $\times$ Time complexity for single state

D.P. No. of Unique States $\times$ u

Space Complexity :- Maximum space allocated at point of time while execution of code.

➡ MAXIMUM DEPTH of Recursion Tree is the space complexity.

⊙ N = 5 (1, 2, 3, 4, 5)

f(N) {1 — N}.

5 4 3 2 1

```
f(N)
{
  if(N==0) return;
  f(N-1);
  cout<< N;
}
```

```
f(N){
    if(N==0) return;
    cout<<N<<" ";
    f(N-1);
}
```

5 4 3 2 1 2 3 4 5

```
f(N1) {
    if(N==1) { cout<<'1'; return;}
    cout<< N <<" ";
    f(N-1);
    cout<< N <<" ";
}
```

⊙ Count No. of binary string of length n s.t. it doesn't contain adjacent 1s.

N=1

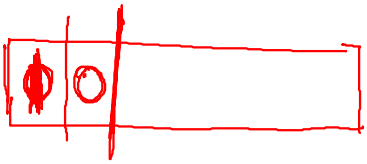
$$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

N=2

$$\begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

N=3

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$



$f(N) = f(N-1) + f(N-2)$ } Time: 2^N

DP: O(N).

```
int f (int N, int last)
    if(N==1) return 1;
    if(last==0)
        ans += f(N-1,0) + f(N-1,1);
    else {
        ans += f(N-1,0);
    return ans;
}.
```

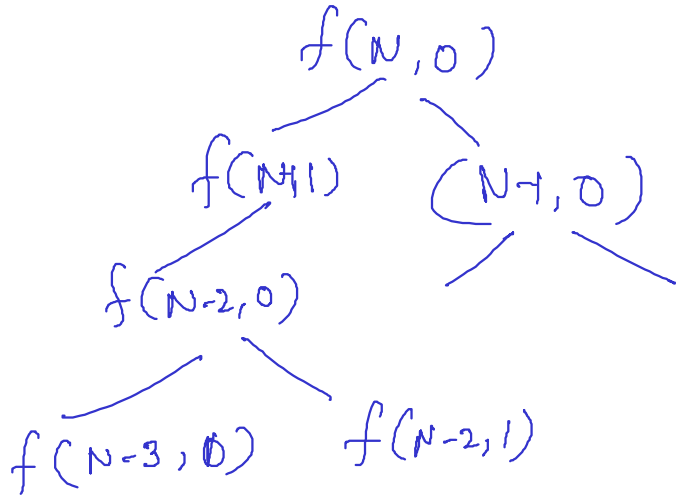
N=1 , last = 0

N=1 → 1 (last=1)

1

f(N,1) + f(N,0)

⊙.



f(1,1) + f(1,0)

2.

⊙ pairing.

N party.

i'th → solo

i'th → pair.

N=3 ABC

(A) (B) (C)

(AB) (C)

(A) (BC)

(B) (AC)

N=4 ABCD →

A B C D

(AB) C D

(A B) (CD)

(A C) B D

(A C) (B D)

(A D) B C

(A D) (B C)

(B C) A D

(B D) A C

f(N)

(N==1) return 1;

(N==2) return 2;

f(N) = f(N-1)

f(N) = f(N-1) + (N-1)·f(N-2) + (N-1)f(N-2)

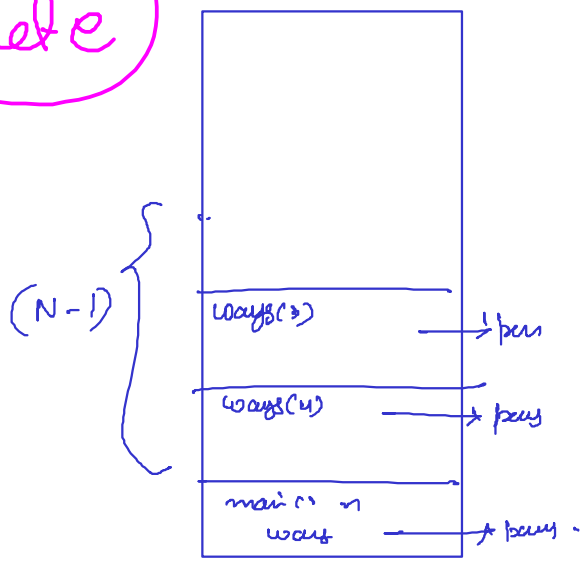
Stack

Heap → Dynamically

new delete ptr

```
int ways(int n) {
    if (n<=2) return n;
    int sub1 = ways(n-1);
    int sub2 = ways(n-2);
    return sub1 + (n-1)*sub2; }
int main() {
    int n; cin>>n;
    cout << ways(n) << endl;
}
```

n = 4



≈ O(N) : Space:

① Pow(a, b) {

```
if (b==0) return 1;
if (a==0) return 0;
int op = a * Pow(a, b-1);
return op;
}
```

a^6.

0^x

// Time : O(b)

// Log O(b)

Pow(a, b) {

```
int op = Pow(a, b/2);
op = op * op;
if (b&1) op = a * op;
return op;
}
```

3^5

$3^{5/2} = (3^2)$

$(3^2)^2 = (3^4) \cdot 3$

// Time : O(log b)

Space : O(log b)

② Pow(a, b)

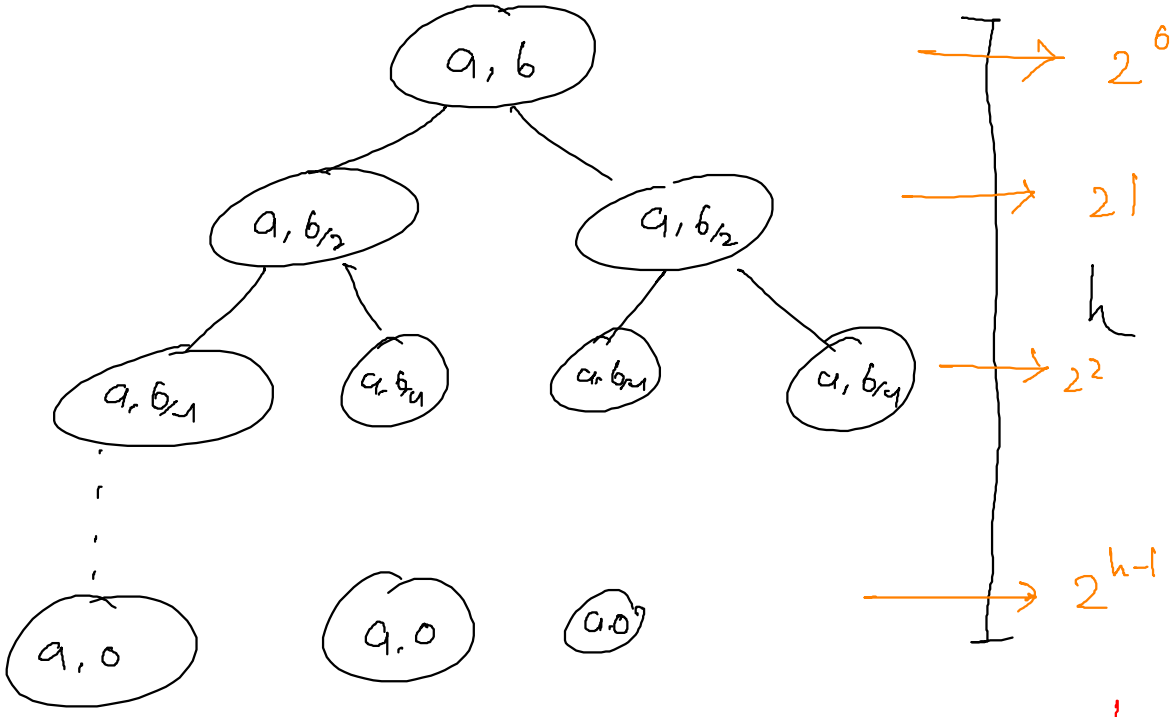
~~if (b==~~

```
int op1 = Pow(a, b/2)
int op2 = Pow(a, b/2);
int ans = 1; if (b&1) ans = a;
```

ans = ans * opt1 * opt2; Time: O(b) ✓

return ans;

