DnC - divide and conquor

Merge two sorted arrays

IP : Arr1 = | 2 | 4 | 6 |
i

Arr 2 = | 3 | 5 | 7 | 9 | 1 |
j

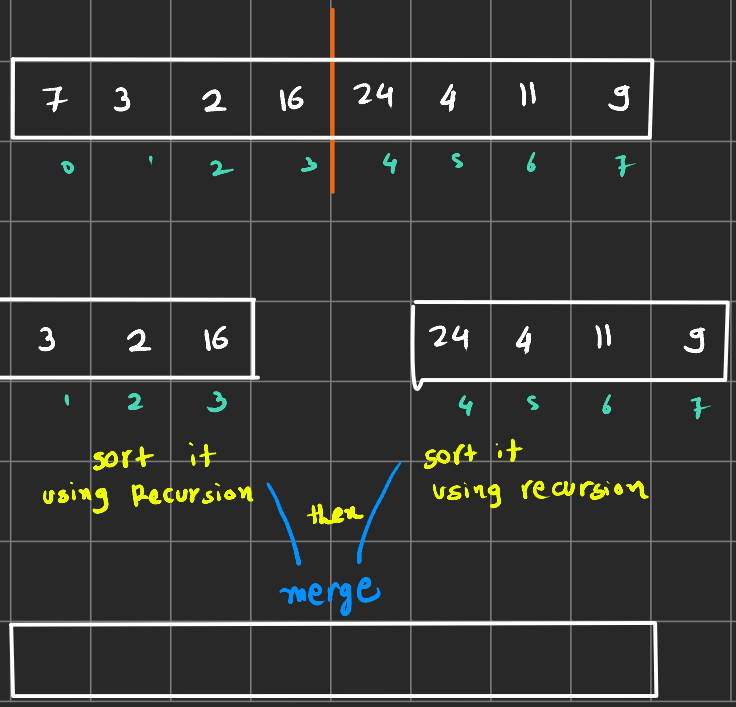
two pointer approach if (i<j) then use it and i++

IF (i is out of bond or j is outof bond)

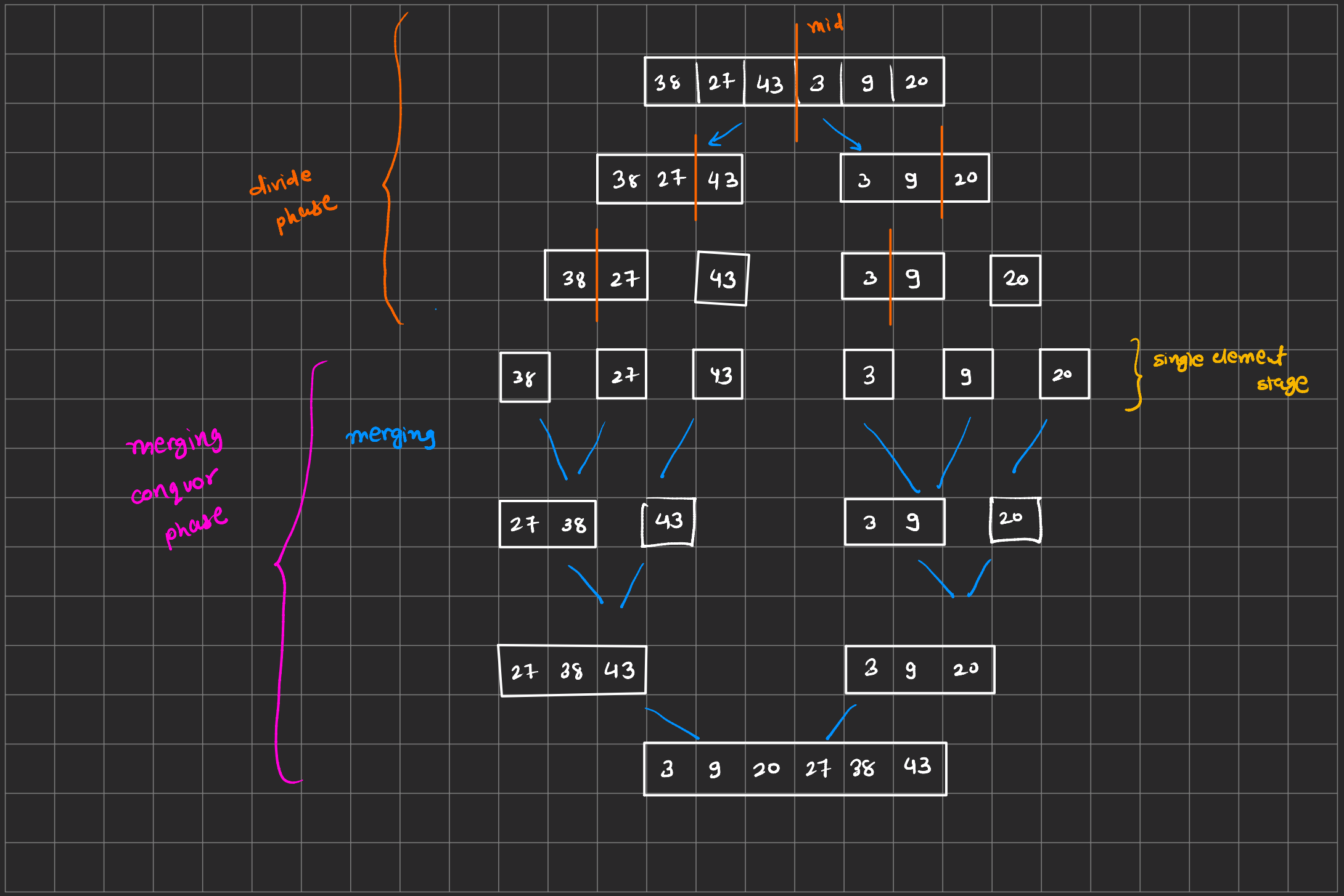
then arrA remaining elements add them

arrB remaining ele add it

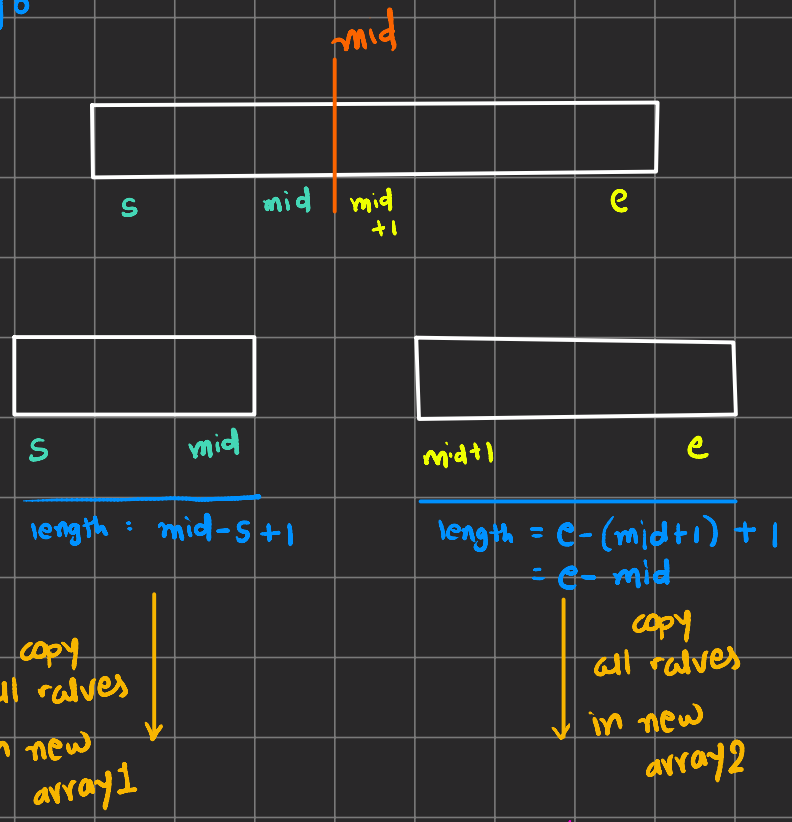
Merge sort




```
void mergeSort(int *arr, int s, int e){

    // base case
    if(s==e){
        // single element
        return;
    }

    // rr
    int mid = (s+e)/2;
    // left part sort
    mergeSort(arr ,s, mid);

    // right part sort
    mergeSort(arr, mid+1, e);

    // merge two sorted arr
    merge(arr,s,e);

}
```



Merging Algo




```
void merge(int *arr, int s, int e){
    int mid = (s+e)/2;

    int len1 = mid-s+1;
    int len2 = e-mid;

    int* leftArr = new int[len1];
    int* rightArr = new int[len2];

    // copy values
    int k = s;
    for(int i=0;i<len1;i++){
        leftArr[i] = arr[k];
        k++;
    }

    k = mid+1;
    for(int i=0;i<len2;i++){
        rightArr[i] = arr[k];
        k++;
    }
```

$$TC = \underbrace{k_1}_{\text{Base case}} + \underbrace{T(n/2)}_{\text{left sort}} + \underbrace{T(n/2)}_{\text{right sort}} + \underbrace{(n * k)}_{\text{merging}}$$

$$= \underbrace{k_1}_{\text{neglect it}} + 2.T(n/2) + n * k$$

$$TC = 2.T\left(\frac{n}{2}\right) + n * k$$

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

$$2\,T\left(\frac{n}{2}\right) = 4\,T\left(\frac{n}{4}\right) + 2\frac{n.k}{2} \quad \times 2$$

$$4.T\left(\frac{n}{4}\right) = 8\,T\left(\frac{n}{8}\right) + 4\frac{n.k}{4} \quad \times 4$$

$$\vdots \qquad \vdots$$

$$T(1) = k$$

} a

$$T(n) = (a-1).n.k + k \qquad \therefore a = \log n$$

according to Binary Search

$$= (\log n - 1).n.k + \underbrace{k}_{\text{neglect}}$$

$$T(n) = \underbrace{n.k.\log n}_{\text{neglect}} - \underbrace{nk}_{\text{neglect}} = n.\log n = O(n.\log n)$$

```
// merge two sorted arrays
int leftIndex = 0;
int rightIndex = 0;
int mainArrIndex = s;

while(leftIndex<len1 && rightIndex<len2){
    if(leftArr[leftIndex] < rightArr[rightIndex]){
        arr[mainArrIndex] = leftArr[leftIndex];
        mainArrIndex++;
        leftIndex++;
    }
    else{
        arr[mainArrIndex] = rightArr[rightIndex];
        mainArrIndex++;
        rightIndex++;
    }
}

// copy logic for left arr
while(leftIndex<len1){
    arr[mainArrIndex] = leftArr[leftIndex];
    mainArrIndex++;
    leftIndex++;
}

while(rightIndex<len2){
    arr[mainArrIndex] = rightArr[rightIndex];
    mainArrIndex++;
    rightIndex++;
}

// Clean up memory
delete[] leftArr;
delete[] rightArr;
}
```

