

If you have any questions or suggestions regarding the notes, please feel free to reach out to me on WhatsApp: **Sanjay Yadav Phone: 8310206130**

Notes -> DSA: Hashing: 1 - 20 - May 2023

- Need of HashMap
- Syntax Of HashMap
- Count Frequency
- Non Repeating Element
- HashSet
- distinct element

Note - Whenever we solve problems using prefix sum with HashMap or HashSet, we should include 0 as the first value. If using a HashSet, add 0 as a key; if using a HashMap, add a key-value pair of (0, -1). This approach covers the edge case where the prefix sum is 0. By following this method, we ensure that if the prefix sum becomes 0 at any point, it will be present in the map; otherwise, we may not get the desired result.

2. To find the longest length subarray with a sum of 0, we'll employ a HashSet and prefix sum. In this approach, the length of the subarray is typically calculated as right - left + 1. However, if two prefix sums in the range are equal, it means the sum of elements between those positions is 0, so in such cases, we adjust the length formula to be right - left to include all elements in the range without counting the element at the "right" position twice.

foreach loop ->

```
for(int ele:array){  
    System.out.print(ele);  
}
```

- Need of HashMap

1. Need of Hashmap:

student's roll no	marks
57	85
62	92
11	59
12	96
7	89

marks:

85	92	59	96	89
0	1	2	3	4

Create an array of size 62
marks:

		89	59	96	85	92	
0	1	...	7	..11	..12	..57	..62

marks[11] = 59
marks[62] = 92
marks[12] = 96
→ wastage of memory

→ Retrieval of marks for any specific roll no. is difficult.

for eg. marks of roll no=11 ?

For these kind of data [key and value pair is available]
we will deal with hash map to store that information.

Syntax of Hashmap:

datatype of value
HashMap< Integer, Integer > map = new HashMap<>();
 ^
 | datatype of key
 | variable name

Syntax Of HashMap -> Whenever we have to store key value pair data we will use HashMap.

HashMap<key datatype, value datatype> variableName = new HashMap<>();

HashMap<Integer, Integer> map = new HashMap<>();

- Add values, **map.put(key, value);** -> If value is not there it will add if it is there it will update exiting value.
- Get Values **map.get(key);** -> If value is present it will return **value** else **null**

- Check Key available `map.containsKey(key)`; -> If key is present it will return **true** else **false**
- Remove key value `map.remove(key)`; -> If key is present it will remove key value pair and return **value** associated with that key. If not present then it will return **null**
- Count keys `map.size()` -> count keys in map Hashmap
- get all keys `map.keySet()` -> It will give you all keys as array

Problem 1: Count Frequency

Given an array A and query Q. find how many time $Q[i]$ is available in array.

(n) A:

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

(q) Q →

2	1	2	4	6	5
0	1	2	3	4	5
↓	↓	↓	↓	↓	↓
2	2	3	1	1	0

Idea 1: Brute force Approach
for every possible value of $Q[i]$ → search frequency in Array $A[i]$

T.C: $O(q * n)$
S.C: $O(1)$

Idea 2: ① Generate frequency map using Hashmap → $O(n)$

② Iterate on every query $Q[i]$ and get $\boxed{\text{frequency}}$ of that particular element → $O(q)$

Hashmap:

Element - frequency of Element

2 →	2	2
1 →	1	2
3 →	1	1
4 →	1	1
6 →	1	1

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

↑
i

3	2	2	1	1	0
2	1	2	4	6	5

↑
i

T.C = $O(n + q)$
S.C: $O(n)$

```

public static void frequencyOfEle(int[] A, int[] Q) {
    // make a frequency map
    HashMap<Integer, Integer> fmap = new HashMap<>();
    /*
     * Key -> element of array A
     * Value -> frequency of that element
     */
    for(int ele : A) {
        if(fmap.containsKey(ele) == false) {
            // insert new pair of ele with freq = 1
            fmap.put(ele, 1);
        } else {
            // increase old freq by 1
            int oldFreq = fmap.get(ele);
            fmap.put(ele, oldFreq + 1);
        }
    }
    // Iterate on every query and just get the freq. from fmap
    for(int val : Q) {
        if(fmap.containsKey(val) == true) {
            System.out.println(val + " : " + fmap.get(val));
        } else {
            System.out.println(val + " : " + 0);
        }
    }
}

```

Probelm2 - Non Repeating Element

Given an array A first non repeating element

A: [2,5,4,5,2,6]

Non repeating -> 4,6

First Not repeating -> 4

```

TODO → task
snippet    int    nonRepeating (int[] A) {
                // TODO: Return that Element
}

```

Approach:

- ① Create a freq. map.
- ② Go back on given array &
at first instance if freq. @
ele == 1 that means that
element is first non-repeating.

Syntax Of HashSet -> Whenever we have to store only unique values we will use it.

```
HashSet<key datatype> variableName = new HashSet<>();
```

```
HashSet<Integer> set = new HashSet<>();
```

- Add values, **set.add(value);**
- **If we print it directly it will print value not address**
- Get Values **It has no get function**
- Check value available **set.contains(value);** -> If value is present it will return **true** else **false**
- Remove value **set.remove(value);** -> If value is present it will remove value and return **true**. If not present then it will return **false**
- Count element **set.size()** -> count keys in map Hashmap

Probelm3 - Given an array A, find distinct element available in array.

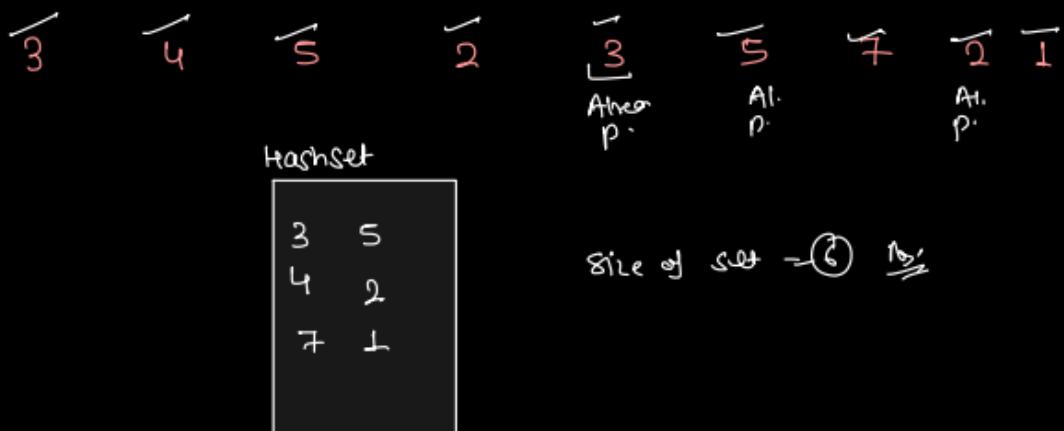
A: 3,9,3,4,5

gdea ① using Hashmap:

- create a freq. map
- map.size() is answer
- if required → keys of map are distinct Element.

gdea 2 using HashSet:

- Add Every Element in set



DSA: Hashing-2 - 22 - May 2023

- Target Sum Pair
- count of pair sum
- Subarray with Sum K
- Count of subarray having sum K

Problem1 - Given an array A[]. Find if there exist a pair such that $A[i]+A[j] == k$ and $i!=j$.

Problem2 : Given an array A[] and value k. Find count of pairs such that sum of $A[i] + A[j] == k$ and where $i!=j$.

Problem3 : Given an array. check if there exist a subaaray with sum k or not

Problem4 - Given an array, count total number of subarray having sum = k.

DSA: Recursion-1- 27 - May 2023

- Call Stack
- Recursion - Start
- Factorial using recursion
- nth fibonacci using recursion
- Euler Diagram OR tree diagram
- Print Increasing number
- Check is subarray of character is palindromic or not
- Recursion -> function make a call to itself is know as recursion.

How to Apply Recursion :

1. Assumption : What the function is going to do ?

2. Main logic : Solving main logic using just smaller subproblems
which Q.S of same type problem.

3. Base condition : when recursion should stop
OR

smallest problem which we are aware from .

```

Assumption : Sum from 1 to n
int sum( int n) {
    // Base case
    if(n==1) {
        return 1;
    }
    // Main logic
    int temp = sum(n-1);
    return temp + n;
}

void main() {
    int n= scn.nextInt();
    System.out.println(sum(n));
}

```

Assumption: Given 'n', return sum for n - natural number.