}



$h = \log_2 b$

$h \Rightarrow 2^h - 1$

$= 2^{\log_2 b} - 1$

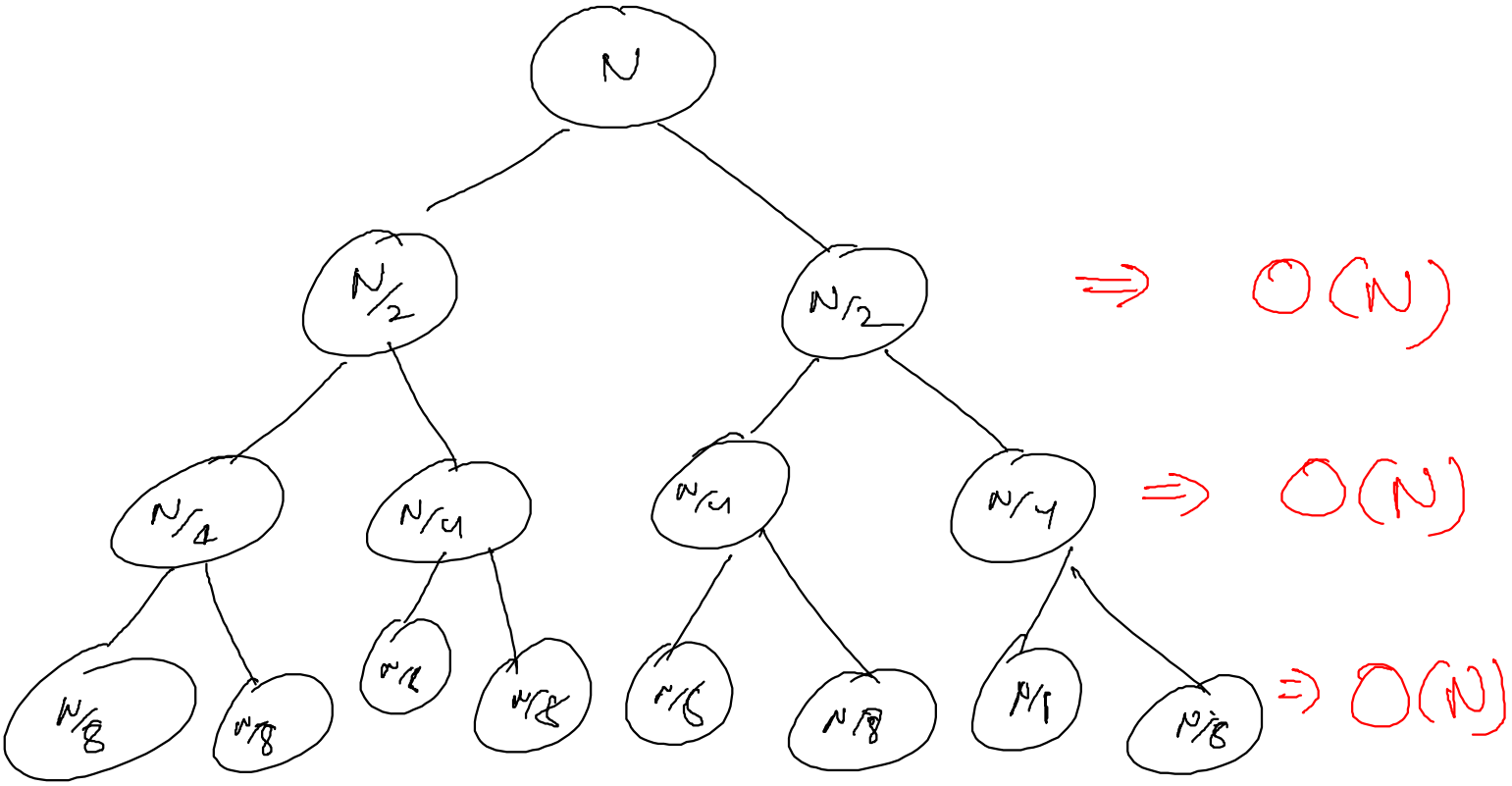
$(b-1)$

Time: O(b)

Space: O(log b).

⊙ Merge Sort:

O(N log N).



$$O(N \cdot \log N)$$

(1', 1'', 0', 0'')

Stable

Insertion →	Yes
Selection →	No
Bubble →	Yes
Quick →	No
Merge →	Yes
Heap →	No
Count →	Yes

⊙. Quick Sort :-

↳ Recursive

↳ Inplace

↳ stable "yes" ? "No"

(4, 1', 0', 1'', 0'')

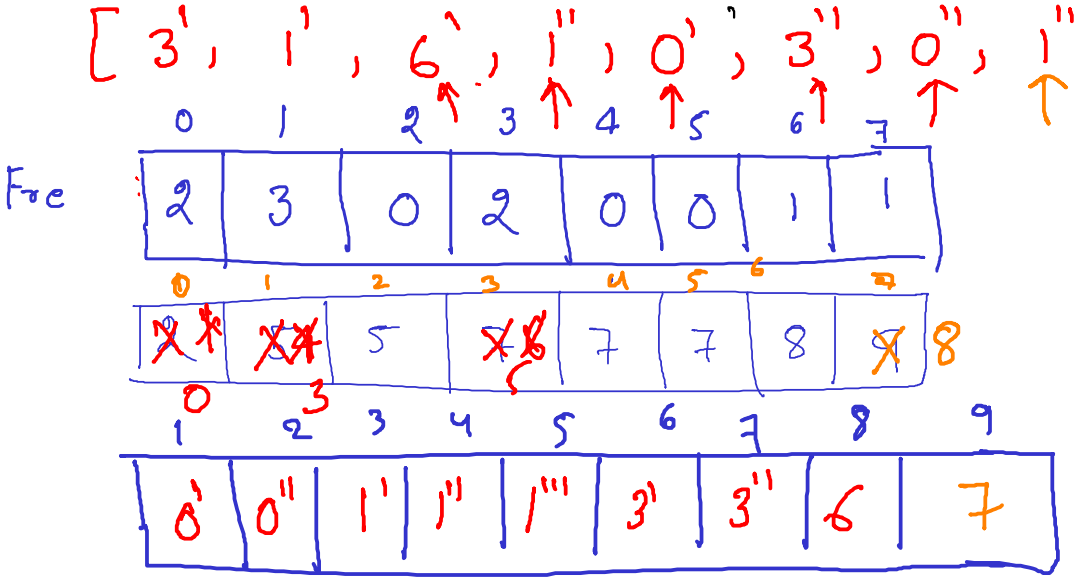
⇓

([0', 0''], [1', 1''], 4) ⇒ stable sort

(0'', 0', 1', 1'', 4)

(0'', 0', 1'', 1', 4)

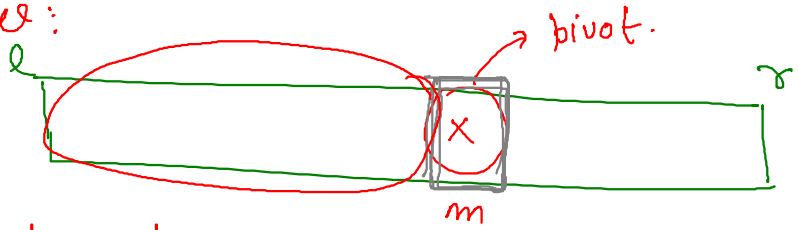
[3', 1', 6', 1'', 0', 3'', 0'', 1''', 7].



⇒ Quick Sort :.

↳ Recursive

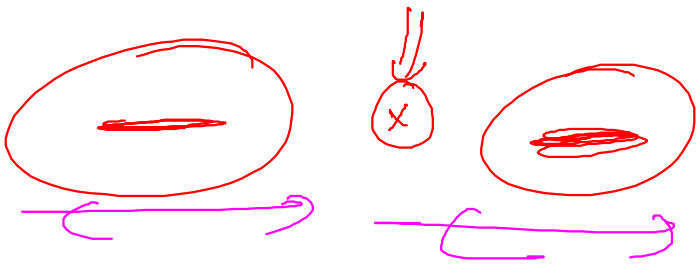
↳ Inplace :

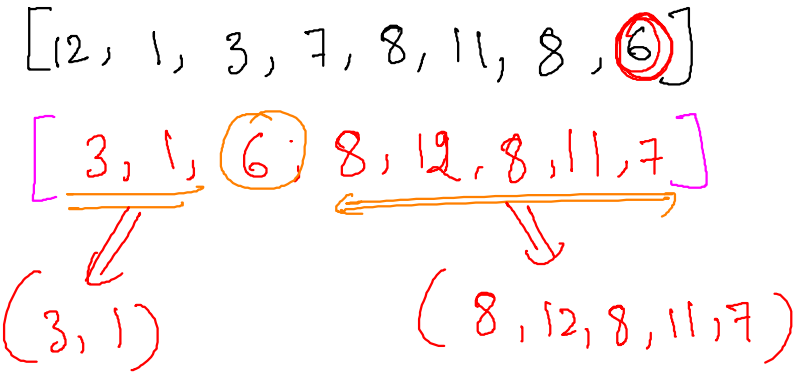


all number less than x are on the left of x & all Number which are greater than or equal to x on the right.

$$QS(l, m-1)$$
$$QS(m+1, r)$$

```
QS( l, r ) {
if( l>=r) return ;
int m = partition(a, l, r) ;
QS( l, m-1) ;
QS( m+1, r) ;
}
```





$$T(n) = T(n-k) + T(k) + O(N).$$

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

$O(N)$

$O(N-1)$

$O(N-2)$

$\approx O(N^2)$

Quick Sort Complexity: coorst $O(N^2)$

↳ pivot → maximum
→ Minm.