Quick Sort

Pivot Index

8	1	3	4	20	50	30
0	1	2	3	4	5	6

s = 0, e = 6

→ Partition logic (1 element ko uski sahi Jagah rakh do)

- element choose (Pivot ele)
- put it at right location → pivotIndex = s
- Pivot
  - chote eles → Pivot se left side set karo
  - bade eles → Pivot se right side set karo

for (s → e)

1 < 8, 3 < 8, 4 < 8

20 < 8 False

countno < pivot element

count = 0 x, x, 3

8 ko 4th element hona chiye

swap (arr [pivotIndex], arr [s+count])

s is not going to be 0 all time

4	1	3	8	20	50	30
0	1	2	3	4	5	6

→ Baki recursion sambhal lega

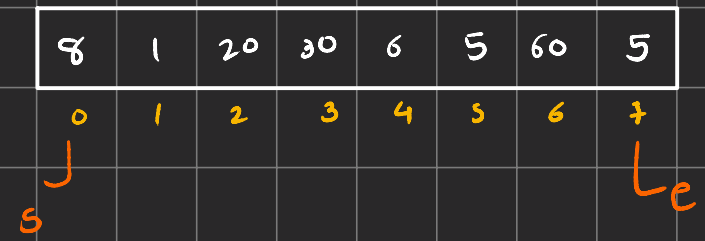
```
void quickSort(int *arr, int n, int s, int e){

    // basecase
    if(s>=e){
        return;
    }

    // partition logic
    int p = partition(arr, s, e);

    // rr
    quickSort(arr,n,s,p-1);   // left
    quickSort(arr,n,p+1,e);  // right

}
```

i j

4	1	3	8	20	50	30

left — arrange all small elements less than pivot element using two pointer approach

right — arrange all big elements which are bigger than Pivot ele using tp method



① pivotIndex = s = 0

② count = 0 for (s → e)

if (currentEle < pivotEle)

count++

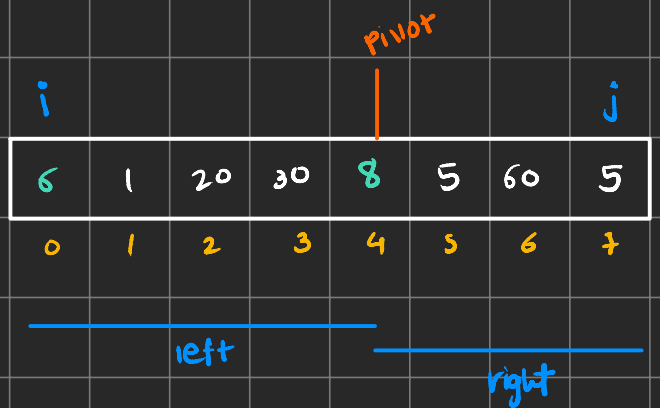
1<8 T 30<8 F 5<8 T 5<8 T

20<8 F 6<8 T 60<8 F

count = 4

so rightIndex = s + count = 0 + 4 = 4

swap (pivotEle, rightIndexEle)



③ if (element > pivot) → in left part { means wrong element

then swap (arr[i], arr[j])

j--

if (element < pivot) → in right part

then swap (arr[i], arr[j])

i++

Break when i == j

```cpp
int partition(int *arr, int s, int e) {
    // 1) choose pivot ele
    int pivotIndex = s;
    int pivotEle = arr[s];

    // 2) find right position for pivot
    int count = 0;
    for (int i = s + 1; i <= e; i++) {
        if (arr[i] <= pivotEle) {
            count++;
        }
    }
    // jab loop se bahar aya to pivot ele ka right position index
    // ready hai
    int rightIndex = s + count;
    swap(arr[pivotIndex], arr[rightIndex]);
    pivotIndex = rightIndex;

    // 3) small in left | big in right
    int i = s;
    int j = e;

    while (i < pivotIndex && j > pivotEle) {
        while (arr[i] <= pivotEle) {
            i++;
        }
        while (arr[i] > pivotEle) {
            j--;
        }

        // two pointer method and swapping to adjust eles
        if (i < pivotIndex && j > pivotEle) {
            swap(arr[i], arr[j]);
        }
    }

    return pivotIndex;
```

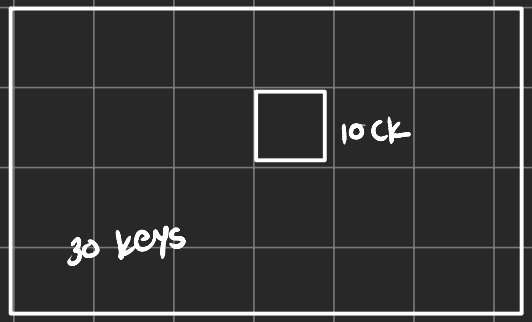
$TC = O(n \log n)$

Recursive tree try to make as merge short type

```cpp
vector<int> smaller, greater;

for (int i = s + 1; i <= e; i++) {
    if (arr[i] <= pivotEle) {
        smaller.push_back(arr[i]);
    } else {
        greater.push_back(arr[i]);
    }
}

// Rebuild the array with smaller, pivot, and greater elements
int index = s;
for (int num : smaller) {
    arr[index++] = num;
}

pivotIndex = index; // Correct position of pivot
arr[index++] = pivotEle;

for (int num : greater) {
    arr[index++] = num;
}


return pivotIndex;
```

```cpp
while (i < pivotIndex && j > pivotIndex)
{
    if (arr[i] < pivotEle)
    {
        i++;
    }
    else if (arr[j] > pivotEle)
    {
        j--; // Move to the next element on the right
    }
    else
    {
        swap(arr[i], arr[j]);
        i++; // After fixing `i`, move right
        j--; // After fixing `j`, move left
    }
}
```

Back Tracking

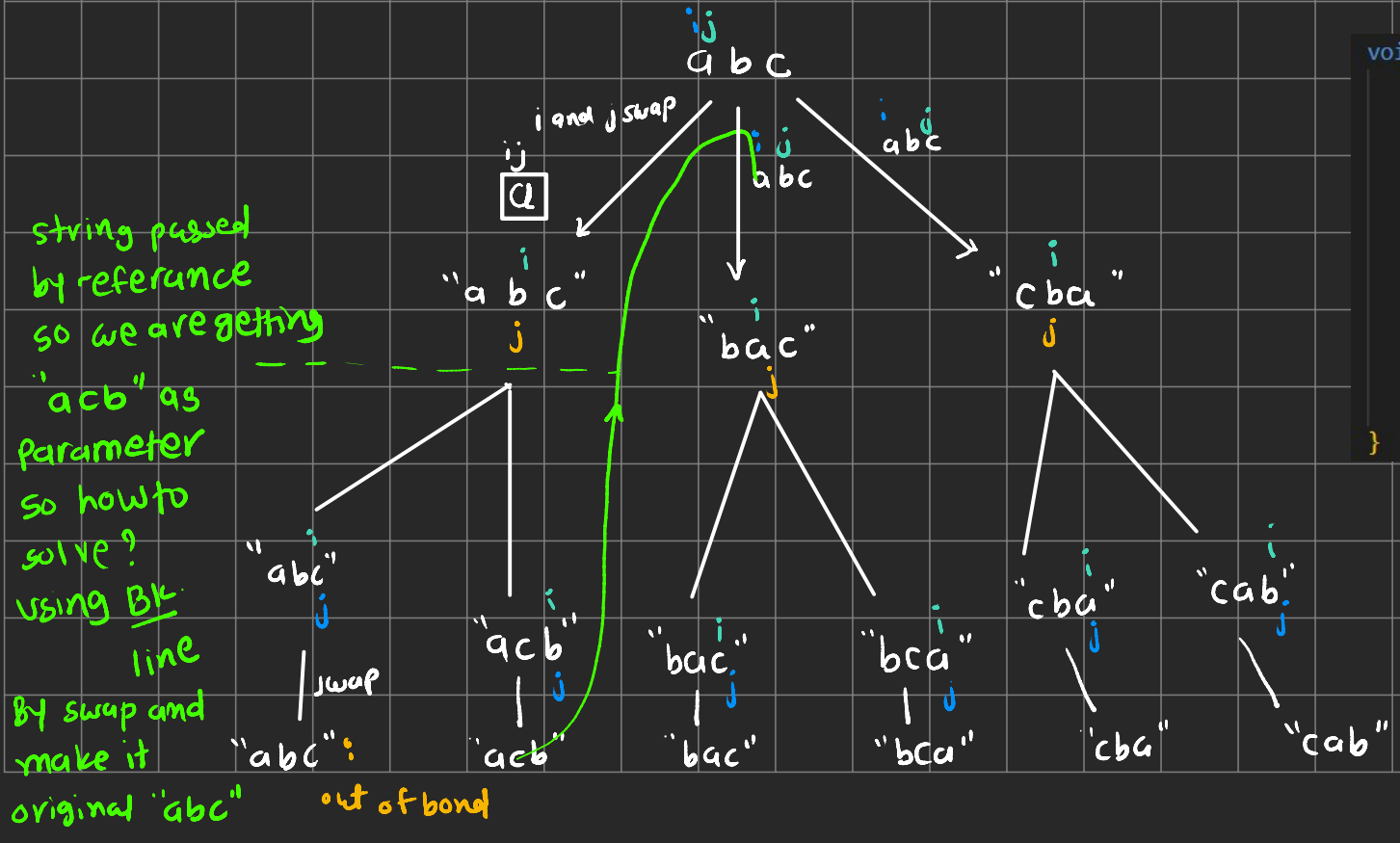


You have to open lock and you have to try all keys one by one to know about which key is answer

string permutations

Que- "abc"

abc acb
bac bca
cab cbc



string passed by referance so we are getting "acb" as parameter so how to solve? using Bk line By swap and make it original "abc"

```
void printPermutation(string &str, int i){

    if(i>= str.length()){
        cout << str << " ";
        return;
    }

    for(int j=i; j<str.length(); j++){
        swap(str[i], str[j]);   // swap
        printPermutation(str, i+1);     // rr
        swap(str[i], str[j]);   // BACKTRACKING
    }
}
```