Main logic:

$$\text{sum}(n) = \underbrace{\text{sum}(n-1)}_{\text{temp}} + \underline{n}$$

Base case: Smallest problem natural number, smallest problem, $n=1$

$$\text{sum}(1) = 1$$

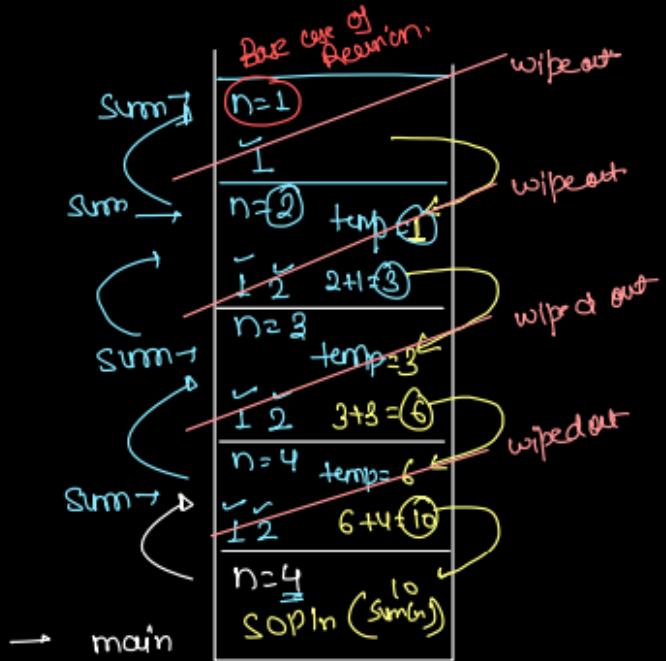
DRY RUN 1:

```

Assumption : Sum from 1 to n
int sum( int n) {
    // Base case
    if(n==1) {
        return 1;
    }
    // Main logic
    int temp = sum(n-1);
    return temp + n;
}

void main() {
    int n= scn.nextInt();
    System.out.println(sum(n));
}

```



Op: 10 Pg:

Problem2: Given the value of n, calculate n factorial using recursion.

$$n! = n * (n-1) * (n-2) * \dots * 3 * 2 * 1$$

$$(n-1)! = (n-1) * (n-2) * (n-3) * \dots * 3 * 2 * 1$$

$n! = n * (n-1)!$

$$5! = 5 * 4 * 3 * 2 * 1$$

$$4! = 4 * 3 * 2 * 1$$

→ Relation:

$$5! = 5 * 4!$$

NOTE:
smallest problem
for factorial is
[n=1, 1!=1]
n=0, 0!=1

// Assumption

```
int fact (int n) {
    // Base case
    if(n==0) {
        return 1;
    }
    // Main logic
    int temp = fact(n-1);
    return n * temp;
}
```

Assumption: Given value of n,
return factorial of n.

Main logic:

$$\text{factorial}(n) = \underbrace{n *}_{\text{main problem}} \underbrace{\text{factorial}(n-1)}_{\substack{\text{just smaller} \\ \text{subproblem}}};$$

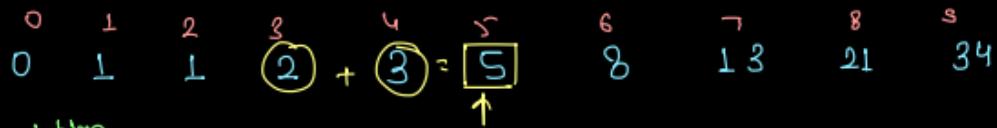
Base case:

$1! = 1$ $\boxed{\text{if}(n==1)}$ $\boxed{\text{return } 1;}$	$0! = 1$ $\boxed{\text{if}(n==0)}$ $\boxed{\text{return } 1;}$
--	--

not needed wrap work
with this
single base case.

Problem3: Given n, find nth fibonacci number (Using recursion)

- in this first 2 numbers are fixed and for next number we will plus first and second and create 3 same for next we will take second and third and create 4th number.



// Assumption

```
int fib(int n) {
    // Base case
    if(n == 0 || n == 1) {
        return n;
    }
    // main logic
    int temp1 = fib(n-1);
    int temp2 = fib(n-2);
    return temp1 + temp2;
}
```

Assumption: Given n , find fibbonacci by n^{th} index

Main logic: $\text{fib}(5) = \text{fib}(4) + \text{fib}(3)$

$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$

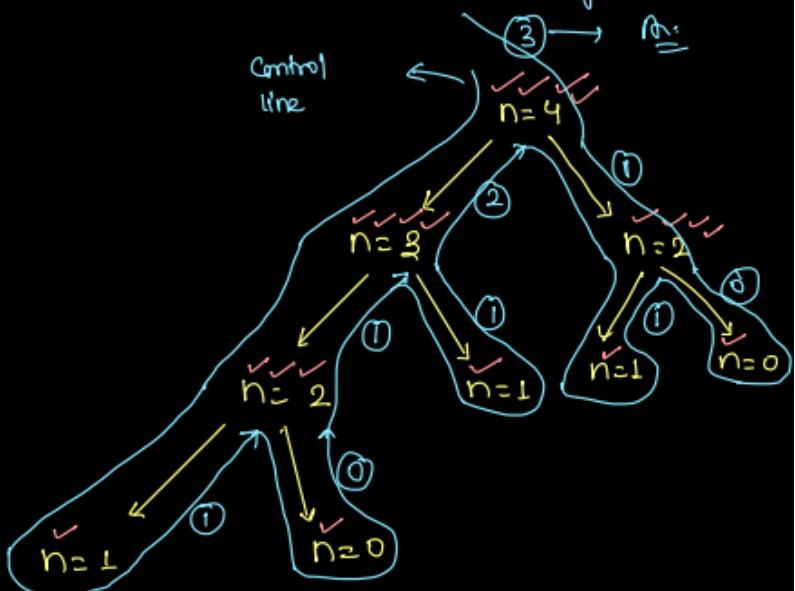
Base case:

```
if(n == 0 || n == 1) {
    return n;
}
```

If more than one call is available, dry run with stack diagram is slightly complex.

→ To avoid that complexity we have Euler Diagram.

OR
tree diagram.



// Assumption

```
int fib(int n) {
    // Base case
    if(n == 0 || n == 1) {
        return n;
    }
    // main logic
    int temp1 = fib(n-1);
    int temp2 = fib(n-2);
    return temp1 + temp2;
}
```

Problem4: Given n, print values from 1 to n [Increasing order] solve using recursion

$n=5 \rightarrow 1 \ 2 \ 3 \ 4 \ 5$ $n=4 \rightarrow 1 \ 2 \ 3 \ 4$ $\left[\begin{array}{l} \text{print}(5) = 1 \ 2 \ 3 \ 4 \ 5 \\ \text{print}(4) = 1 \ 2 \ 3 \ 4 \end{array} \right]$ $\hookrightarrow \text{print}(5) = \left[\begin{array}{l} \text{print}(4) \\ + \\ \text{sopln}(5) \end{array} \right]$	<p style="text-align: right;"><i>Using Recursion</i></p> <p><u>In generic</u></p> $\text{print}(n) \rightarrow \text{print}(n-1) + \text{sopln}(n)$				
<p><i>// Assumption</i></p> <pre>void gncPrint (int n) { // Base case if (n == 0) { return; } // main logic gncPrint(n-1); sopln(n); }</pre>	<p><i>print(n) → 1 2 3 ... n</i></p> <p><i>Assumption: Given n, print numbers from 1 to n</i></p> <p><i>Main logic:</i></p> $\text{gncPrint}(n) \rightarrow \text{gncPrint}(n-1); \text{sopln}(n);$ <p><i>base case: (Both are valid)</i></p> <table border="1" style="margin-left: auto; margin-right: auto; border-collapse: collapse;"> <tr> <td style="padding: 5px; vertical-align: top;"> $n=1$ $\left\{ \begin{array}{l} \text{if}(n==1) \\ \text{sopln}(1) \\ \text{return}; \end{array} \right.$ </td> <td style="padding: 5px; vertical-align: top; text-align: center;"> $\overbrace{\text{if}(n==0)}^{\text{n=0 parallel}} \quad \underline{\underline{\text{if}(n==0)}}$ </td> </tr> <tr> <td style="padding: 5px; vertical-align: top;"> $\left\{ \begin{array}{l} \text{if}(n==0) \\ \text{return}; \end{array} \right.$ </td> <td style="padding: 5px; vertical-align: top;"></td> </tr> </table>	$n=1$ $\left\{ \begin{array}{l} \text{if}(n==1) \\ \text{sopln}(1) \\ \text{return}; \end{array} \right.$	$\overbrace{\text{if}(n==0)}^{\text{n=0 parallel}} \quad \underline{\underline{\text{if}(n==0)}}$	$\left\{ \begin{array}{l} \text{if}(n==0) \\ \text{return}; \end{array} \right.$	
$n=1$ $\left\{ \begin{array}{l} \text{if}(n==1) \\ \text{sopln}(1) \\ \text{return}; \end{array} \right.$	$\overbrace{\text{if}(n==0)}^{\text{n=0 parallel}} \quad \underline{\underline{\text{if}(n==0)}}$				
$\left\{ \begin{array}{l} \text{if}(n==0) \\ \text{return}; \end{array} \right.$					