Randomise Quick Sort:-

Algorithm

Deterministic

Randomise.

[1, 4, 3, 2]

⇓

[1, 3, 2, 4]

I/p → [4, 3, 1, 2] → Algorithm → o/p [1, 2, 3, 4]

steps

[1, 4, 3, 2]

⇒ [1, 2, 4, 3]

Randomise

$T(n) = T(n/3) + T(\frac{2n}{3}) + O(N)$

average → $(N \log_{3/2} n)$

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

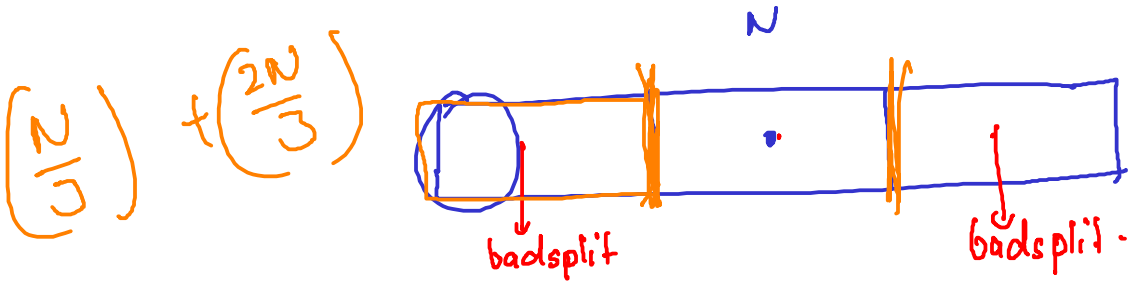
worst: → $O(N^2)$

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

best →

$(N \log N)$

Time: $= O(N \log N)$



$T(n) \leq \left(T(n-1) + T(1) + O(N)\right)\frac{2}{3} + \left(T(n/3) + T(\frac{2n}{3}) + O(N)\right)\frac{1}{3}$

$T(n) \leq \frac{2}{3}T(n-1) + \frac{2}{3}T(1) + O(N) + \frac{1}{3}T(n/3) + \frac{1}{3}T(\frac{2n}{3})$ &

$T(n) \leq \frac{2}{3}T(n) + \frac{1}{3}T(n/3) + \frac{1}{3}T(\frac{2n}{3}) + O(N)$

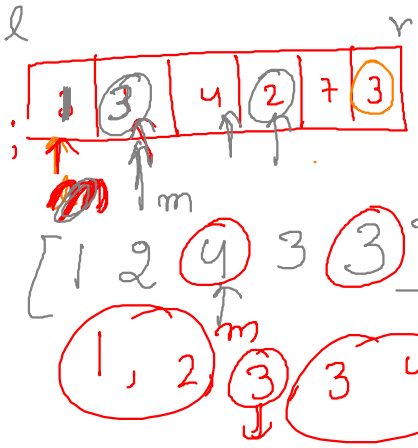
$T(n) - \frac{2}{3}T(n) \leq \frac{2}{3}T(n/3) + T(\frac{2n}{3}) + O(N)$

$\frac{T(n)}{3} \leq \frac{1}{3}T(n/3) + \frac{1}{3}T(\frac{2n}{3}) + O(N)$

$= T(n/3) + T(\frac{2n}{3}) + O(N).$

Quick Sort

```
partion (a, l, r) {
    int ridx = rand() % (r-l+1) + l;
    swap(arr(r), arr(ridx));
    int pivot = arr(r);
    int m = l;
    for (int i = l; i < r; i++)
    { if (arr(i) < pivot){
        swap(arr(i), arr(m));
        m++;
      }
    }
    swap(arr(m), arr(r));
}
return m;
```



rand()
↳ 0, rand. MAX.

rand() % N = (0 - N-1)

rand() % (r-l+1) + l
⇒ (0, r-l)
⇒ (l, r)

```
QS(a, l, r) {
  if(l >= r) return;
  int p = partion(a, l, r);
  QS(l, p-1);
  QS(p+1, r);
}
```

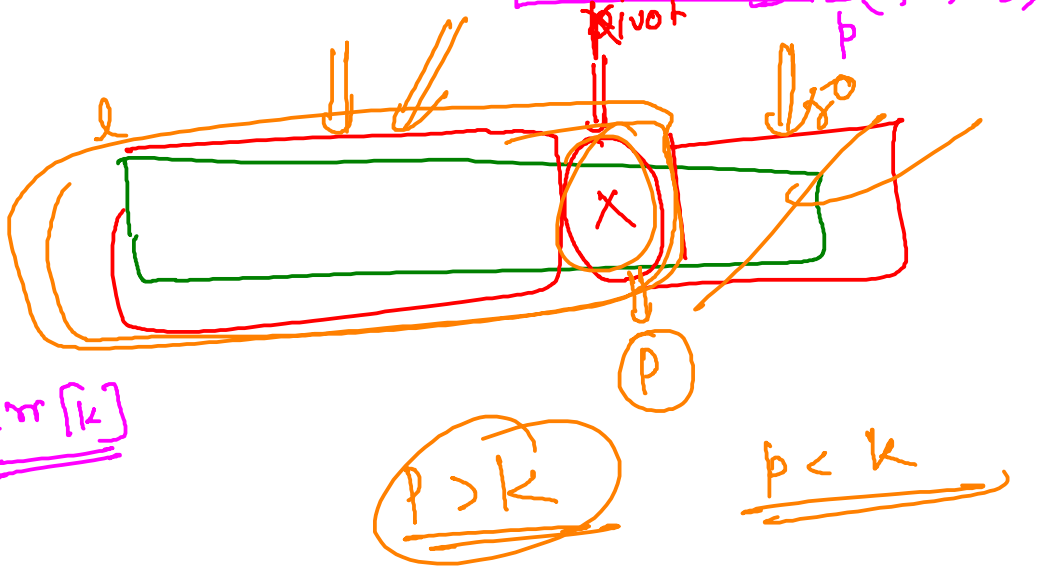
⊙ kth - smallest / largest O(N).

↳ sort → N log N

↳ priority_queue → N log K.

↳ quick-select . O(N)

```
if(p == k)
    return a[k];
if(p > k)
    QS(l, p-1)
else
    QS(p+1, r);
```



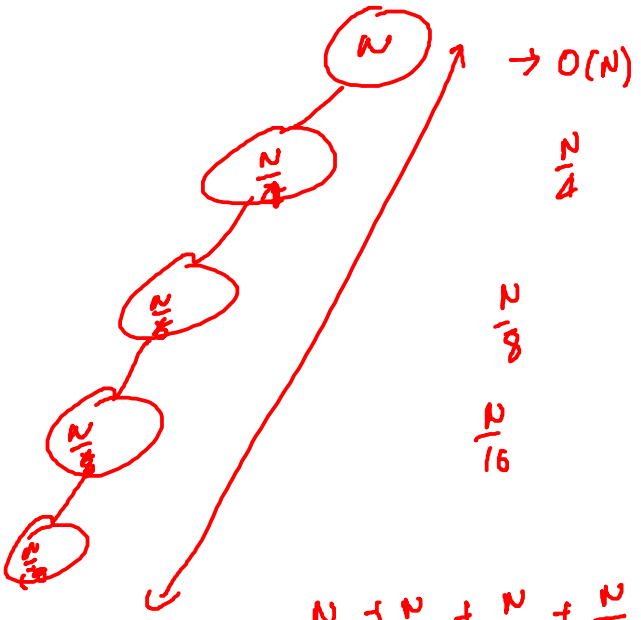
```cpp
// K : Find Kth smallest element and return using th

int Part(int arr[], int l,int r){
    int ridx = rand() % (r - l + 1) + l;
    swap(arr[ridx], arr[r]);
    int m = l;
    int p = arr[r];
    for(int i = l; i < r;  i++){
        if(arr[i] < p){
            swap(arr[i], arr[m++]);
        }
    }
    swap(arr[r], arr[m]);
    return m;
}
int QS(int l,int r, int arr[], int k){
    if(l > r)return -1;
    int p = Part(arr, l, r);
    if( p == k)return arr[k];
    if(p > k)
        return QS(l, p - 1, arr, k);
    else
        return QS(p + 1, r, arr, k);
}

int kthSmallest(int arr[], int l, int r, int k) {
    return QS(l,r,arr,k-1);
}
};
```

Time!