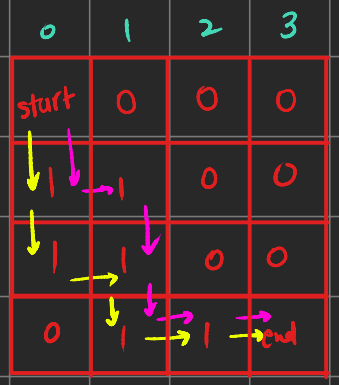
→ without this line of

abc acb cab cba abc acb

after Bk.

abc acb bac bca cba cab

☆ Rat in a Maze

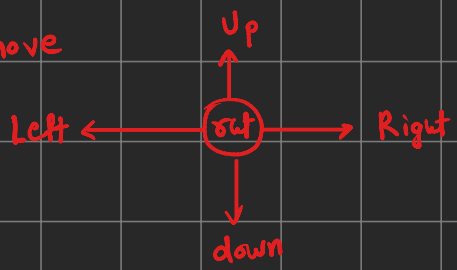


0 → Rasta blocked hai

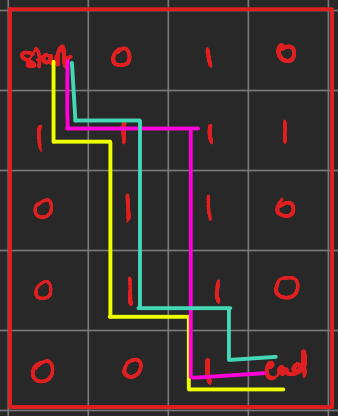
1 → Rasta open hai

- final all solution to reach destination - donot find kitkar ways

- rat can move



like



3 ways only

Sol. DDRDRR
DRDDRR

(0,0) → ~~D~~ | L | R | U

↓

(1,0) → ~~D~~ | ~~L~~ | ~~R~~ | U

↓

(1,1) → ~~D~~ | L | R | U

↓

(2,1) → ~~D~~ | L | R | U

↓

(3,1) → ~~D~~ | ~~L~~ | ~~R~~ | U

↓

(3,2) → ~~D~~ | L | R | U

then (3,1) } Here has jata infinite loop me (3,2) ⇔ (3,1) like this } if visited then marking it in visited 2D array

maze	0	1	2	3
0	start	0	0	0
1	1	1	1	0
2	0	1	1	0
3	0	1	1	end

	0	1	2	3 visited
0	1	0	0	0
1	1	1	0	0
2	0	1	0	0
3	0	1	1	1

(0,0) → D | L | R | U

↓ visited mark [1][0] = true

(1,0) → D | L | R | U

↓ visited [1][1] = true

(1,1) → D | L | R | U

↓ visited [2][1] = true

(2,1) → D | L | R | U

↓ visited [3][1] = true

(3,1) → D | L | R | U

visited [3][2] = true →

(3,2) D | L | R | U

visited already

visited [3][3] →

(3,3) D | L | R | U

Base condition destination pahuch gaye

check down is possible?
- index inside array
- that index value should be 1
- not visited hona chiye

=

mark unvisited

mark unvisited

mark unvisited

mark unvisited to call that we did before

=

while coming BACK

if you are on (i,j) then

D = (i+1, j)

L = (i, j-1)

R = (i, j+1)

U = (i-1, j)

```cpp
int main() {
    // Initialize the maze
    int maze[3][3] = {
        {1, 0, 0},
        {1, 1, 0},
        {1, 1, 1}
    };

    int row = 3;
    int col = 3;

    // Create a visited array and mark the source as visited
    vector<vector<bool>> visited(row, vector<bool>(col, false));
    visited[0][0] = true; // Mark the source value as true

    // Prepare to store paths
    vector<string> path;
    string output = "";

    // Solve the maze
    solveMaze(maze, row, col, 0, 0, visited, path, output);

    // Print all possible paths
    for (auto i : path) {
        cout << i << " ";
    }

    return 0;
}
```

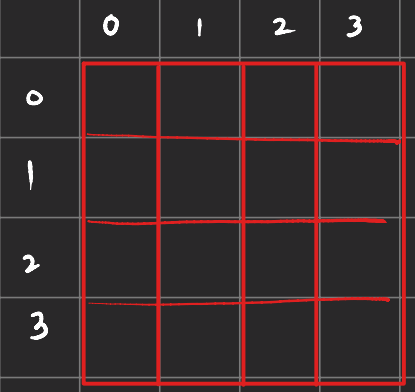
```cpp
void solveMaze(int arr[3][3], int row, int col, int i, int j, vector<vector<bool>> &visited, vector<string> &path, string output) {
    // base case
    if (i == row - 1 && j == col - 1) {
        path.push_back(output);
        return;
    }

    // down -> i+1, j
    if (isSafe(i + 1, j, row, col, arr, visited)) {
        visited[i + 1][j] = true;
        solveMaze(arr, row, col, i + 1, j, visited, path, output + 'D');
        visited[i + 1][j] = false; // BACKTRACK
    }

    // left -> i, j-1
    if (isSafe(i, j - 1, row, col, arr, visited)) {
        visited[i][j - 1] = true;
        solveMaze(arr, row, col, i, j - 1, visited, path, output + 'L');
        visited[i][j - 1] = false; // BACKTRACK
    }

    // right -> i, j+1
    if (isSafe(i, j + 1, row, col, arr, visited)) {
        visited[i][j + 1] = true;
        solveMaze(arr, row, col, i, j + 1, visited, path, output + 'R');
        visited[i][j + 1] = false; // BACKTRACK
    }

    // up -> i-1, j
    if (isSafe(i - 1, j, row, col, arr, visited)) {
        visited[i - 1][j] = true;
        solveMaze(arr, row, col, i - 1, j, visited, path, output + 'R');
        visited[i - 1][j] = false; // BACKTRACK
    }
}
```

```cpp
bool isSafe(int i, int j, int row, int col, int arr[][3], vector<vector<bool>> &visited) {
    if ((i >= 0 && i < row) && (j >= 0 && j < col) && (arr[i][j] == 1) && (visited[i][j] == false)) {
        return true;
    }
    return false;
}
```

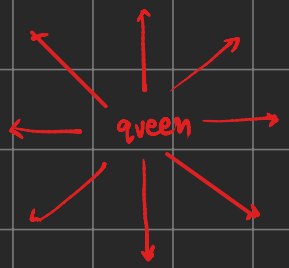
TC ⇒ exponential (4 calls DLRU)

N queen

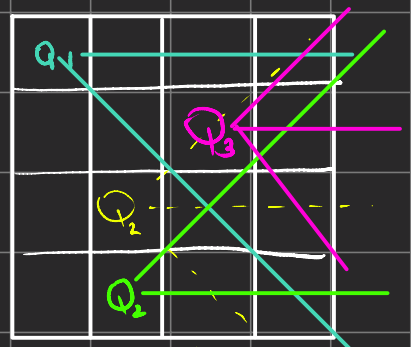
set N-queen in such a way so that no queen can attack the other queen



n×n
so place n queens

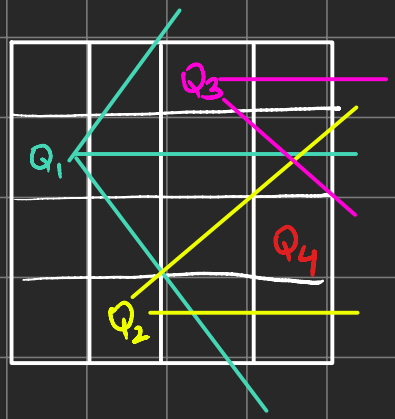


queen can walk in
8 direction

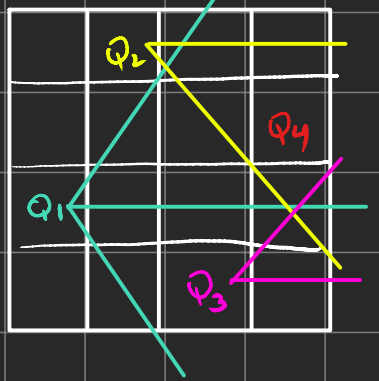


if I place Q_2 there then for Q_3 all ways are blocked so go down for Q_2

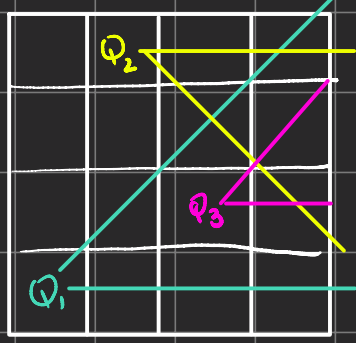
for Q_4 there is no place so sol. not possible



all placed



all placed

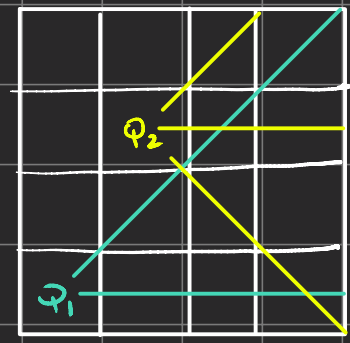


with this not possible

so think that we did some mistake while organizing Q_3, Q_2 in past

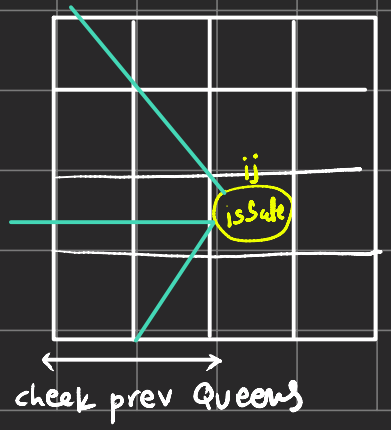
for Q_3 only one place was there for Q_2 there was two place so change