Prolem5: Given a char[] A and start index 's' and end index 'e' find if that subarray (from s to e) is palindromic or not

A: a b t t b c a
 0 1 2 3 4 5 6
 ↑ ↑ ↑
 s 4 5
 s=1, e=4 → true
 s=2, e=5 → false.

A: b s=0 e=0] → true
 0 ↑ ↑
 s e

a b c c b a
 ↑ ↑
 s e

```
if (A[s] != A[e]) {
    return false;
}
```

```
if (A[s] == A[e]) {
    return isPalind(A, s+1, e-1);
}
```

```
boolean isPalindromic (char[] A , int s, int e) {
```

(s==e) return true
s > e → return

```
if(s==e || s>e)
    return true;
```

```
if(A[s] != A[e])
    return false;
```

```
} else {
    boolean ans= isPalindromic(A,s+1,e-1);
```

```
return ans;
```

```
}
```

```
}
```

// Assumption

```
void gncPrint (int n) {
```

// Base case

```
1. [ if (n == 0) {
```

 | return n;

 | }

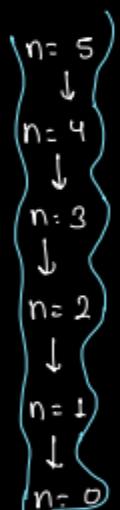
 // main logic

```
2. [ gncPrint(n-1);
```

```
3. [ sopln(n);
```

3

tree Diagram:



o/p screen:

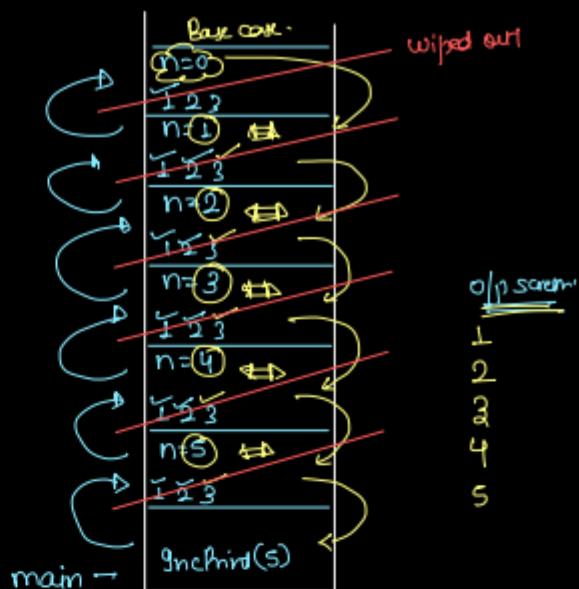
1

2

3

4

5



NOTE: Because of single call, Euler Diagram is linear

To Do: Dry Run it

& observe it & figures out
some conclusion from
problem ④.

```
void fun (int n) {
```

 if (n == 0) {

 | return;

 | }

 fun(n-1);

 }

3

[use Revision]

qst → loop \equiv $\frac{\text{loop}}{\text{loop}} \in \text{loop}$

<u>TODOS</u> (Dry Run)										train
A: a b c n f g n c b a γ										
g1 [A: 0 1 2 3 4 5 6 7 8 9 10] s = 0 e = g										
g2 [A: n f h k r t γ k h s p] s = .										

Notes -> DSA: Recursion 2 - 30 - May 2023

Agenda =>

1. Stack Overflow
2. sum of digit using recursion
3. calculate a^n
4. calculate $a^n \% p$
5. Time Complexity of recursive code

1. Stack Overflow -----

Reason behind stack overflow

- If base case is not there
- If base case is present but value is high in such a way that we are causing the limit it will throw stack overflow.
- Stack range
in C++ -> 10^5 levels approx more than 2.4 lac
in Java -> 10^4 level approx 10K

2. Sum of digit using recursion

Problem 1: Given n , ($n > 0$), find sum of all digits.

$$N = 1104 \rightarrow 6 \quad \text{sum} = 1+1+0+4 = 6$$

$$N = 128 \rightarrow 11, \quad \text{sum} = 1+2+8 = 11$$

$$\text{sum}(1248) = \frac{\text{sum}(124)}{10} + 8$$

Assumption: Given n , return sum of digits of n .

Math logic:

$$\text{sum}(n) = \text{sum}(n/10) + (n \% 10)$$

base case:

$$\text{if}(n=0)\{ \quad \text{return } 0; \}$$

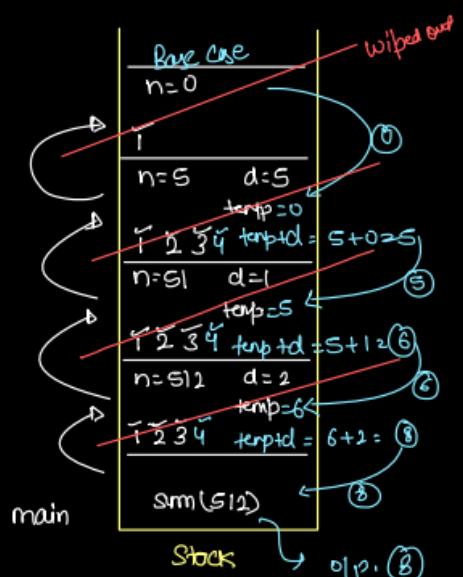
```
int sumOfDigit(int n){  
    if(n==0){  
        return 0;  
    }  
    int d = n%10;  
    int temp = sumOfDigit(n/10);  
    return temp+d;  
}
```

3

```
int sumOfDigit(int n){  
    if(n==0){  
        return 0;  
    }  
    int d = n%10;  
    int temp = sumOfDigit(n/10);  
    return temp+d;  
}
```

3

$$512 = 5 + 1 + 2
= 8 \quad \text{Ans}$$



TODO: Dry Run

$$\textcircled{1} \quad n=418$$

$$\textcircled{2} \quad n=926$$

$$\textcircled{3} \quad n=3142$$

$$\textcircled{4} \quad n=10034$$

3. calculate a^n

Problem 2: Given 'a' and 'n'. calculate a^n .

$$a=2, n=3 \rightarrow a^n = 2^3 = 8$$

$$a=3, n=4 \rightarrow a^n = 3^4 = 81$$

$$a=1, n=100 \rightarrow a^n = 1^{100} = 1$$

$$a=200, n=0 \rightarrow a^n = 200^0 = 1$$

$$\text{pow}(2, 5) = \underbrace{2 *}_{\text{Main problem}} \underbrace{\text{pow}(2, 4)}_{\substack{\text{just} \\ \text{smaller} \\ \text{problem}}}$$

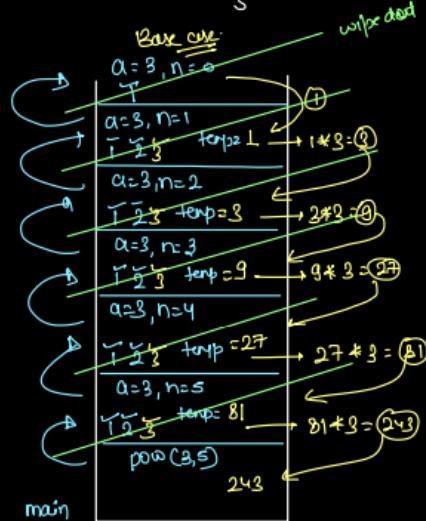
```
int pow (int a, int n) {
    if (n == 0) {
        return 1;
    }
    int temp = pow(a, n-1);
    return temp * a;
}
```

Assumption: Given a, n
return a^n .