O(N)



$N + \frac{N}{4} + \frac{N}{8} + \frac{N}{16}$

$N\left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots\right)$

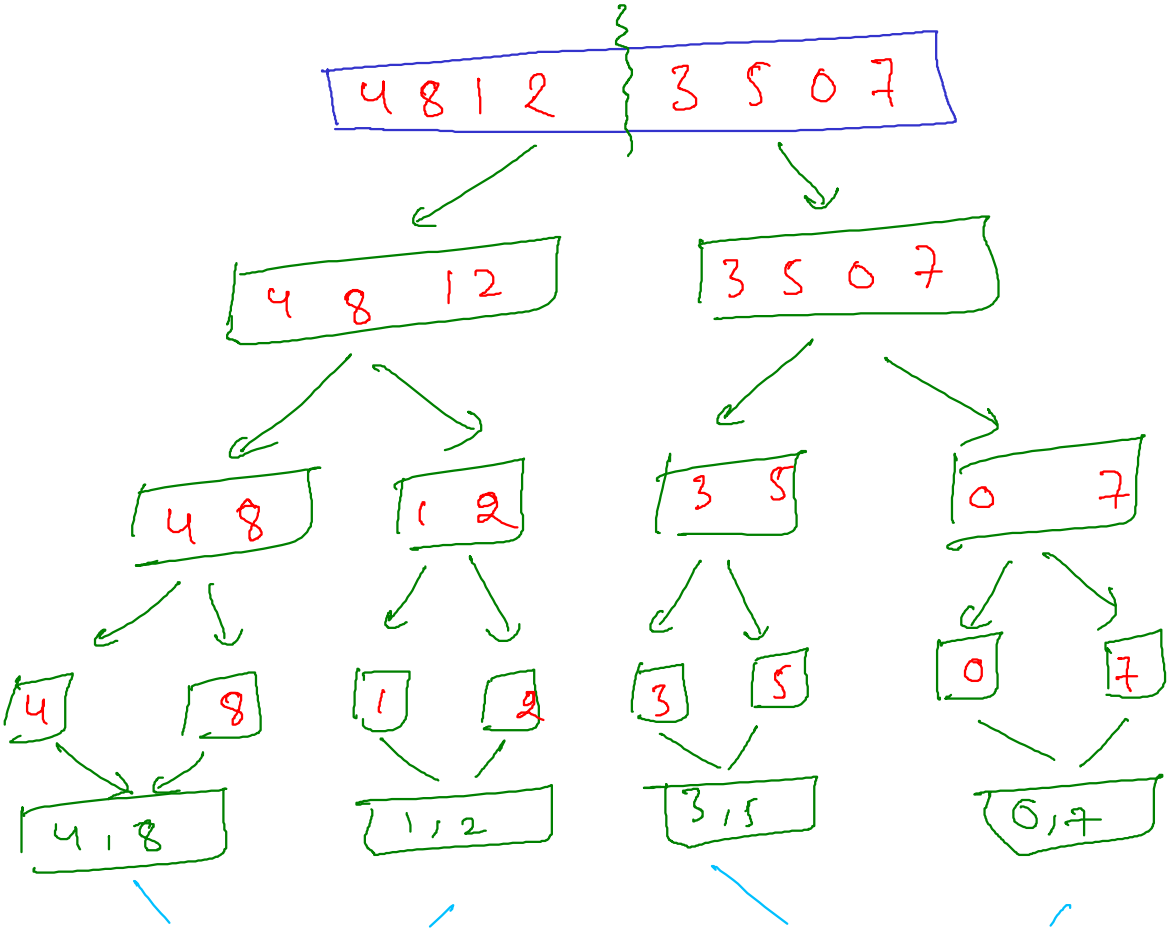
$N(2)$ $2N$ $O(N)$

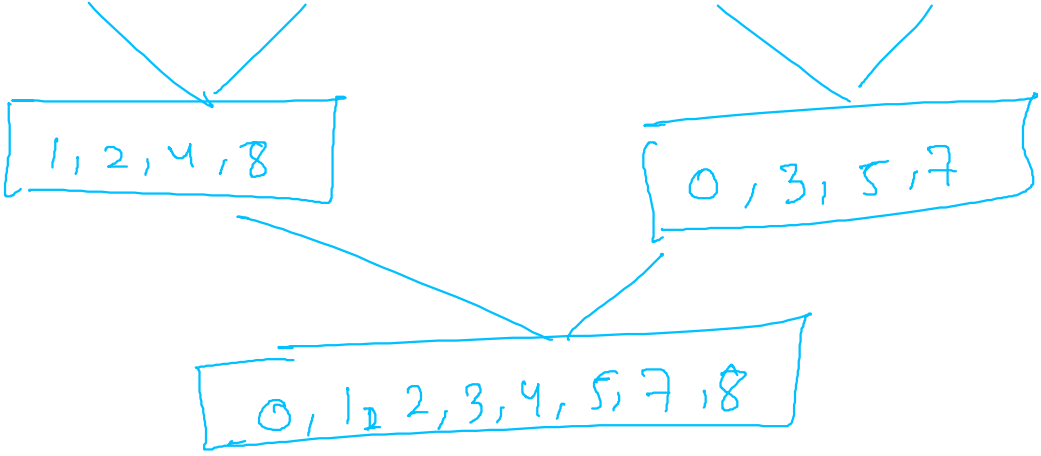
==x==x==x==x==

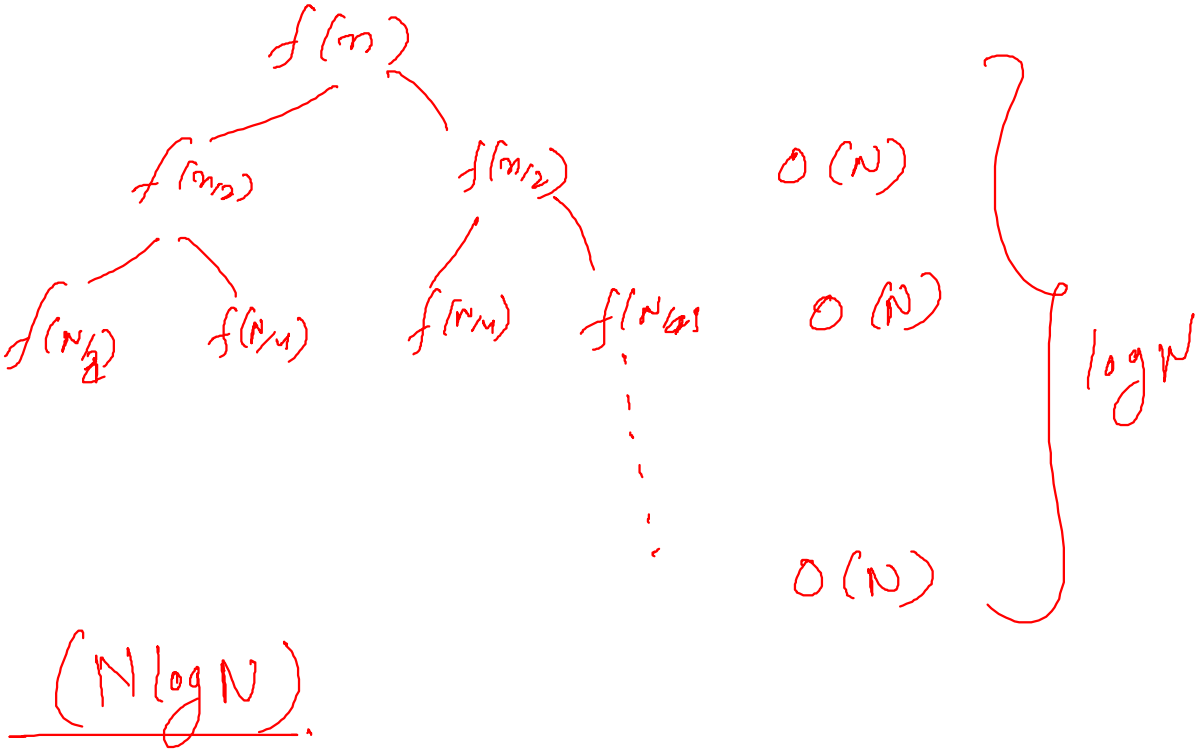
MERGE SORT :-

↳ Recursive

↳ Stable







$(N \log N)$.

Q. Check the given array is sorted in non-dec order or not. (recursively).

0 1 2 3 4

1, 2, 2, 3, 4 "yes"

1, 2, 2, 3, 2 "No".