Explained more in last case



Aab Q3 ke liye jagah nahi so pichli placement mei galti hai for Q2 we tried both but not working for both ways

so piche jao aur Q1 we tried all before so leve it.



```cpp
void solve(vector<vector<int>> &board, int col, int n) {
    // Base case
    if (col >= n) {
        printSolution(board, n);
        return;
    }

    // Place queen in each row of the current column
    for (int row = 0; row < n; row++) {
        if (isSafe(row, col, board, n)) {
            board[row][col] = 1;  // Place queen
            solve(board, col + 1, n); // Recur for next column
            board[row][col] = 0;  // Backtrack
        }
    }
}
```

```cpp
void printSolution(vector<vector<int>> &board, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            // cout << board[i][j] << " ";
            cout << (board[i][j] == 1 ? "Q " : ". ");
        }
        cout << endl;
    }
    cout << endl;
}
```

```cpp
bool isSafe(int row, int col, vector<vector<int>> &board, int n) {
    // check current cell [i][j] par queen rakh sakta hoon
    int i = row, j = col;

    // Check row on left
    while (j >= 0) {
        if (board[i][j] == 1)
            return false;
        j--;
    }

    // Check upper left diagonal
    i = row, j = col;
    while (i >= 0 && j >= 0) {
        if (board[i][j] == 1)
            return false;
        i--;
        j--;
    }

    // Check lower left diagonal
    i = row, j = col;
    while (i < n && j >= 0) {
        if (board[i][j] == 1)
            return false;
        i++;
        j--;
    }

    return true;
}
```

optimization issafer → TC O(n) ⟶ how to make it O(1)

by using Map → DS. that stores key value pair and It has TC O(1)

while adding Queen

do like map[rowno] = true

so In isSafer we can check

for row like If (map[i] == true)

its True means that row has Queen

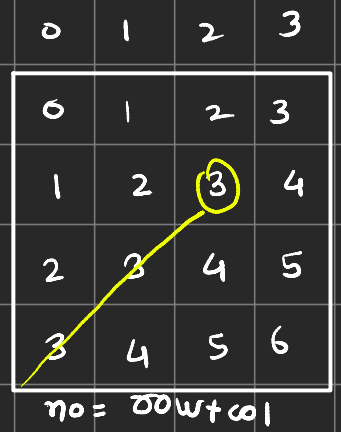
map < string, int > m ;

key value

INPUT: m["love"] = 98 | love→98

m["babbar"] = 36 | babbar→36

Access: cout << m["love"]



no = row+col

For bottomleft diagonal,

make another new map with pattern of no.

check like IF (map[i+j] == true)

its True means that row has Queen

For upperleft diagonal,

make another new map with pattern of no.

IF (map[(n-1)+col-row] == true)

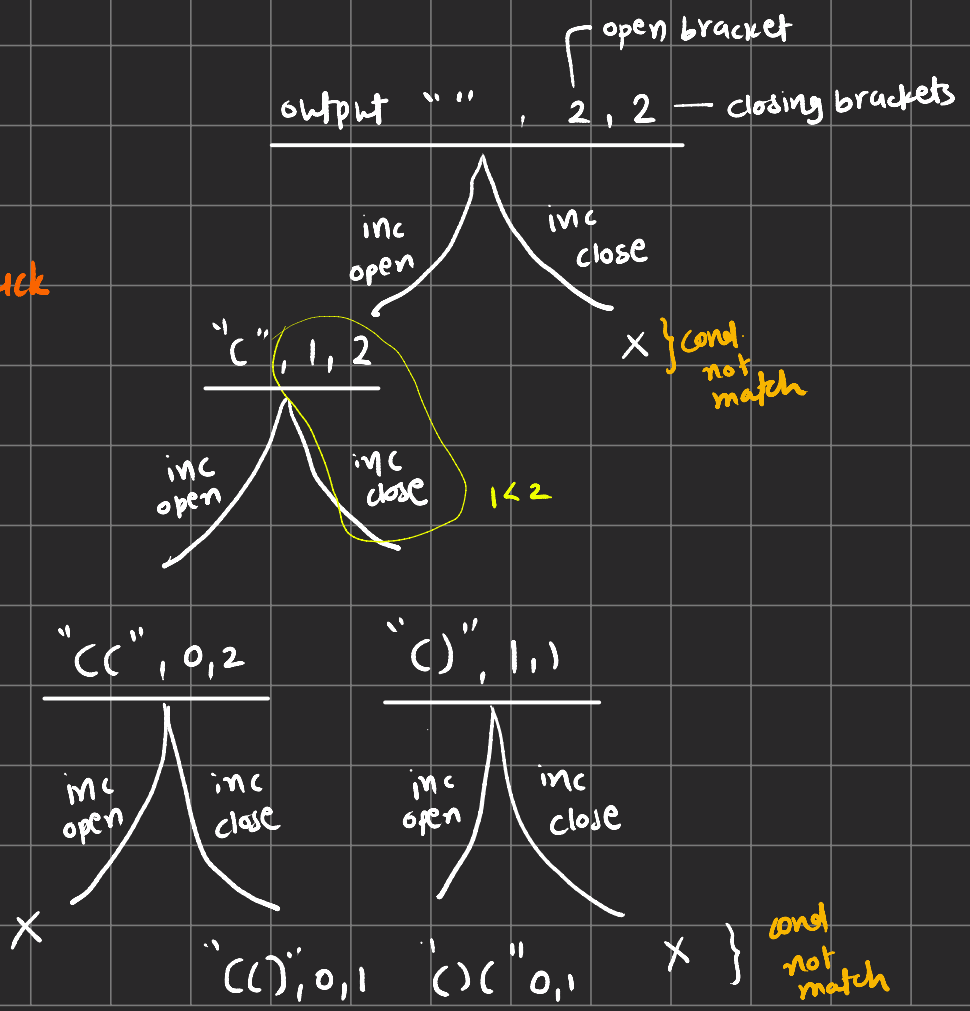
n = 4

no = (n-1)+col-row

Generate Parentheses

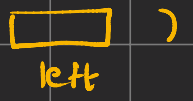
n = 1	( ) ✓	) (✗ (not paranthesis	1
n = 2	(( ))	( ) ( )	2
n = 3	((( )))	(( ) ( ))	( ) ( ) ( )	( ) (( ))	(( )) ( )	5

no of Brackets
(→ 1
) → 1



If (open > 0)
then include open bruck

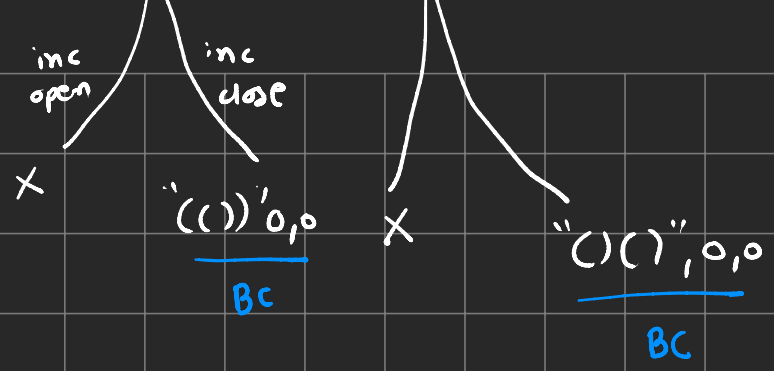
condition to include closing bracket



open brackets > close brackets

for

open < close } remaining Brackets




```cpp
void solve(vector<string> &ans, int n, int open, int close, string output){
    // base case
    if( open<=0 && close<=0 ){
        ans.push_back(output);
        return;
    }

    // rr
    // include open bracket
    if(open>0){
        // output.push_back('(');
        // solve(ans, n, open-1, close, output);
        // output.pop_back();  // backtracking

        solve(ans, n, open-1, close, output + '(');
    }

    // include close bracket
    if(open<close){
        // output.push_back(')');
        // solve(ans, n, open, close-1, output);
        // output.pop_back();  // backtracking

        solve(ans, n, open, close-1, output + ')' );
    }
}
```