Main logic:
 $a^n = a * a^{n-1}$
 $\text{pow}(a, n) = a * \text{pow}(a, n-1)$

Base case:

```
if (n == 0) {
    return 1;
}
```



```
int pow (int a, int n) {
    if (n == 0) {
        return 1;
    }
    int temp = pow(a, n-1);
    return temp * a;
}
```

3

$$a=3 \\ n=5 \rightarrow \underbrace{3 * 3 * 3 * 3}_{9 * 9 * 9} = 81 * 3 = 243$$

$$\text{O(p. } 243) = 3^3$$

Observation:

$$5^8 \Rightarrow 5^4 * 5^4$$

$$5^9 \Rightarrow 5^4 * 5^4 * 5$$

$$a^n \rightarrow \begin{cases} a^{\frac{n}{2}} * a^{\frac{n}{2}} & [\text{if } n \text{ is even}] \\ a^{\frac{n-1}{2}} * a^{\frac{n-1}{2}} * a & [\text{if } n \text{ is odd}] \end{cases}$$

int pow (int a, int n) {

1. { if ($n == 0$) {
| | return 1;
| } }

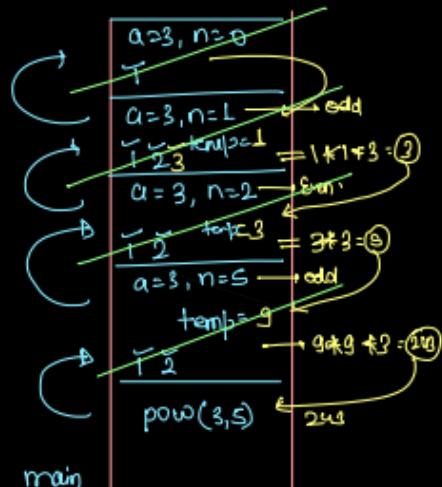
2. { int temp = pow(a, n/2);

if ($n \% 2 == 0$) {
| | return temp * temp;

2. { 3 else {
| | return temp * temp * a;

3

$$3^5 = 243$$



a=2, n=8 [Analysis of Depth of Recursion]

power 1 code

pow(2, 8) \rightarrow 256

↓
pow(2, 7) \rightarrow 128

no. of
levels = n
levels

↓
pow(2, 6) \rightarrow 64

↓
⋮

↓
pow(2, 0) \rightarrow 1

power 2 code

pow(2, 8) \rightarrow 256

↓
pow(2, 4) \rightarrow 16

no. of
levels = log n
levels

↓
pow(2, 2) \rightarrow 4

↓
pow(2, 1) \rightarrow 2

↓
pow(2, 0) \rightarrow 1

4. calculate $a^n \% p$

Problem 3: Given a , n & p . calculate $a^n \% p$. $\lceil (a^b)^c = a^{b+c} \rceil$

$$1 \leq a \leq 10^9$$

$$1 \leq b \leq 10^5$$

$$1 \leq p \leq 10^9$$

$$a = 10^0 = 10^2$$

$$b = 10$$

$$a^b = (10^2)^{10} = 10^{20}$$

```

long int pow (int a, int n, int p) {
    if(n == 0) {
        return 1;
    }
    long temp = pow(a, n/2, p);
    if(n % 2 == 0) {
        return (temp * temp) % p;
    } else {
        return (temp * temp * a) % p;
    }
}

```

$(a * b) \% p$
 $= ((a \% p) * (b \% p)) \% p$

~~o to p-1~~ ($A + \text{mor} \leq 10^9$)
 $\rightarrow (temp * temp \% p) \% p$
 we can store Result
 $(A * B) \% p = (A \% p * B \% p) \% p$
 $= (((temp * temp \% p) \% p) * (a \% p)) \% p$

5. Time Complexity of recursive code

Time complexity of Recursive Code:

Recursive Code: A function which is getting call multiple time from itself is recursive code.

T.C. for single function = α

total Number of function call = γ

Overall T.C. = $\alpha * \gamma$

```
int sum( int n) {
    if(n == 1) {
        | return n;
    }
    int temp = sum(n-1);
    return temp+n;
}
```

Time complexity of
single function = $O(1)$

Total no. of calls = n

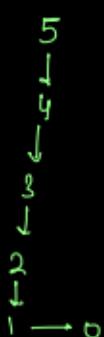


$$\begin{aligned} \text{T.C.} &= 1 * n \\ &\equiv O(n) \end{aligned}$$

```
int fact( int n) {
    if( n == 0) {
        | return 1;
    }
    int temp = fact(n-1);
    return temp*n;
}
```

T.C. of single function = $O(1)$
No. of calls = $n+1$

$$\begin{aligned} \text{T.C.} &= 1 * (n+1) \\ &\equiv O(n) \end{aligned}$$

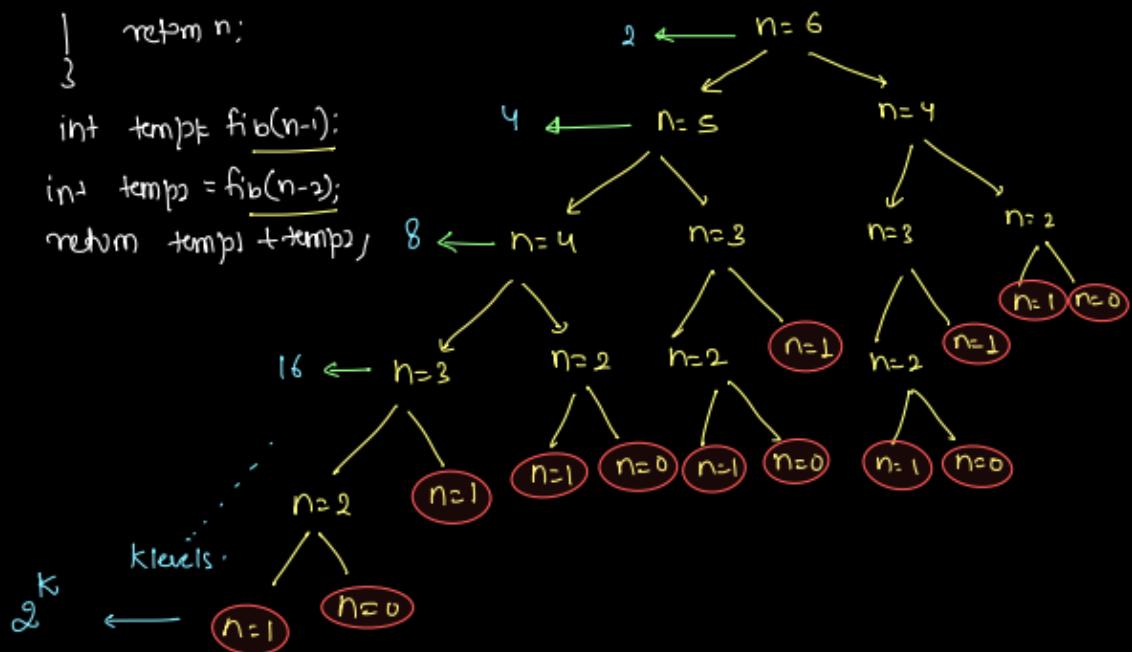


```

int Fib(int n) {
    if(n==0 || n==1)
        return n;
    int temp1 = fib(n-1);
    int temp2 = fib(n-2);
    return temp1 + temp2;
}