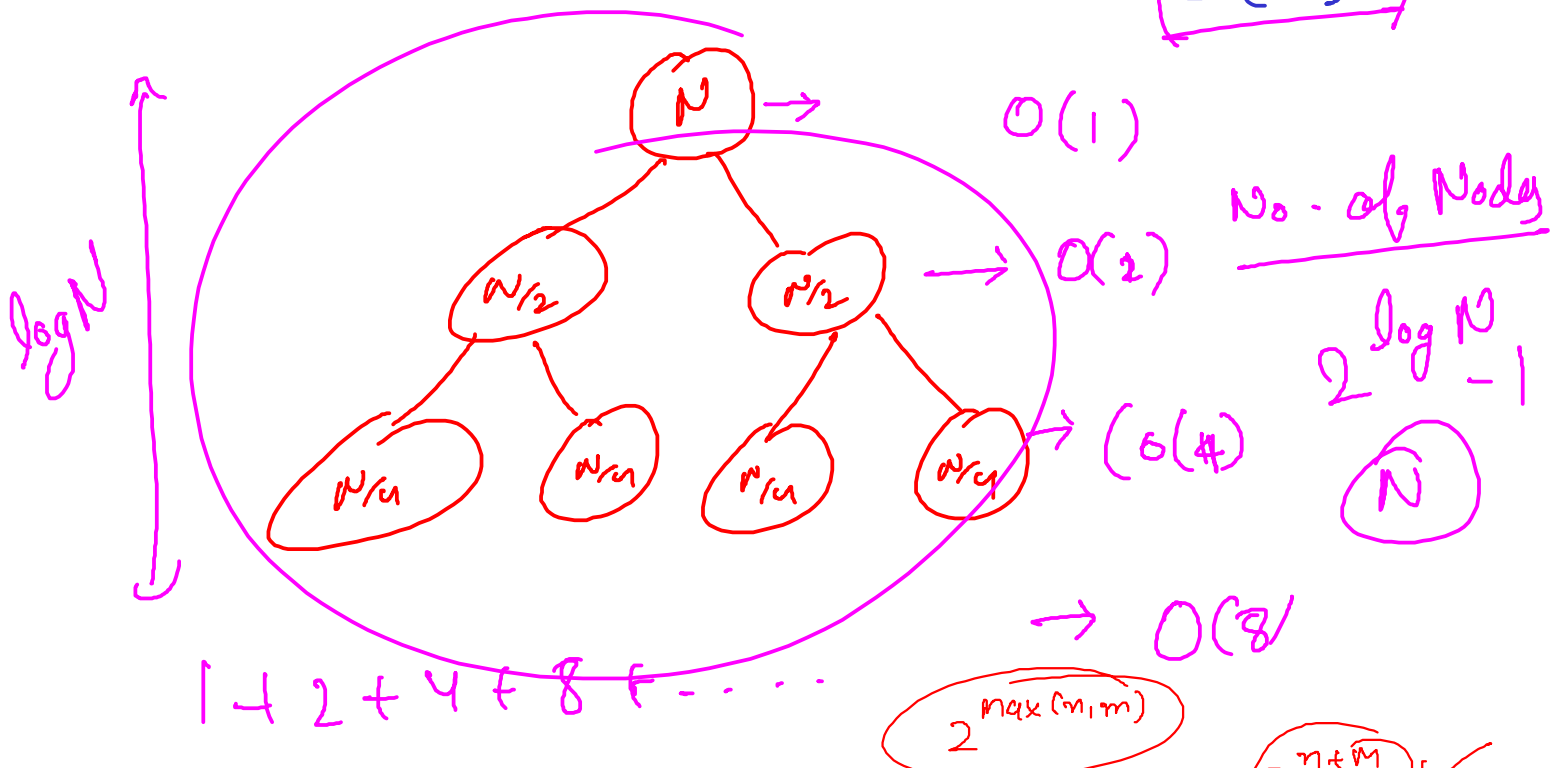
```
f(Arr, N) {
  if (N==1) return true;
int a=f(Arr, N-1)
  if (a && Arr[N-1] >= Arr[N-2])
      return true;
  return false;
```

O(N).

```
f(Arr, l, r) {
  if(l>=r) return true;        mid = (l+r)/2;
  int l = f(Arr, l, mid);
  int r = f(Arr, mid+1, r);
  if([[l & r && A[mid] <= A[mid+1]])
      return true;
  return false;
}
```

Time: ~~log N~~.

O(N). ✓



No. of Nodes

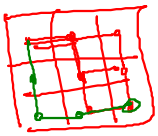
$2^{\log N} - 1$

N

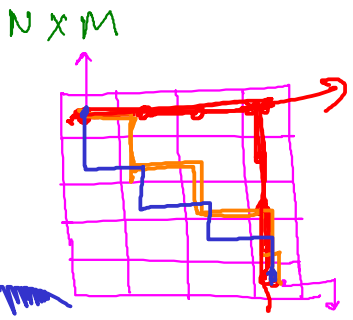
$2^{\max(n,m)}$

2^{n+m} ✓

Q. Grid Path.



N+m



(i, j) → (i, j+1)

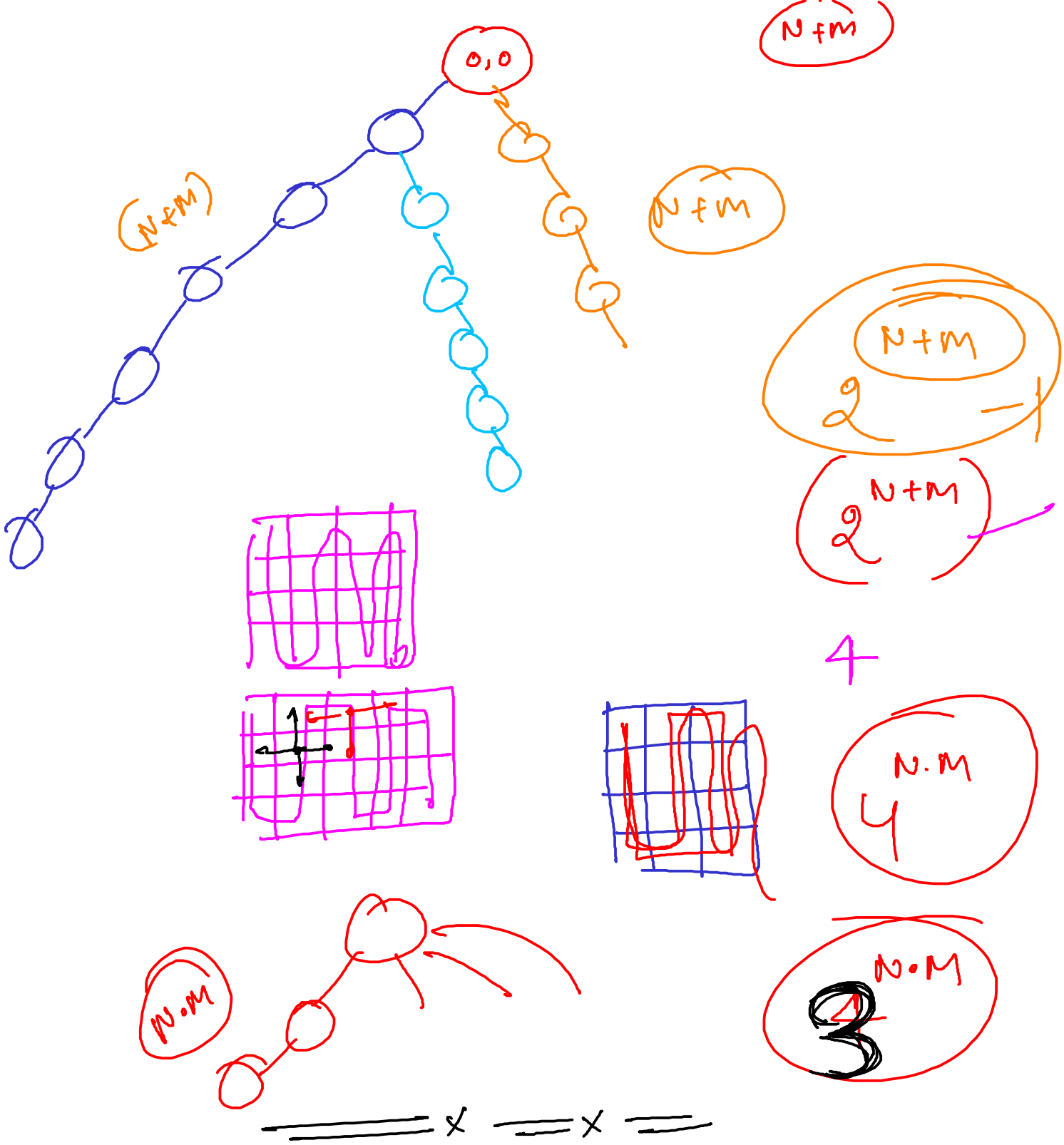
↓

(i+1, j).

RRDD
RDDR
RDRD
DDRR

$2^{(N \cdot M)}$

$2^{N \cdot M}$. (N x M)



- Subset sum = k (print) ✓
- Print all subset ✓
- Perm
- Print all whole Numbers (1 — N) in lexicographical order.

0 1 10 11 12 13 14 15 16

17 18 19 2 20 3 4 5 6

7 8 9.