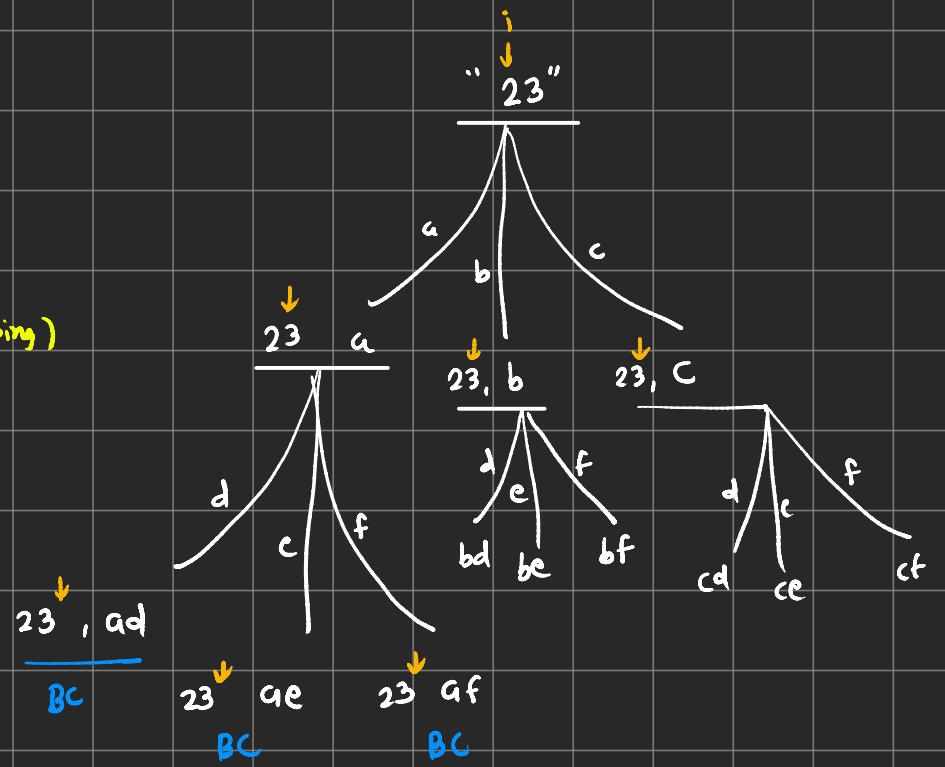
Letter combination of Phone number

```
void solve(vector<string> &ans, int index, string output, string digits, vector<string> mapping) {
    // base case
    if (index >= digits.length()) {
        ans.push_back(output);
        return;
    }

    // rr
    int digit = digits[index] - '0';    // int convert
    string value = mapping[digit];

    for (int i = 0; i < value.length(); i++) {
        char ch = value[i];
        output.push_back(ch);
        solve(ans, index + 1, output, digits, mapping);
        output.pop_back();  // backtracking
    }
}
```

solve (ans, i+1, output + ch, digits, mapping)

without BT

```
vector<string> letterCombinations(string digits) {
    vector<string> ans;
    if (digits.length() == 0) {
        return ans;
    }

    int index = 0;
    string output = "";

    vector<string> mapping(10);
    mapping[2] = "abc";
    mapping[3] = "def";
    mapping[4] = "ghi";
    mapping[5] = "jkl";
    mapping[6] = "mno";
    mapping[7] = "pqrs";
    mapping[8] = "tuv";
    mapping[9] = "wxyz";

    solve(ans, index, output, digits, mapping);

    return ans;
}
```



count Inversion

{8,4,2,1} → descending sorted array hai so maan lo ye sorted nahi hai

{8,4,2,1}
 i j

Inversion kab hoga? i<j, a[i] > a[j]

(8,4) (4,2) (2,1)
(8,2) (4,1)
(8,1)

ascending so isko sorted karne k liye kitne inversions (swapping) chiye honge

(8,4) {8,4,2,1} {4,2,1,8} (4,2)

(8,2) {4,8,2,1} {2,4,1,8} (4,1)

(8,1) {4,2,8,1} {2,1,4,8} (2,1)

{4,2,1,8} → this array more close to sorted array

{1,2,4,8} → sorted after all inversions

simple approach

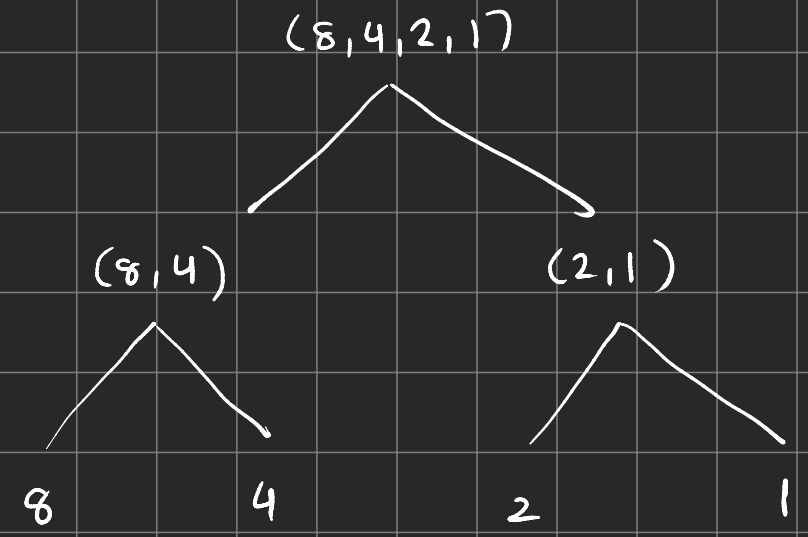
{ 8, 4, 2, 1 }
  i j

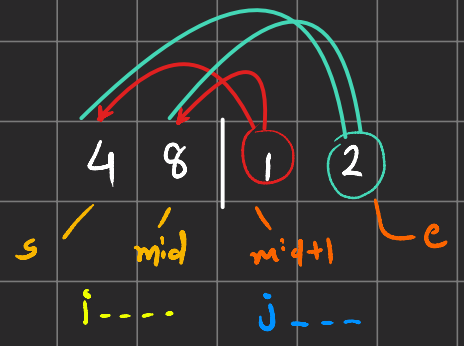
```
for (i=0 → end) {
    for (j=i+1 → end) {
        if ( i<j  &&  a[i] > a[j] ) → count++
    }
}
```

TC = $O(n^2)$

SC = $O(1)$

In Merge Sort





if (arr[i] > arr[j])

(4,1) (4,2)

(8,1) (8,2)

count = mid - i + 1

(2,1)
(6,1)

when j=0 → a[i] > a[j] = True

and during this left array is going to be sorted every time so all combinations on left side are counted as inversion

(2,1)
(6,1)

when j=1 → a[i] > a[j] = False

```
long merge(vector<int> &arr, vector<int> &temp, int s, int mid, int e){
    int i = s, j = mid + 1, k = s;

    long count = 0;

    while( i <= mid && j <= e ){
        if( arr[i] <= arr[j]){
            temp[k++] = arr[i++];
        }
        else{   // arr[j] > arr[j]  // INVERSION COUNT CASE
            temp[k++] = arr[j++];
            count = count + (mid - i + 1);
        }
    }

    while (i <= mid){
        temp[k++] = arr[i++];
    }
    while(j <= e){
        temp[k++] = arr[j++];
    }

    // copy temp array to original array
    while(s <= e){
        arr[s] = temp[s];
        s++;
    }

    return count;
}
```

```
long mergeSort(vector <int> &arr, vector <int> &temp, int s, int e){
    long count = 0;
    if(s == e){
        return 0;
    }

    int mid = s + (e - s) / 2;

    count += mergeSort(arr, temp, s, mid);
    count += mergeSort(arr, temp, mid + 1, e);

    count += merge(arr, temp, s, mid, e);

    return count;
}
```

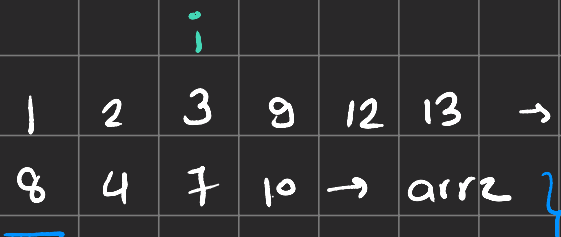
In Place Merge Sort → In merge sort we are using temp array that creates space complexity of O(n)

so how to do it with O(1)

Method ①

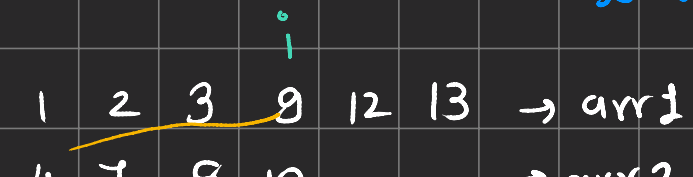
if (a[i] < a[j]) then nothing

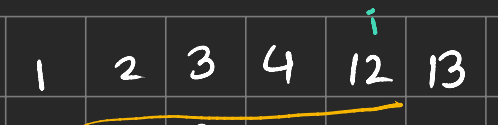
else ↳ swap (a[i], a[j])

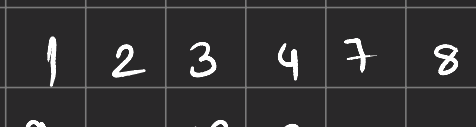


} this array is not sorted bcz of 8

so use PARTITION METHOD from quicksort.







total Length

$\lceil (n+m)/2 \rceil + ((n+m)\ \%\ 2)$

or

Method ② GAP method

$gap = \text{ceil}\left((n+m)/2\right)$

ceil do $2.7 \simeq 3$

1 2 8 9 12 13 → n=6

3 4 7 10 → n=4

if (a[i] > a[j])

↳ do swap

gap = $\frac{4+6}{2} = 5$

j = i + gap = 0 + 5 = 5

index

1 > 13 False i++ j++

2 > 3 False i++ j++

8 > 4 True swap i++ j++

9 > 7 True swap i++ j++

12 > 10 True swap i++ j++

updated GAP = ceil (gap/2) = $\frac{5}{2}$ = 2.5 ≈ 3

j = i + gap
= 0 + 3

same

1 > 7 F

7 > 3 T swap

updatedGap = ceil(gap/2) = (3/2) = 1.5 ≈ 2

do same again

1	2	4	3	8	9	7	10	13	12
0	1	2	3	4	5	6	7	8	9

updatedGap = ceil(gap/2) = (2/2) = 1

1	2	4	3	8	9	7	10	13	12

do same again

```
void mergeInPlace(int *arr, int s, int e){

    int totalLength = (e-s) + 1;
    int gap = (totalLength/2) + (totalLength%2);

    while (gap>0)
    {
        int i = s;
        int j = s+gap;

        while(j<=e){
            if(arr[i]>arr[j]){
                swap(arr[i],arr[j]);
            }
            i++;
            j++;

        }

        // gap agar 1 hai to usko break karne k liye 0 de diya
        gap = gap<=1 ? 0 : ((gap/2) + (gap%2));

    }

}
```

updatedGap = ceil(gap/2) = (1/2) = 0.5 = 1

again 1 so stop at 1

Merge two arrays

```cpp
void merge(vector<int> &a, vector<int> &b) {
    int n = a.size();
    int m = b.size();
    int total_len = n + m;
    int gap = (total_len / 2) + (total_len % 2);

    while (gap > 0) {
        int i = 0, j = gap;

        while (j < total_len) {
            // Compare elements in the first array
            if (j < n && a[i] > a[j]) {
                swap(a[i], a[j]);
            }
            // Compare elements across both arrays
            else if (j >= n && i < n && a[i] > b[j - n]) {
                swap(a[i], b[j - n]);
            }
            // Compare elements in the second array
            else if (j >= n && i >= n && b[i - n] > b[j - n]) {
                swap(b[i - n], b[j - n]);
            }

            i++;
            j++;
        }
        gap = gap <= 1 ? 0 : (gap / 2) + (gap % 2);
    }
}

int main() {
    vector<int> arrA = {1, 2, 8, 9, 12, 13};
    vector<int> arrB = {3, 4, 7, 10};

    merge(arrA, arrB);

    cout << "Merged arrays: ";
    for (int num : arrA) {
        cout << num << " ";
    }
    for (int num : arrB) {
        cout << num << " ";
    }
```