```

3



value of $k = ?$

value of n is equal to number of levels

$$k = n.$$

total number of call = $2^1 + 2^2 + 2^3 + \dots + 2^n$

Sum of $\underbrace{G.P}$ with $a=2, r=2, t=n$ \Rightarrow equal

Geometric Progression. $\text{Sum} = \frac{a(r^t - 1)}{r-1}$ \Rightarrow equal

$$\text{total call} = \frac{2(2^n - 1)}{2-1} = 2^{n+1} - 2 \approx 2^n$$

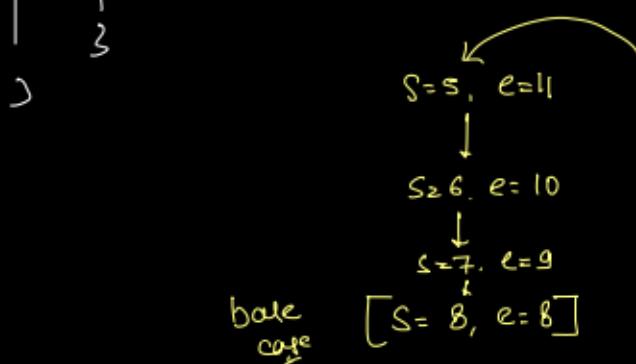
$$T.C. = 1 + 2^n$$

$$= O(2^n)$$

```

boolean isPalindrome(char[] A, int s, int e) {
    if (s == e || s > e) {
        return true;
    }
    if (A[s] != A[e]) {
        return false;
    } else {
        return isPalindrome(A, s+1, e-1);
    }
}

```



$$T.C = 1 * \frac{n}{2} = \frac{n}{2}$$

$$T.C = O(n)$$

Power [Linear Power]

```

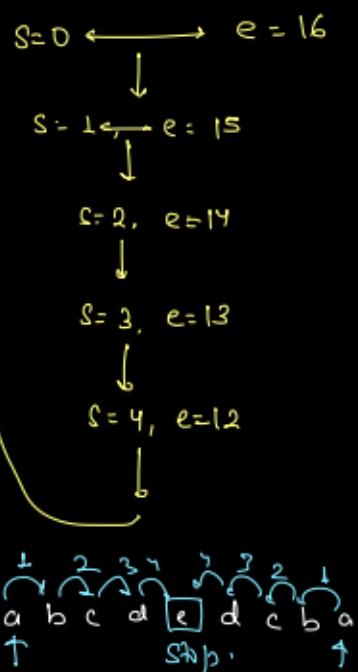
int pow(int a, int n) {
    if (n == 0) {
        return 1;
    }
    int temp = pow(a, n-1);
    return temp * a;
}

```

3, 5
↓
3, 4
↓
3, 3
↓
3, 2
↓
3, 1
↓
3, 0

T.C for single function = $O(1)$
No. of call = n calls

$$T.C = 1 + n = O(n)$$



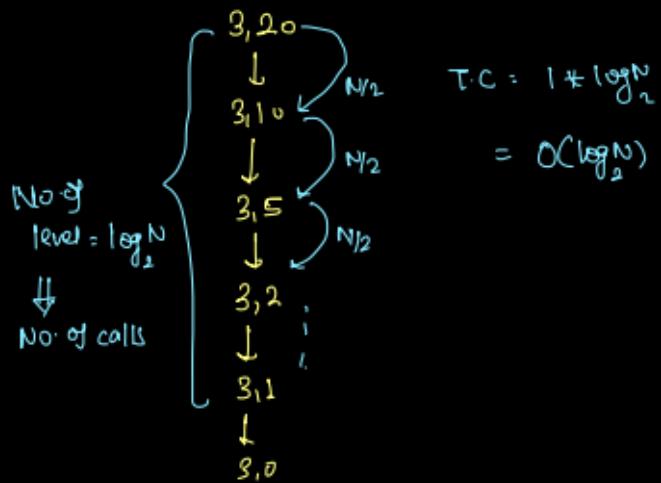
```

int pow(int a, int n) {
    if(n == 0) {
        return 1;
    }
    int temp = pow(a, n/2);
    if(n % 2 == 0) {
        return temp * temp;
    } else {
        return temp * temp * a;
    }
}

```

T.C. for single function = $O(1)$

$$\text{No. of calls} = \log_2 N$$



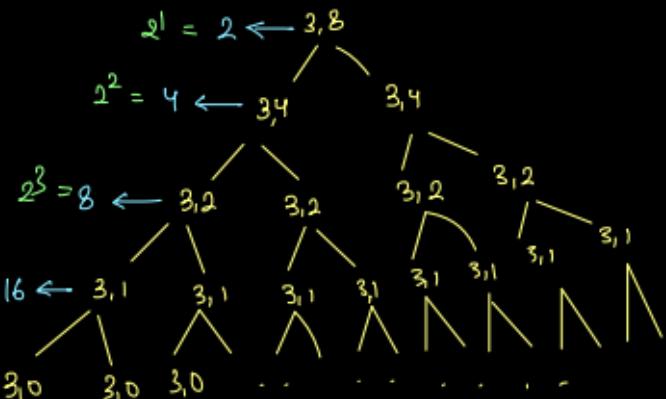
False Smart / Pseudo power:

T.C. for single function = $O(1)$
No. of calls ?

```

int pow(int a, int n) {
    if(n == 0) {
        return 1;
    }
    if(n % 2 == 0) {
        return pow(a, n/2) * pow(a, n/2);
    } else {
        return pow(a, n/2) * pow(a, n/2) * a;
    }
}

```



$$\text{Value of } K \rightarrow \text{No. of levels} \rightarrow \log N \Rightarrow K = \log N$$

$$\text{total call} = 2^1 + 2^2 + 2^3 + \dots + 2^K$$

$$= \frac{2(2^K - 1)}{2-1} = 2^{K+1} - 2 \quad \left[a^{\log_a b} = b \right]$$

$$\equiv 2^{\log_2 N}$$

$$\text{total call} = n$$

$$\left[a^{\log_a b} = b \right]$$

$$TC = O(4) * n$$

$$= O(n)$$

fake smart or pseudo power have same complexity as linear power.

doubt session:

$$\frac{2(2^k - 1)}{2-1} \rightarrow \text{Numerator}$$

$$= 2^k - 1 = ①$$

$$\begin{aligned} \text{Numerator} &\rightarrow 2(2^k - 1) & A(B - C) \\ &= 2 \cancel{*} 2^k - 2 & \Rightarrow A*B - A*C \\ &= 2 \cancel{*} 2^k - 2 & a^b \cancel{*} a^c = a^{b+c} \\ &= [2^{k+1} - 2] \end{aligned}$$

Problem1: Given n , ($n > 0$), find sum of all digits

<https://www.interviewbit.com/snippet/703833adb2b2f622fcec/>

```
import java.util.*;
// Create MinStack using stack
class Main{
public static int printNumSum(int n){
if(n == 0){
return 0;
}
int num = n % 10;
count = printNum(n/10);
return count + num;
}
public static void main(String args[]) {
```

```

// Problem1: Given n, (n>0), find sum of all digits
System.out.print(printNumSum(1104));
}
}

```

Problem2: Given 'a' and 'n'. calculate a^n ;

<https://www.interviewbit.com/snippet/13461d4eb767e69ee29b/>

```

import java.util.*;
// Create MinStack using stack
class Main{
public static int powerOfNum(int a, int n){
if(n == 0){
return 1;
}
int count = powerOfNum(a, n-1);
return a * count;
}
// this is optimized version
public static int powerOfNumOpt(int a, int n){
if(n == 0){
return 1;
}
int temp = powerOfNum(a, n/2);
if(n % 2 == 0){
return temp * temp;
}
return temp * temp * a;
}
public static void main(String args[]) {
// Problem2: Given a and n. calculate  $a^n$ 
int a = 2;
int n = 5;
System.out.print(powerOfNumOpt(a,n));
}
}

```

Problem3: Given o, n & p. calculate $a^n \% p$.

Notes -> DSA: Recursion 3 - 1 - Jun 2023

Tower of Hanoi

Given N plates and 3 tower (Source, Destination and Helper). Transfer all plates from 'src' to 'dest' and print instruction by keeping the following rule in mind :

- a. We can move only one plate at a time.
- b. Big plate can't be placed over a small plate.
- c. We can only move top-most plate from a tower.