20

1 <int, int>

highstvalue

1 → 10

2 → 20

3 → ~~30~~ 3.

20

```
f idx=0)                                   N=20)
    if(idx==N){
for(i=0;  i<=20;  i++){
    swap(arr(i), arr(idx));
    f(idx+1, arr);
}
    swap(arr(i), arr(idx))
```

V = {1 ——— N)

Sort (v.begin(), v.end(), [&](int a, int b){

~~if~~ return to_string(a) < to_string(b)}

};

return

```
class Solution {
public:
    vector<int> lexicalOrder(int n) {
        vector<int> a(n, 1);
        for(int i = 1 ; i < n ; i++)a[i] += a[i - 1];
        sort(a.begin(), a.end(), [&](int z, int y){
            return to_string(z) < to_string(y);
        });
        return a;
    }
};
```

f(N) {

if(N > limit) return;

cout << N;

for(int i=0; i<a; i++)

f(N*10+i);

}.

f(1) 0*10+0

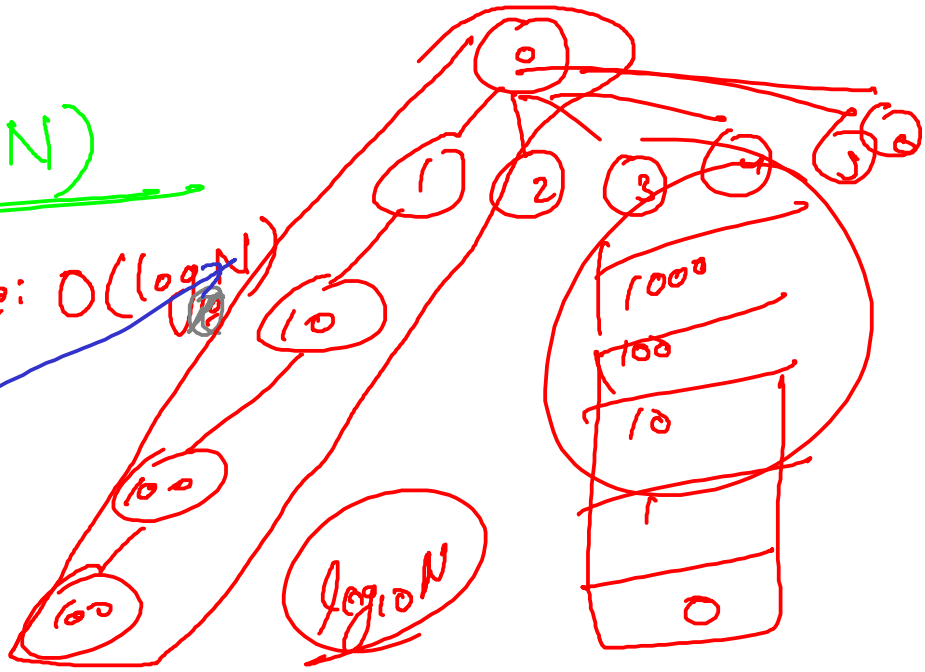
0*10+0

0

```cpp
class Solution {
public:
    vector<int> ans;
    void dfs(int n, int limit){
        if(n > limit)
            return;
        if(n)
            ans.push_back(n);
        for(int i = 0; i < 10 ; i++){
            if(n == 0 and i == 0)
                continue;
            dfs(n * 10 + i, limit);
        }
    }
    vector<int> lexicalOrder(int n) {
        dfs(0, n);
        return ans;
    }
};
```

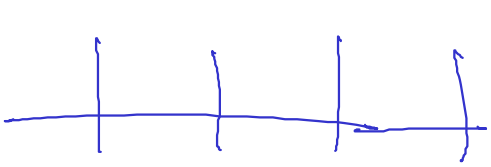
Time: O(N)

Space: O(log N)



L - R ( )

L - R (1)



9

L=1=4 R=N

9 2 X-4 9 9 x 9 x 2 x 9

int f(int idx, int f, int c, int cout,

int f(idx, f, int c, cout)

{ if(idx >= N) return cnt ; else { 0 - 9

7 8 3 4 5 ->

(0)

if (c){

for(int i=0; i<=s[idx]-'0'; i++)

ans += f(idx+1, s[idx]=='0'+i ; cout+

Meet - In - the - Middle :-

① Subset Sum problem.

$[a_0 \quad a_1 \quad a_2 \ldots a_{n-1}]$

No. of subset (sum = k);

$N = 20$, $A_i = 10^9$

↳ $2^{20} = 10^6$ ( )

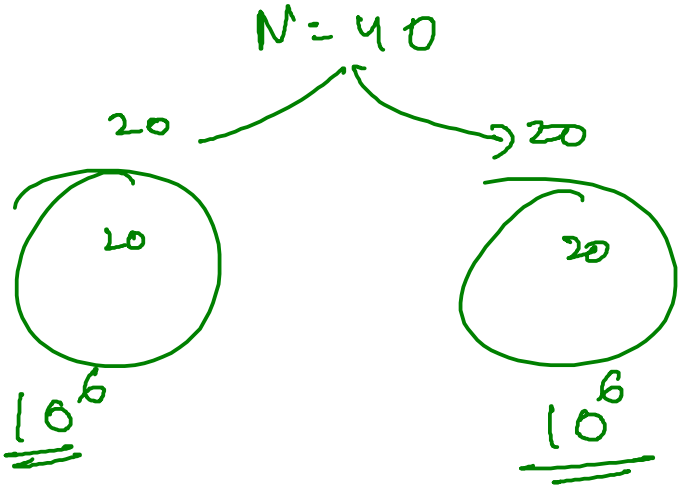
$N = 1000$, Sum = 10,000

N x Sum

DP

$N \le 40$

$1 \le A_i, Sum \le 10^9$



⇒ Map[S] ++;

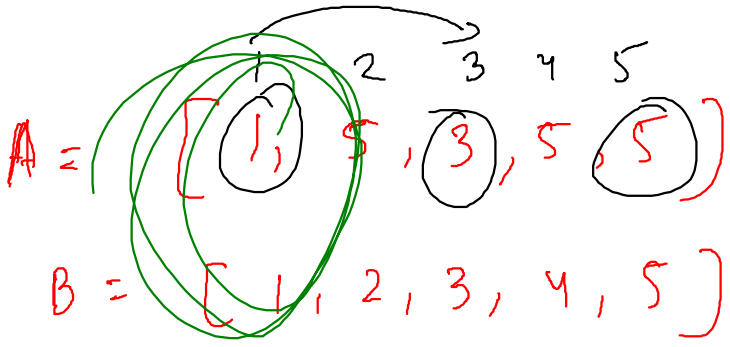
```
for(int i=0; i< subl.size(); i++){
    int sum = subl[i];
    int y = k - sum;
    ans += Map[y];
}
return ans;
```

y + Sum = k

Sum

$(n \times 2^{n/2})$ y = k - sum

$O(2^{n/2} + 2^{n/2} + (2^{n/2} \times \log$



A[i] != B[i].

∀ i:

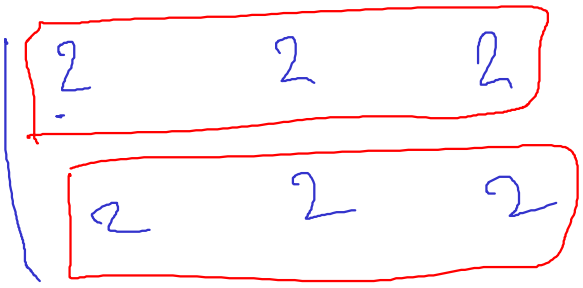
4 i, j

+6

10

A(i) , A(j)

(i+j)