} Here arrA and arrB both are different

(inplace merge has single array that are cut into two parts using mid)

} 1,2,3,4,7,8,9,10,12,13

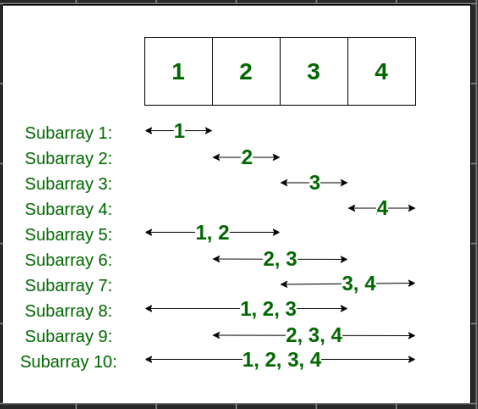
* Maximum Subarray Sum → find the subarray with the largest sum and return its sum

if all elements are positive in array so → the sum of all element is ans

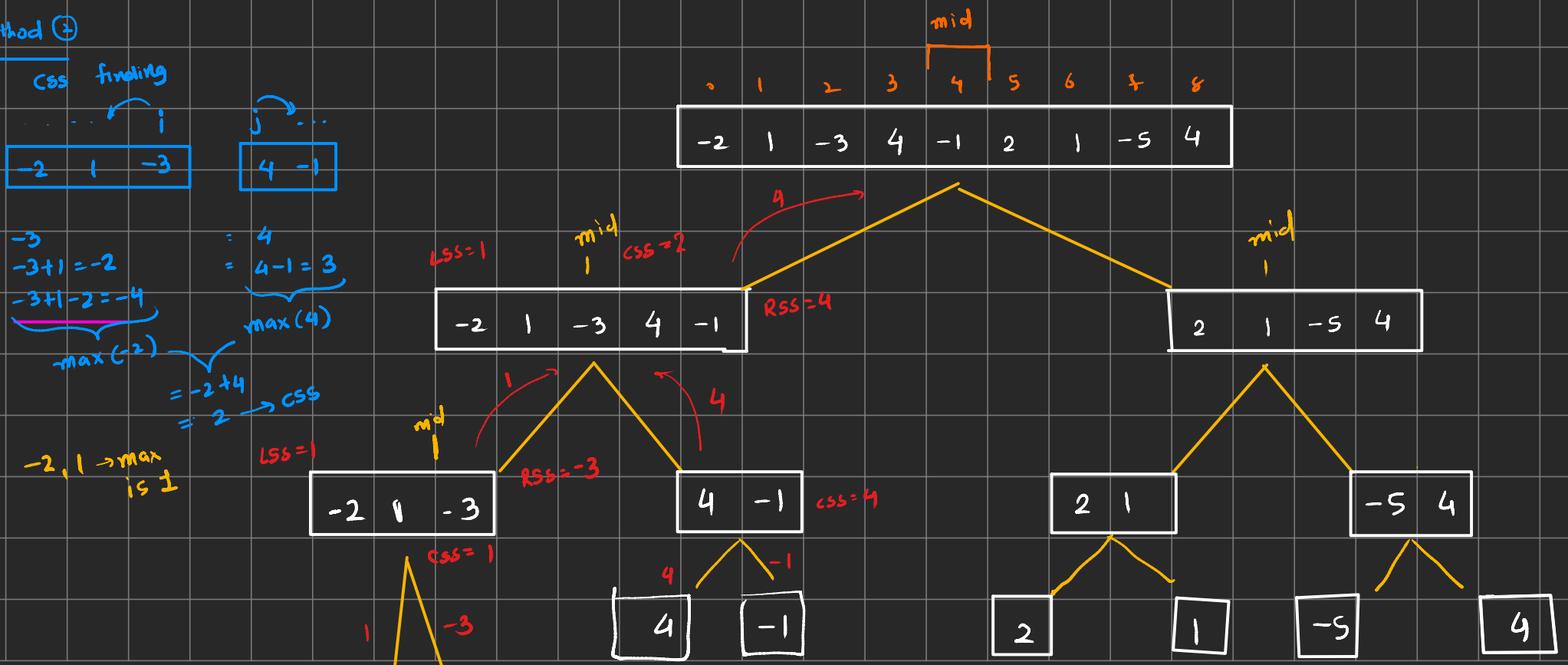
But here in qve also negative eles are there in array

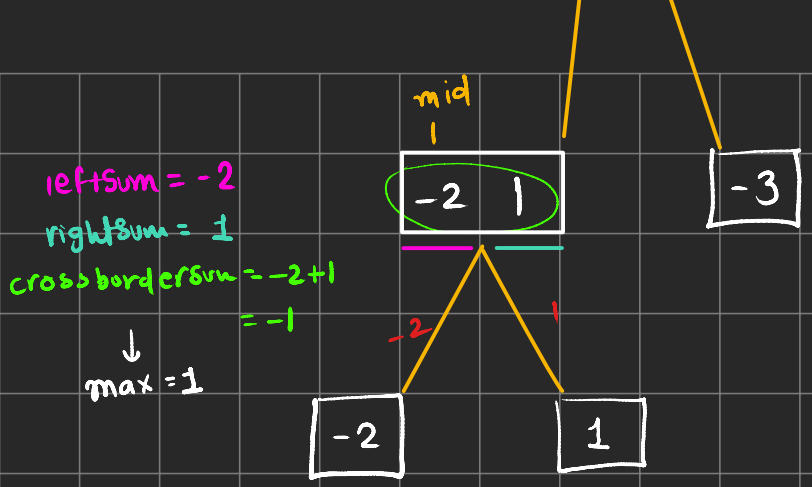
Method ① → find all subarrays
→ find max of them

```
for (i=0 → len)
    for (j=0 → len)
        for (k=i → j)
```
} O(n³)



Method ②





leftSum = -2

rightSum = 1

crossborderSum = -2+1

= -1

↓

max = 1

```cpp
int maxSubArray(vector<int> &arr, int s, int e) {
    // Base case: single element
    if (s == e) {
        return arr[s];
    }

    int maxOfLeftBorderSum = INT_MIN;
    int maxOfRightBorderSum = INT_MIN;

    int mid = (s + e) / 2;

    // Recursively find the maximum sum in the left and right halves
    int maxLeftSum = maxSubArray(arr, s, mid);
    int maxRightSum = maxSubArray(arr, mid + 1, e);

    // Calculate maximum cross-border sum
    int leftBorderSum = 0;
    for (int i = mid; i >= s; i--) {
        leftBorderSum += arr[i];
        maxOfLeftBorderSum = max(maxOfLeftBorderSum, leftBorderSum);
    }

    int rightBorderSum = 0;
    for (int i = mid + 1; i <= e; i++) {
        rightBorderSum += arr[i];
        maxOfRightBorderSum = max(maxOfRightBorderSum, rightBorderSum);
    }

    int crossBorderSum = maxOfLeftBorderSum + maxOfRightBorderSum;

    // Return the maximum of the three cases
    return max(maxLeftSum, max(maxRightSum, crossBorderSum));
}
```

Quick sort (end element as pivot)

→ Partitioning

→ pivot = end

pivot = 7

→ i = s-1

j = s

here i is right position of pi

```
void quickSort(int a[], int start, int end)
{
    if(start >= end) return;
    int pivot = end;
    int i = start - 1;
    int j = start;

    while(j < pivot){
        if(a[j] < a[pivot]){
            ++i;
            swap(a[i], a[j]);
        }
        ++j;
    }
    ++i;
    swap(a[i], a[pivot]);
    quickSort(a, start, i - 1);
    quickSort(a, i + 1, end);
}
```

→ set smaller ele left of pivot and bigger to right of pivot

```
if ( a[j] >= a[pivot] ) → j++
else → i++
     → swap(a[i], a[j])
     → j++
```

(or)