The diagram shows three vertical rods. The leftmost rod, labeled 'source', has three blue elliptical plates stacked vertically, labeled 1 (top), 2, and 3 (bottom). A red curved arrow points from the source to the middle rod, labeled 'Destination'. The rightmost rod, labeled 'Helper', is empty. The text $n=3$ is written above the source rod.

The diagram illustrates three stages of the Hanoi move:

- Stage 1: The 'source [s]' tower has 3 plates (1, 2, 3) on it.
- Stage 2: The 'Destination [d]' tower has 2 plates (1, 2) on it, and the 'Helper [h]' tower has 1 plate (3) on it.
- Stage 3: The 'source [s]' tower is empty, the 'Destination [d]' tower has 3 plates (1, 2, 3) on it, and the 'Helper [h]' tower is empty.

Agenda 1: transfer first 3 plates from source to helper using destination.

Agenda 2: transfer 4th plate from source to destination.

Agenda 3: transfer all 3 plates from helper to dest using source

1. move 1 from s to h
2. move 2 from s to d
3. move 1 from h to d
4. move 3 from s to h

Assumption: $\text{toh}(n, \underline{s}, \underline{d}, \underline{h})$

↳ 'toh' function will print all the instruction to move all plates from source to destination using helper by following all rules.

Main logic:

① transfer $(n-1)$ plates from source to help using dest.

$\text{toh}(n-1, \underline{\text{source}}, \underline{\text{dest}}, \underline{\text{hlp}})$;

② transfer n^{th} plate from source to dest

SCALER ↗ ③ transfer $(n-1)$ plates from help to dest using source.

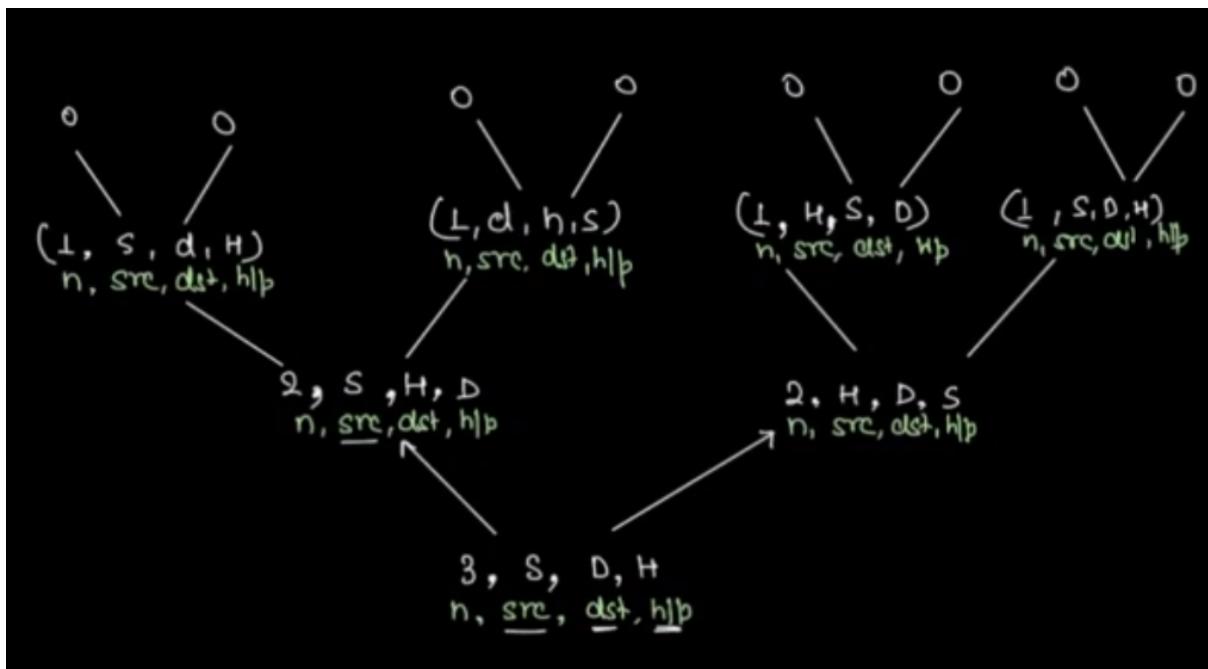
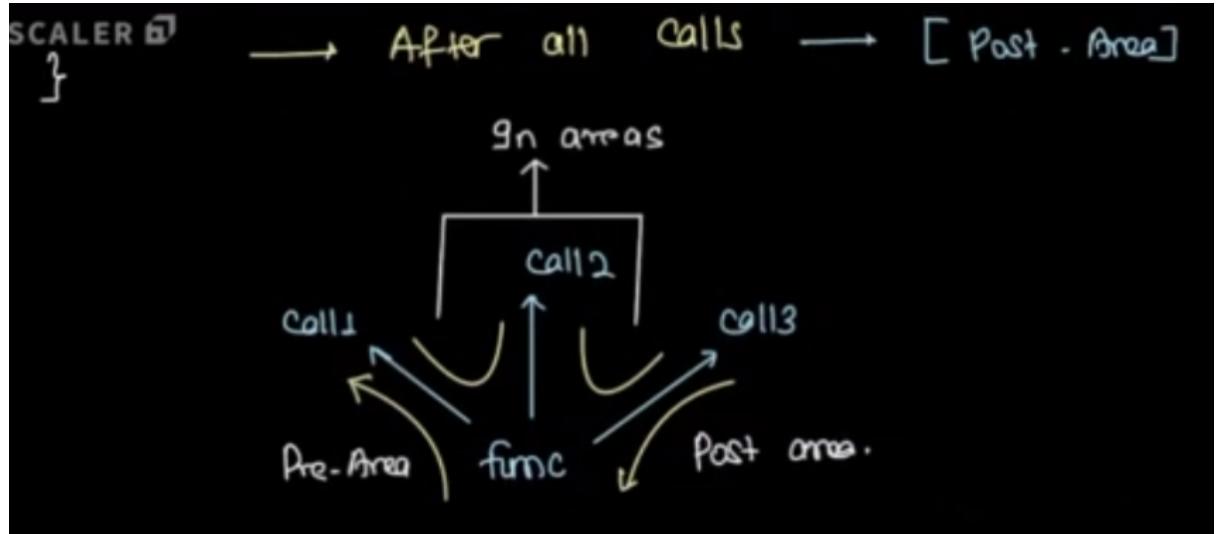
The image shows a Java code editor on the left and an output window on the right. The code editor contains the following Java code:

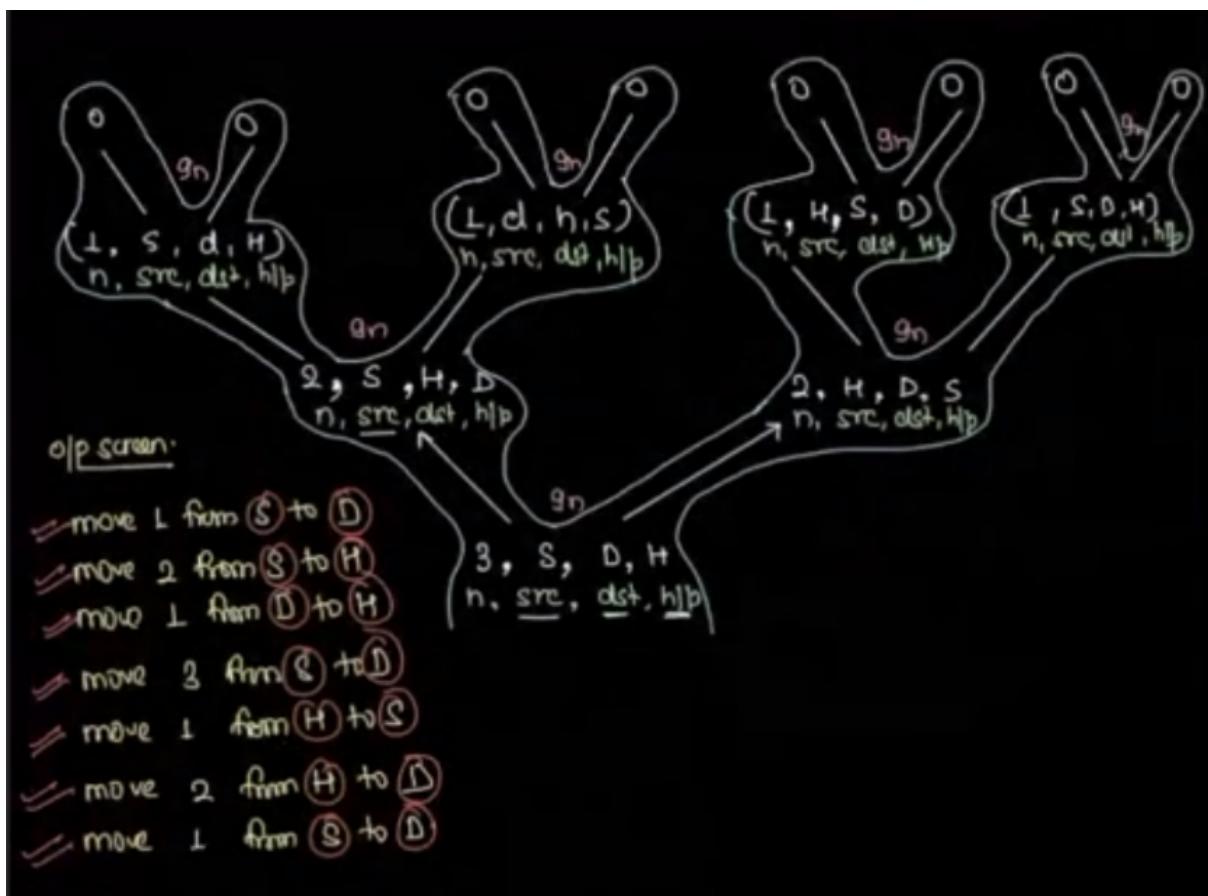
```
class Main {  
    public static void toh(int n, char src, char dst, char hlp) {  
        // smallest problem, if n = 0 -> don't do anything  
        // base case  
        if(n == 0) {  
            return;  
        }  
        // move n-1 plates from src to help using destination  
        toh(n-1, src, hlp, dst);  
        // move nth plate from src to dest  
        System.out.println("Move " + n + " plate from " + src + " to " + dest);  
        // move n-1 plates from hlp to dest using src  
        toh(n-1, hlp, dst, src);  
    }  
  
    public static void main(String args[]) {  
        // Your code goes here  
        int n = 3;  
        toh(n, 'S', 'D', 'H');  
    }  
}
```

The output window on the right shows the execution results:

Custom Input ⓘ

Output
[Success] Your code was executed successfully
Move 1 plate from S to D
Move 2 plate from S to H
Move 1 plate from D to H
Move 3 plate from S to D
Move 1 plate from H to S
Move 2 plate from H to D
Move 1 plate from S to D

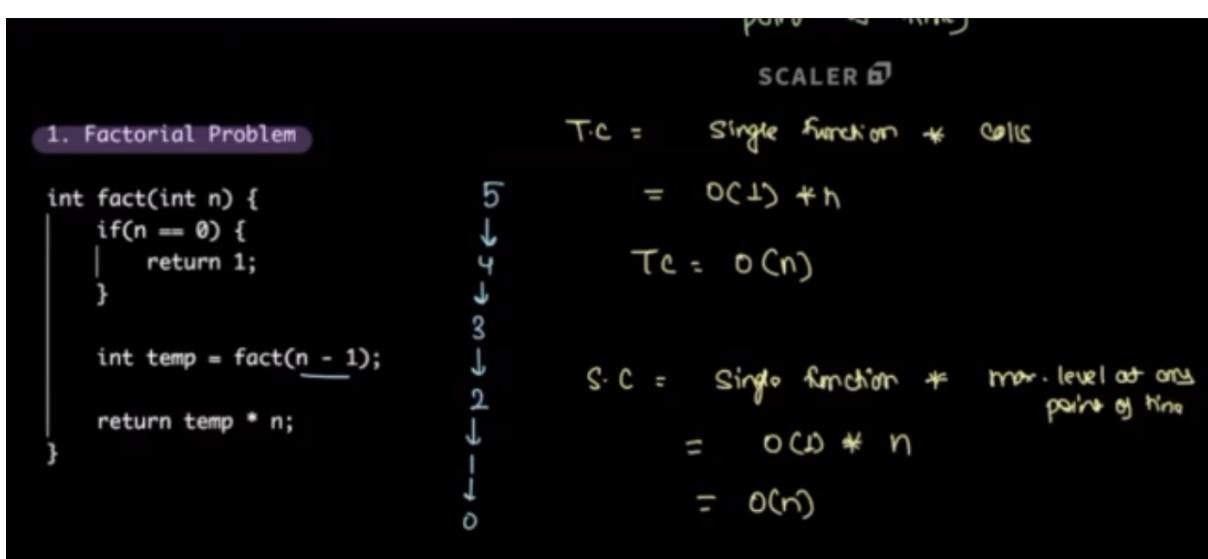




Space Complexity in recursion

T = Single function * No. of calls

S = Single function * Max level of any point of time



2. Fibonacci Problem

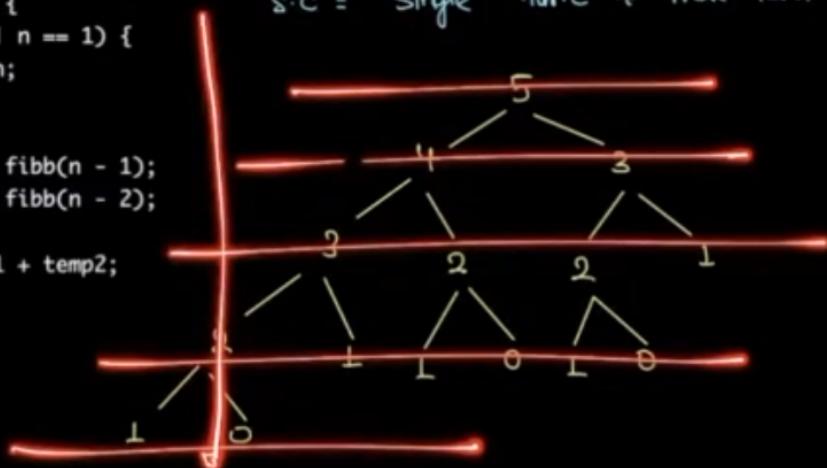
T.C.: $O(2^n)$

```
int fibb(int n) {
    if(n == 0 || n == 1) {
        return n;
    }

    int temp1 = fibb(n - 1);
    int temp2 = fibb(n - 2);

    return temp1 + temp2;
}
```

S.C = Single func at max level



2. Fibonacci Problem

T.C.: $O(2^n)$

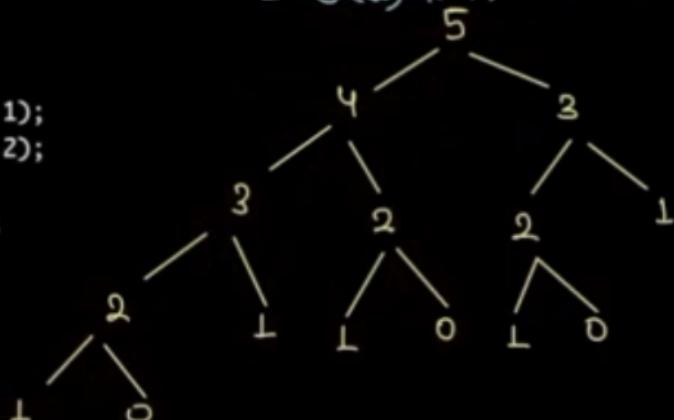
```
int fibb(int n) {
    if(n == 0 || n == 1) {
        return n;
    }

    int temp1 = fibb(n - 1);
    int temp2 = fibb(n - 2);

    return temp1 + temp2;
}
```

$$S.C = \text{Single func} \neq \text{max level}$$

$$= O(1) * n = O(n)$$

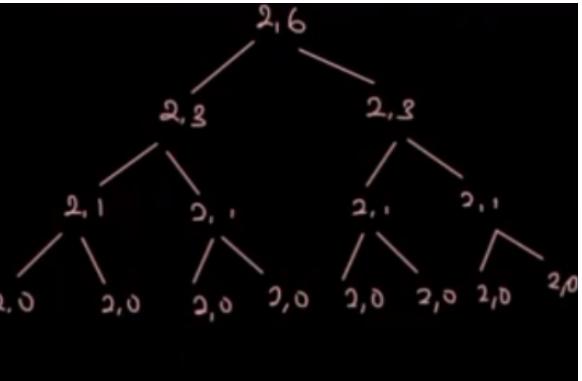


```

3. Power (Pseudo Better)
SCALER
int pow3(int a, int n) {
    if(n == 0) {
        return 1;
    }

    if(n % 2 == 0) {
        return pow3(a, n / 2) * pow3(a, n / 2);
    } else {
        return pow3(a, n / 2) * pow3(a, n / 2) * a;
    }
}