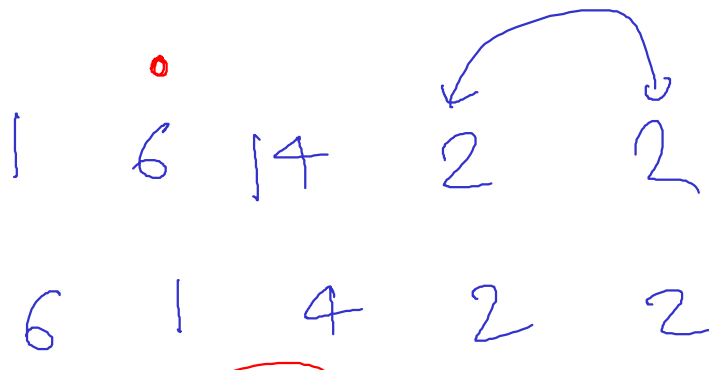
3 3

3 3

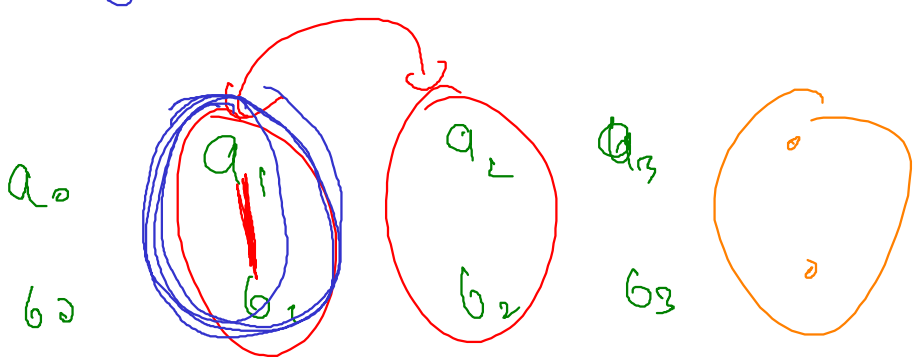
N/2

-1

i+j



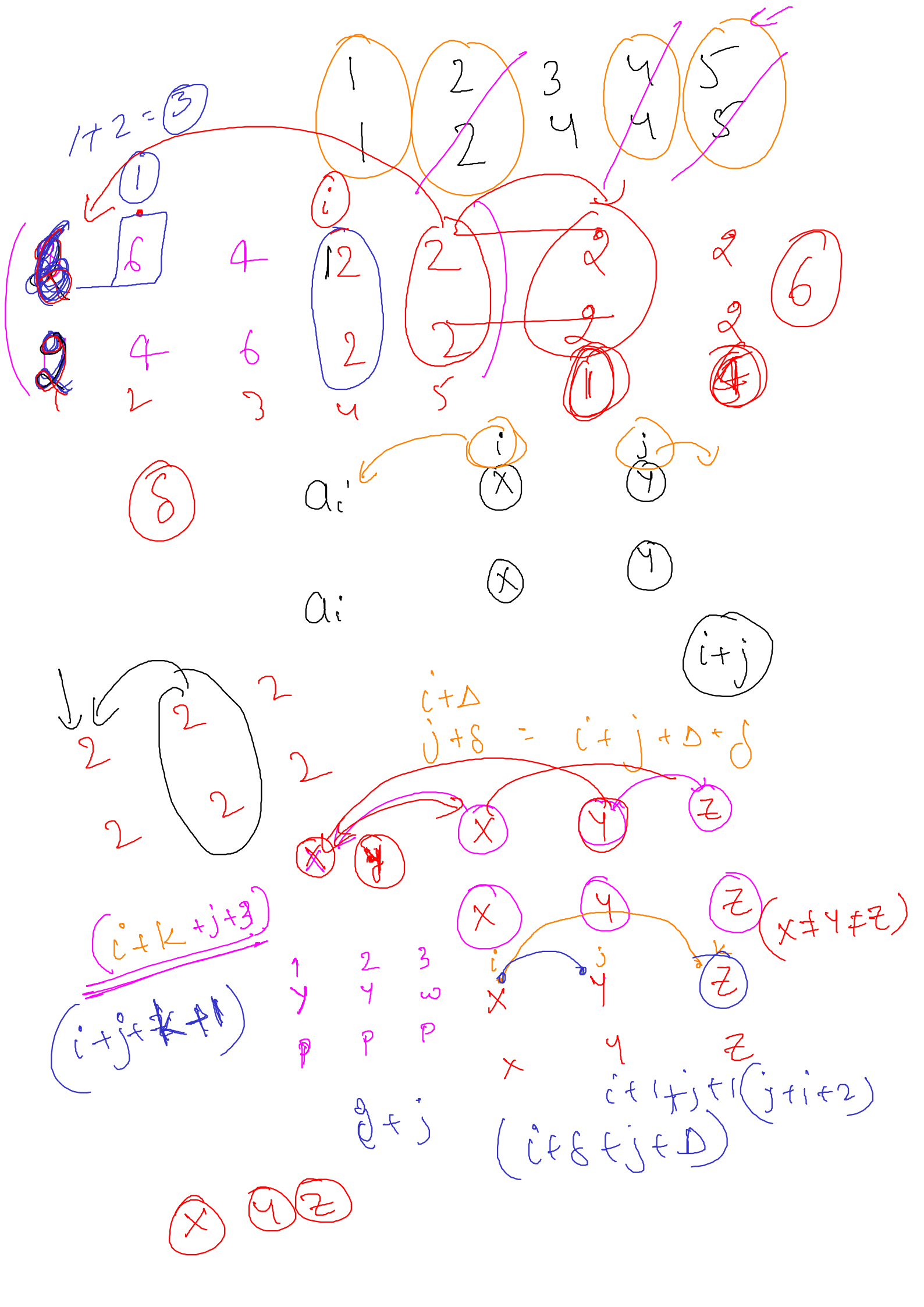
i+

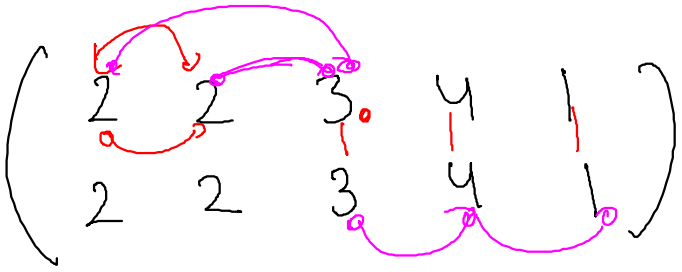


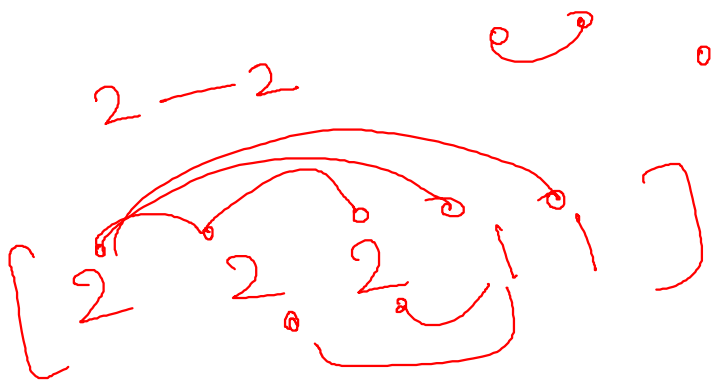
ai
bi

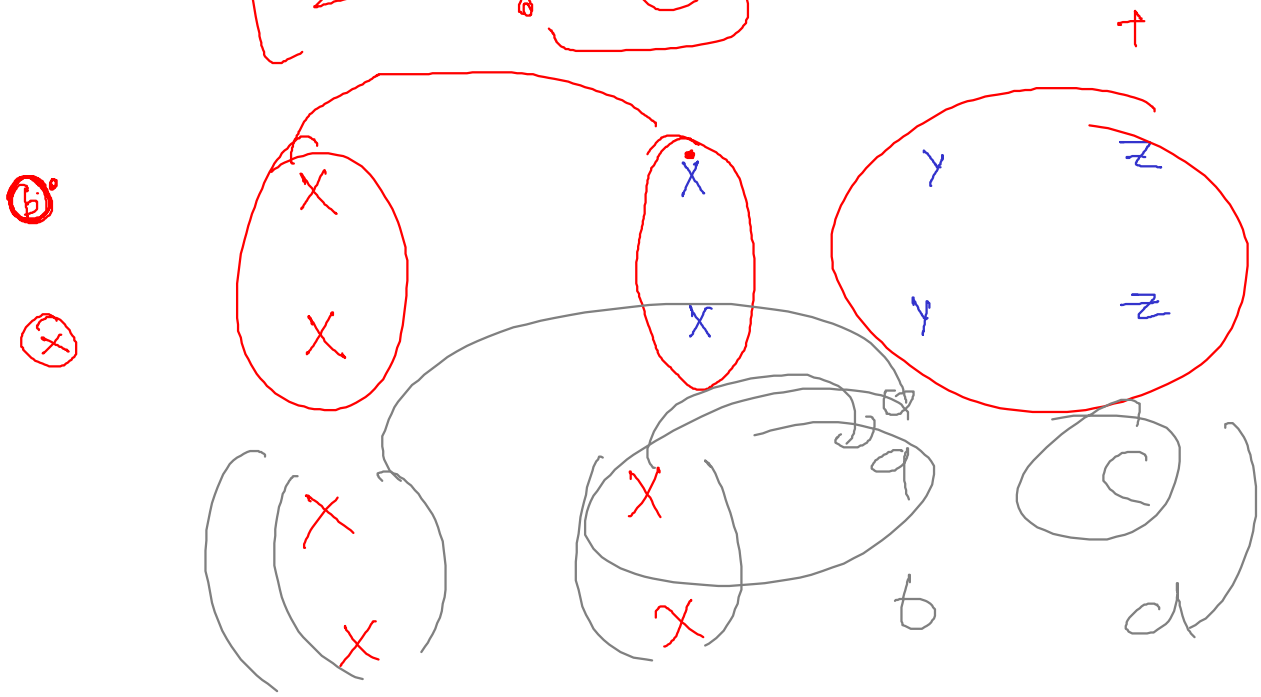
i+j
ai+5
bi+5

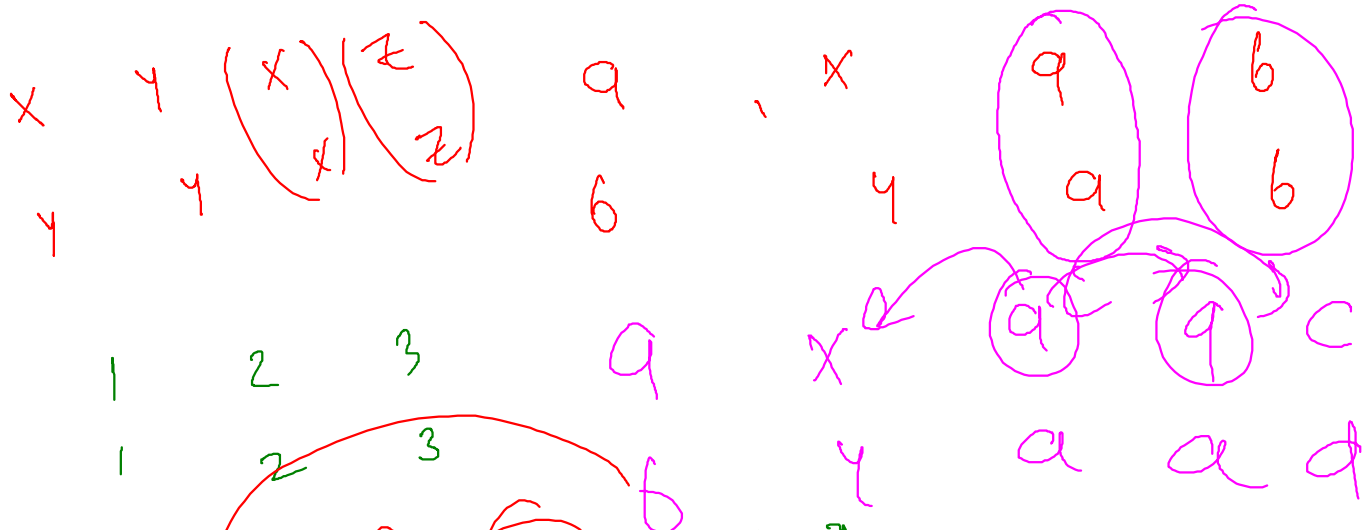
i+i+5
2i+5

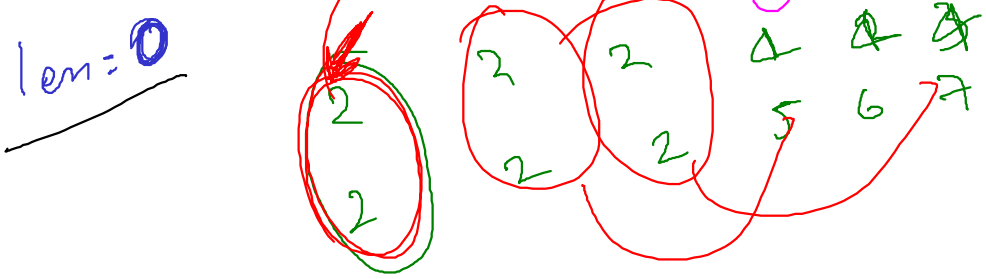

1+2=3
i
a_i'
a_i
i+j
i+Δ
j+δ = i+j+Δ+δ
(i+k+j+3)
(i+j+k+1)
(x≠y≠z)
(i+1+j+1)(j+i+2)
(i+δ+j+Δ)

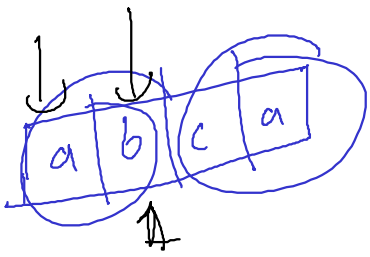


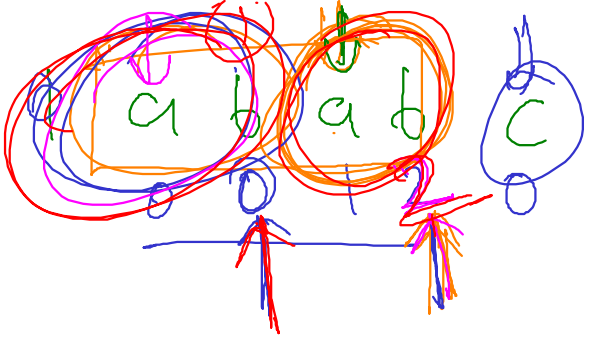