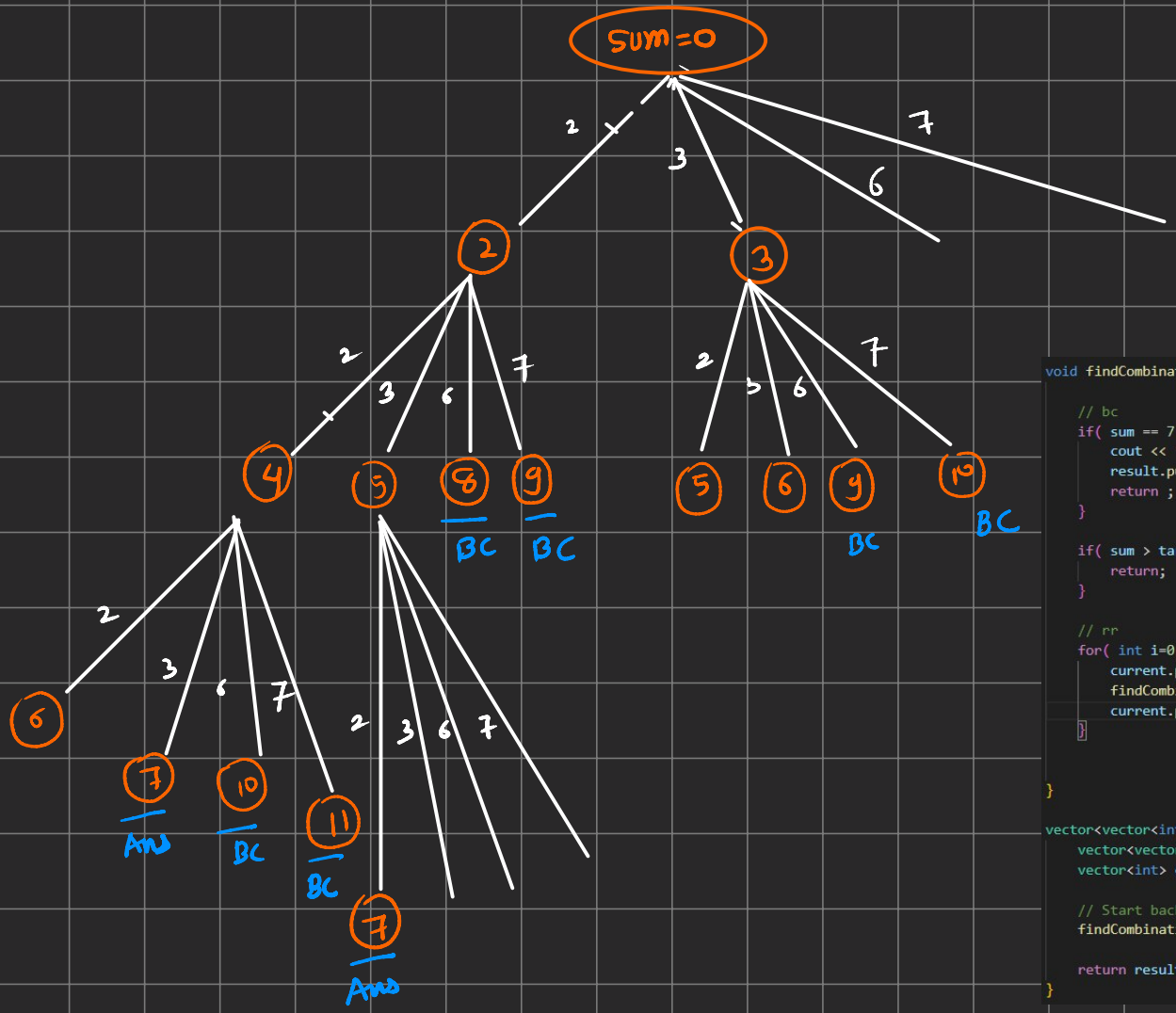
```
while (j < pivot)
{
    if (a[j] < a[pivot])
    {
        i++
        swap (a[i], a[j])
    }
}
i++
swap (a[i], a[pivot])
```

★ combination Sum

ex. 2 | 3 | 6 | 7 , target = 7

(2,2,3) → 7

(7) → 7




```cpp
void findCombinations(int sum, vector<int>& candidates, int target, vector<int>& current, vector<vector<int>>& result) {

    // bc
    if( sum == 7 ){
        cout << " matched target" << endl;
        result.push_back(current);
        return ;
    }

    if( sum > target ){
        return;
    }

    // rr
    for( int i=0; i<candidates.size(); i++){
        current.push_back(candidates[i]);
        findCombinations(sum+candidates[i], candidates, target, current, result);
        current.pop_back();
    }


}

vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
    vector<vector<int>> result;
    vector<int> current;

    // Start backtracking
    findCombinations(0, candidates, target, current, result);

    return result;
}
```

```
Combinations that sum to 7 are:
[ 2 2 3 ]
[ 2 3 2 ]
[ 3 2 2 ]
[ 7 ]
```

How to avoid this ?

```cpp
void findCombinations(int sum, vector<int>& candidates, int target, vector<int>& current, vector<vector<int>>& result, int indexNow) {

    // bc
    if( sum == target ){
        cout << " matched target" << endl;
        result.push_back(current);
        return ;
    }

    if( sum > target ){
        return;
    }

    // rr
    for( int i=indexNow; i<candidates.size(); i++){
        current.push_back(candidates[i]);
        findCombinations(sum+candidates[i], candidates, target, current, result, i);
        current.pop_back();
    }


}

vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
    vector<vector<int>> result;
    vector<int> current;

    // Start backtracking
    findCombinations(0, candidates, target, current, result, 0);

    return result;
}
```

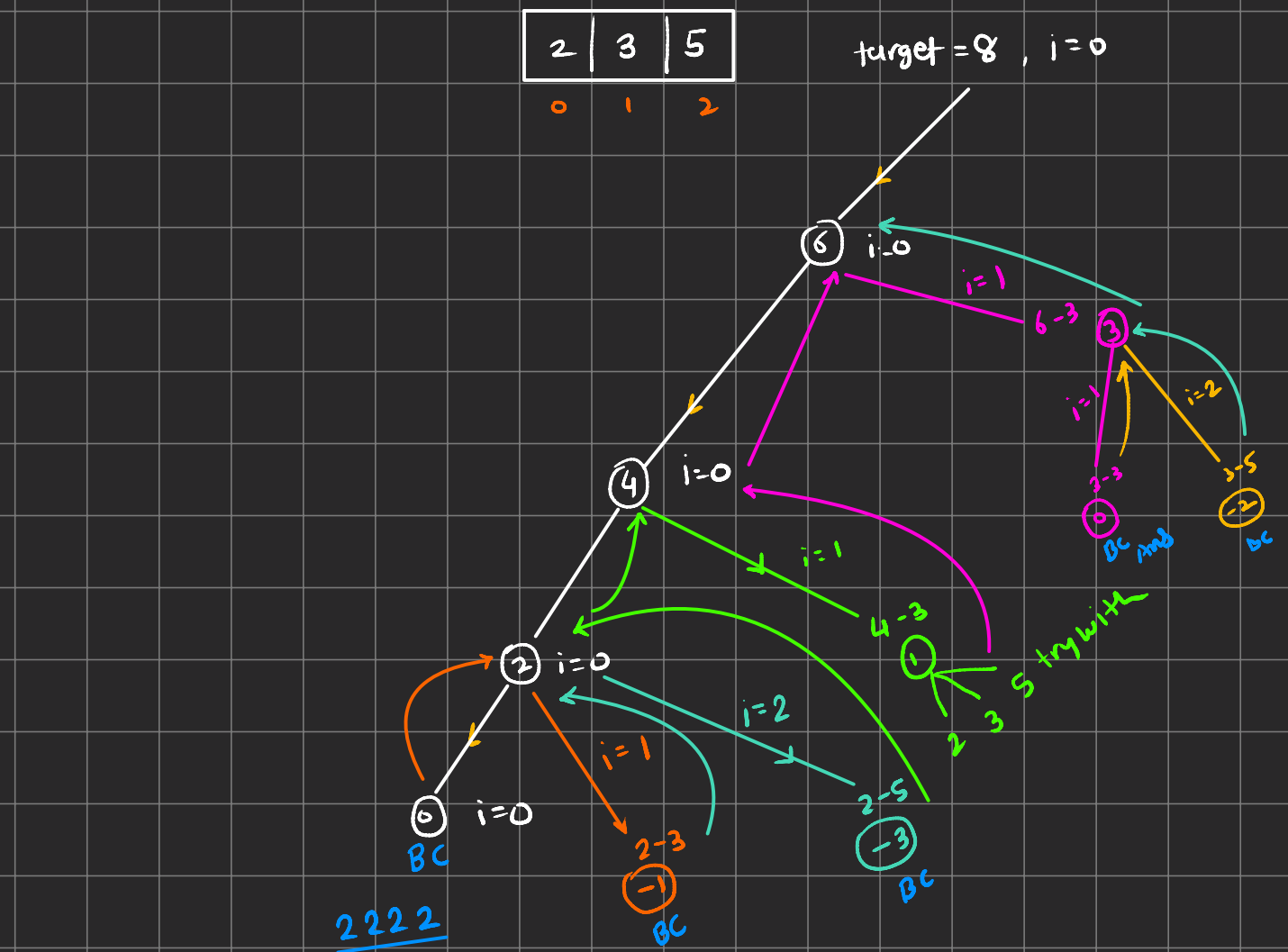
```
Combinations that sum to 7 are:
[ 2 2 3 ]
[ 7 ]
```

so every time that stops from coming with that index again like

consider that same num again

loop starts from Index

not 0 every time



Combinition Sum II

→ Each candidate only be picked / used once

→ no duplicates combination

```
void findCombinations(int sum, vector<int>& candidates, int target, vector<int>& current, vector<vector<int>>& result, int indexNow) {

    // bc
    if( sum == target ){
        cout << " matched target" << endl;
        result.push_back(current);
        return ;
    }

    if( sum > target ){
        return;
    }

    // rr
    for( int i=indexNow; i<candidates.size(); i++){
        current.push_back(candidates[i]);
        findCombinations(sum+candidates[i], candidates, target, current, result, i+1);
        current.pop_back();
    }


}

vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
    vector<vector<int>> result;
    vector<int> current;

    // Start backtracking
    findCombinations(0, candidates, target, current, result, 0);

    return result;
}
```

→ Each candidates only be used once

target=5

2	5	2	1	2

↑

After using go to next element by i+1

↓ getting

→ sort the que array so we got all outputs in same order

① 2 2 1

② 2 1 2 →

③ 5

④ 2 1 2 →

} use SET map

```cpp
vector<int> candidates = {2, 5, 2, 1, 2};
int target = 5;

// to get unique ans
sort(candidates.begin(), candidates.end());

vector<vector<int>> result = combinationSum(candidates, target);

// using SET map to get unique ans
set<vector<int>> st;
for(auto e: result){
    st.insert(e);
}

result.clear();

for(auto ele: st){
    result.push_back(ele);
}
```

} gives TLE

```cpp
for(int i=index;i<candidates.size();i++){
    if(i > index && candidates[i] == candidates[i - 1]){
        continue;
    }
    v.push_back(candidates[i]);
    combinationSum_helper(candidates,target-candidates[i],v,ans,i + 1);
    v.pop_back();
}
```

In sorted que

when we are here it is going to give same ans as that (i-1) if i = i-1 same element

Permutation - II → no duplicates allowed

→ can use set method as above but taking long time

```
void permuteUniqueHelper(vector<int>& nums, vector<vector<int>>&ans, int start){
    //base
    if(start == nums.size()){
        ans.push_back(nums);
        return;
    }

    unordered_map<int, bool>visited;
    for(int i=start;i<nums.size();i++){
        if(visited.find(nums[i]) != visited.end()){
            continue;
        }
        visited[nums[i]] = true;
        swap(nums[i], nums[start]);
        permuteUniqueHelper(nums, ans, start + 1);
        swap(nums[i], nums[start]);
    }
}
```

} means → num[i] ko ekbaar start k saath swap kiya hai to dubara nahi karunga to avoid duplicates

Beautiful Arrangement

`1 <= i <= n` means — one indexed array

→ `perm[i]` is divisible by `i` → `perm[i] % i == 0`

OR `i` is divisible by `perm[i]` → `i % perm[i] == 0`

Brute Force

→ 1 3 2

Index 1 2 3

`1 % 1 == 0` ✓ | `1 % 1 == 0` ✓

`3 % 2 ≠ 0` X | `2 % 3 ≠ 0` X → not beautiful arrangement

→ 3 2 1

1 2 3

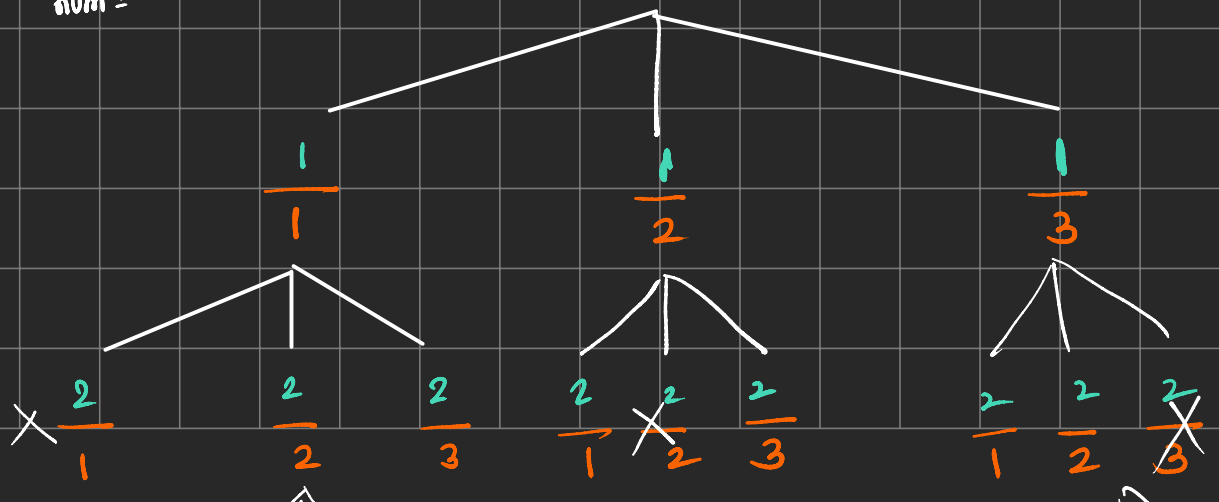
`3 % 1 == 0` ✓ | `1 % 3 ≠ 0` X

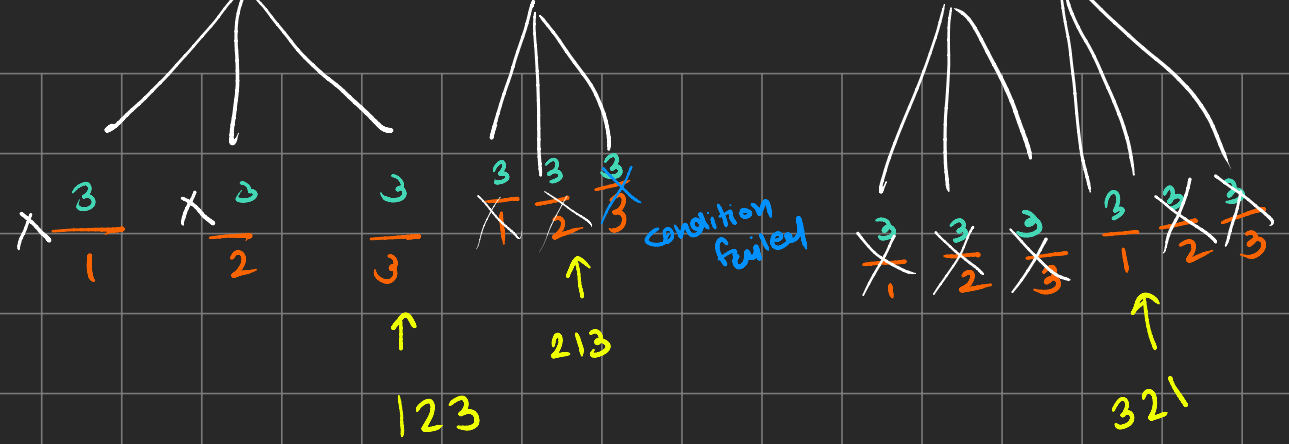
`2 % 2 == 0` ✓ | `2 % 2 == 0` ✓

`1 % 3 ≠ 0` X | `3 % 1 == 0` ✓

so, Yes BA.

Optimal (BT) num =





Ans = 3 Beautiful Arrangements

```cpp
void counterHelper(vector<int>& v, int& n, int& ans, int currNum) {
    // Base case
    if (currNum == n + 1) {
        ans++;
        for (int i = 1; i <= n; i++) {
            cout << v[i] << " ";
        }
        cout << endl;
        return;
    }

    // Try placing `currNum` in each valid position
    for (int i = 1; i <= n; i++) {
        if (v[i] == 0 && (currNum % i == 0 || i % currNum == 0)) {
            v[i] = currNum; // Place `currNum`
            counterHelper(v, n, ans, currNum + 1); // Recurse for the next number
            v[i] = 0; // Backtrack
        }
    }
}

int countArrangement(int n) {
    vector<int> v(n + 1, 0); // Vector to track used positions (1-based indexing)
    int ans = 0;             // To count valid arrangements
    counterHelper(v, n, ans, 1);
    return ans;
}
```

Distributing Repeating Integer

1	1	2	2
0	1	2	3

→ nums

2	2
0	1

→ quantity

customer 0 wants 2 int and that should be same everytime → so I am allocating 1, 1 (this both should be same)

mean total two integers / nums

customer 1 wants 2 int and that should be same everytime → so 2, 2

1	1	2	2	2	2	3	3

→ nums

cus0 cus1 cannot allocate to cus2 cannot allocate to cus2

2	2	3
0	1	2

→ quantity

cus0 → 1 1

cus1 → 2 2

cus2 → needs three nums (cannot allocate)

BackTrack Here

so again with new allocation order

cus 0 → | 1 | 1 |

cus 1 → | 3 | 3 |

cus 2 → | 2 | 2 | 2 |

↳ done

1	1	2	2	2	2	3	3

cus 0 cus 2 cus 1

1	1	2	2	2	2	3	3

num	howmany
1	2
2	4
3	2

counts

1	2	3
2	4	2
0	1	2

quantity

2	2	3
0	1	2

index of quantity, index of counts

(0, 0)

→ quantity[0] wants two nums that can be satisfied with counts[0] which has 2 num

(1, 1)

(1, 2)

0

~~2~~	4	2

2

0	~~4~~	2

0

0	4	~~2~~

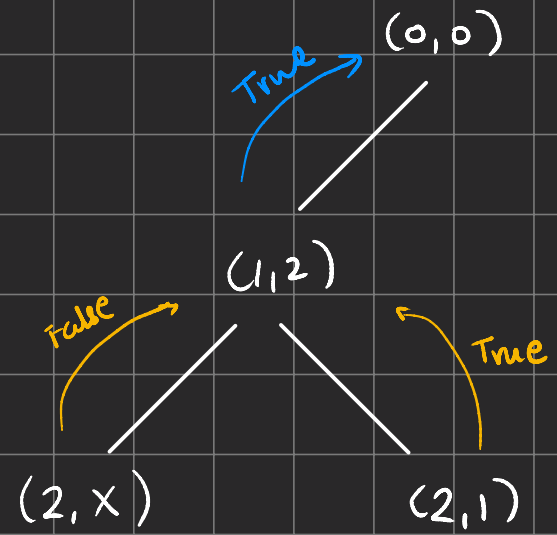
False BK

0	2	2

(2,)

↳ we need three nums that are donot have in counts arr.

so try with different order here we need to do Backtrack



optimized

sort quantity array in decreasing order

3	2	2

because sabse highest quantity ke satisfied nahi hogi to jaldi false nikal paunga

```cpp
bool canDistributeHelper(vector<int> &counts, vector<int> &quantity, int ithCustomer) {
    // Base case: all customers are satisfied
    if (ithCustomer == quantity.size()) {
        return true;
    }

    // Try to satisfy the ith customer
    for (int i = 0; i < counts.size(); i++) {
        if (counts[i] >= quantity[ithCustomer]) {
            counts[i] -= quantity[ithCustomer]; // Deduct quantity for the current custome

            if (canDistributeHelper(counts, quantity, ithCustomer + 1)) {
                return true;
            }

            counts[i] += quantity[ithCustomer]; // Backtrack
        }
    }

    return false;
}

bool canDistribute(vector<int> &nums, vector<int> &quantity) {
    unordered_map<int, int> countMap;

    // Count frequency of each number in nums
    for (int num : nums) {
        countMap[num]++;
    }

    // Convert frequency map to a vector
    vector<int> counts;
    for (auto it : countMap) {
        counts.push_back(it.second);
    }

    // Sort quantities in descending order to optimize backtracking
    sort(quantity.rbegin(), quantity.rend());

    return canDistributeHelper(counts, quantity, 0);
}
```