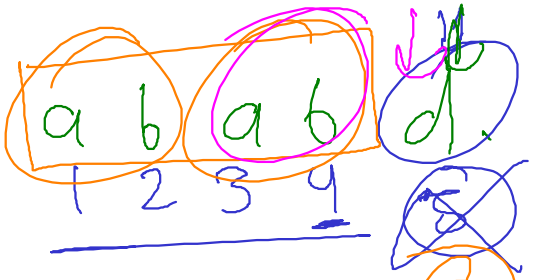








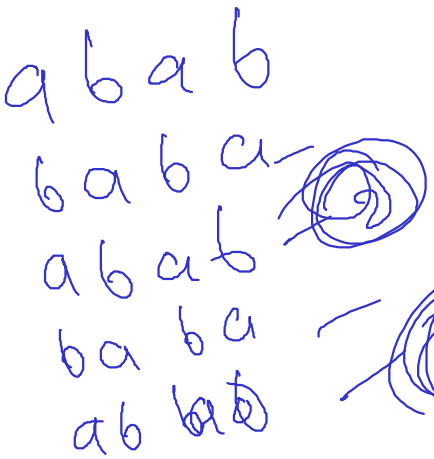

a b a b c
0 0 1 2 0

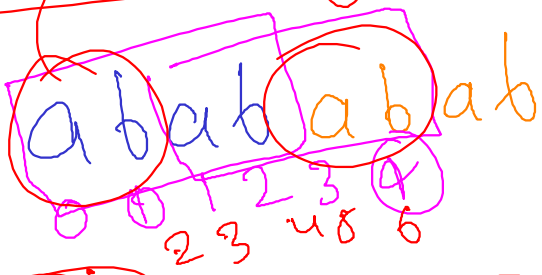

a b a b d
1 2 3 4


8
8

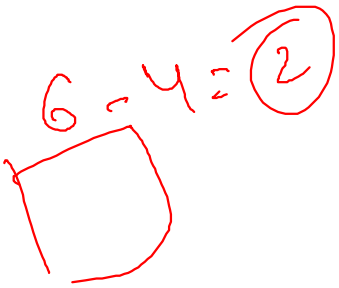

4

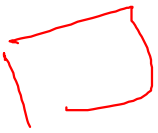
lps[len-1]


a b a b
b a b a
a b a b
b a b a
a b a b


a b a b a b a b
0 0 1 2 3 4
2 3 4 0 6

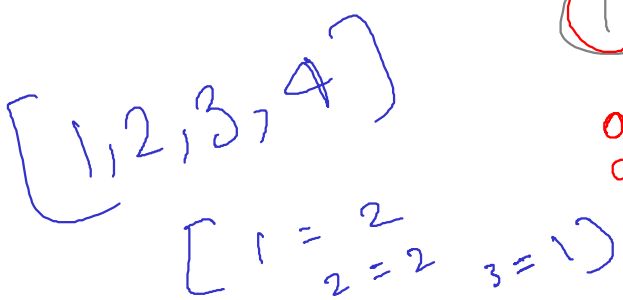

N=4

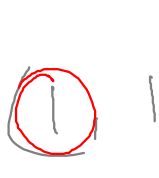

6-4=2

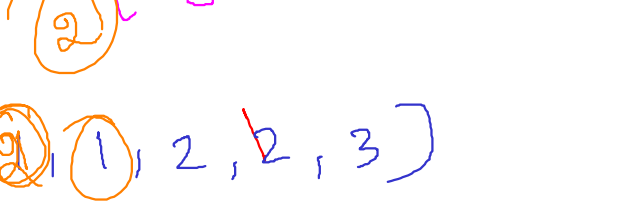



n=3
[1]=1
[2]=2
3,2,1


2, 2 (3,2,1)
[1, 1, 1]
5


1,2,3,4
[1=2
2=2 3=1]


1 1 1
2


[1]=3=[2]=2
[0 1, 2, 2, 3]