```



Time complexity = $O(n)$

$$\begin{aligned}
 \text{Space complexity} &= \text{single funcn} * \text{max level} \\
 &= O(1) * \log n \\
 &= O(\log n)
 \end{aligned}$$

Recursive OR reoccurrence relation

Recursive OR reoccurrence relation:

$$f(n) = 3n + 2$$

$$n \rightarrow T.C = T(n)$$

$$\begin{aligned}
 f(2) &= 3*2 + 2 \\
 &= 6 + 2 = 8
 \end{aligned}$$

1. Factorial Problem

```

int fact(int n) {
    if(n == 0) {
        return 1;
    }

    int temp = fact(n - 1);

    return temp * n;
}

```

$$(n-1)!$$

$$T.C = T(n-1)$$

$$\begin{aligned}
 f(5) &= 3*5 + 2 \\
 &= 15 + 2 = 17
 \end{aligned}$$

$$\begin{aligned}
 f(n-2) &= 3(n-2) + 2 \\
 &= 3n - 6 + 2 \\
 &= \boxed{3n - 4}
 \end{aligned}$$

2. Fibonacci Problem

```
int fibb(int n) {
    if(n == 0 || n == 1) {
        return n;
    }

    int temp1 = fibb(n - 1);
    int temp2 = fibb(n - 2);

    return temp1 + temp2;
}
```

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

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

n is change two times in RHS

we can't solve it simply.

→ we have to make some eq. to relate T.C.

$$T'(n) = T(n-1) + T(n-2) + k$$

$$T'(n) = 2T(n-1) + k$$

$$T''(n) = T(n-2) + T(n-3) + k$$

$$T''(n) = 2T(n-2) + k$$

$$T''(n) < T(n) < T'(n) \rightarrow \text{Relation}$$

DSA: Sorting 1 - 6 - June 2023

- What is sorting
- Bubble Sort
- Insertion Sort
- Merge Two Sorted Array
- Merge Sort

What is sorting → Arranging data to some specific order is sorting. Arrays.sort(arr), Collections.sort(ArrayList) this is the in build function which has complexity O(NlogN).

Bubble Sort ->

In this method, we need to find the maximum element and place it at the end of the array. This process is called a "sorted region." In each step, we compare the current element with the next element and swap them if the current element is greater. We repeat this process for every element using two nested loops.

- First loop will run 0 to $i < n - 1$ // we are assuming first one is already sorted if we reach end
- Second loop run 0 to $j < n - 1 - i$

The time complexity of this method is $O(n^2)$, where n is the number of elements in the array. In the best case scenario, when the array is already sorted, the time complexity is $O(n)$ as there are no swaps needed.

Please note that this method is not the most efficient for sorting large arrays, but it can be useful for smaller arrays or educational purposes.

L. Bubble Sort:

3 8 6 2 4
0 1 2 3 4

After gtr 1: 3 6 2 4 8 *Sorted Region*

After gtr 2: 3 2 4 6 8

*If n elements is there
(n-1) gtr is required.*

After gtr 3: 3 2 4 6 8

*Steps!
iterate and swap max.
data in end and make
end part sorted.*

After gtr 4: 2 3 4 6 8

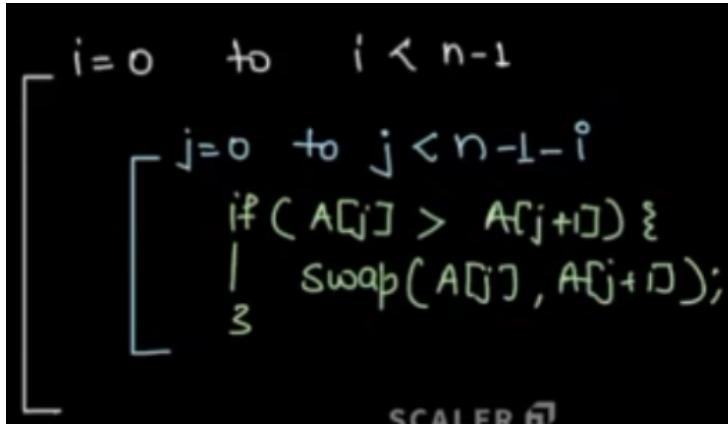
3 8 6 2 4
0 1 2 3 4

gtr 1: 3 ~~6~~ ~~2~~ ~~8~~ 8
 ↓ ↓ ↓ ↓
 j j+1

gtr 2: 3 ~~2~~ ~~4~~ 6 8
 ↑ ↑
 j j+1

*if(A[j] > A[j+1]) {
| swap(A[j], A[j+1]);
}*

gtr 2: ~~2~~ ~~3~~ 4 6 8
 ↑ ↑
 j j+1



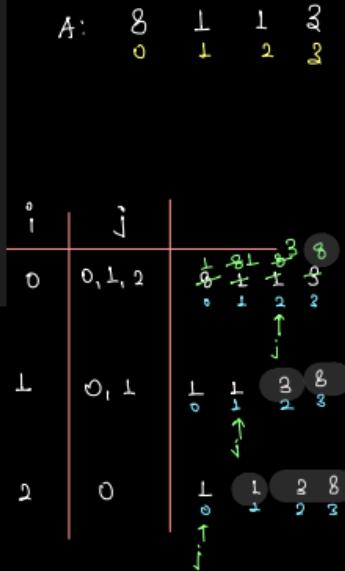
```

// bubble sort algorithm
public static void bubbleSort(int[] A) {
    int n = A.length;
    for(int i = 0; i < n - 1; i++) {
        for(int j = 0; j < n - 1 - i; j++) {
            if(A[j] > A[j + 1]) {
                // swap the data of A[j] and A[j+1]
                int temp = A[j];
                A[j] = A[j + 1];
                A[j + 1] = temp;
            }
        }
    }
}

```

T.C.: $O(n^2)$

S.C.: $O(1)$



Bubble Sort:

Best Case T.C. \Rightarrow Bubble sort not depend on order of data.

But if no. of swapping is 0

that means data is already sorted.

→ If data is already sorted stop the algorithm.

```

// bubble sort algorithm
public static void bubbleSort(int[] A) {
    int n = A.length;
    for(int i = 0; i < n - 1; i++) {
        boolean swapping = false;
        for(int j = 0; j < n - 1 - i; j++) {
            if(A[j] > A[j + 1]) {
                // swap the data of A[j] and A[j+1]
                int temp = A[j];
                A[j] = A[j + 1];
                A[j + 1] = temp;
                // mark swapping as true for
                // identification of step
                swapping = true;
            }
        }
        if(swapping == false) {
            return;
        }
    }
}

```

Best case: \rightarrow data is sorted

T.C. $\rightarrow O(n)$

Worst case \rightarrow data is not sorted

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

S.C. $\rightarrow O(1)$

8:19 - 8:30 \rightarrow Break Time

Insertion Sort -> In this algorithm, we start by assuming that the first element is already in its sorted position. We then iterate through the remaining elements, comparing each element with the previous one. If the previous element is greater, we swap them. This process continues until we reach the end of the array or until no more swaps are needed.

To implement this, we use two loops. The first loop starts from the beginning of the array and goes up to the second-to-last element ($n - 1$), as we assume the first element is already in place. The second loop starts from the element next to the current element ($i + 1$) and continues as long as j is greater than 0. Within the second loop, we compare the current element with its previous element, and if a swap is necessary, we perform it.

The algorithm breaks the inner loop if no swap is made in an iteration, indicating that the array is already sorted. This optimization helps improve the efficiency of the algorithm for partially sorted or already sorted arrays.

- First loop

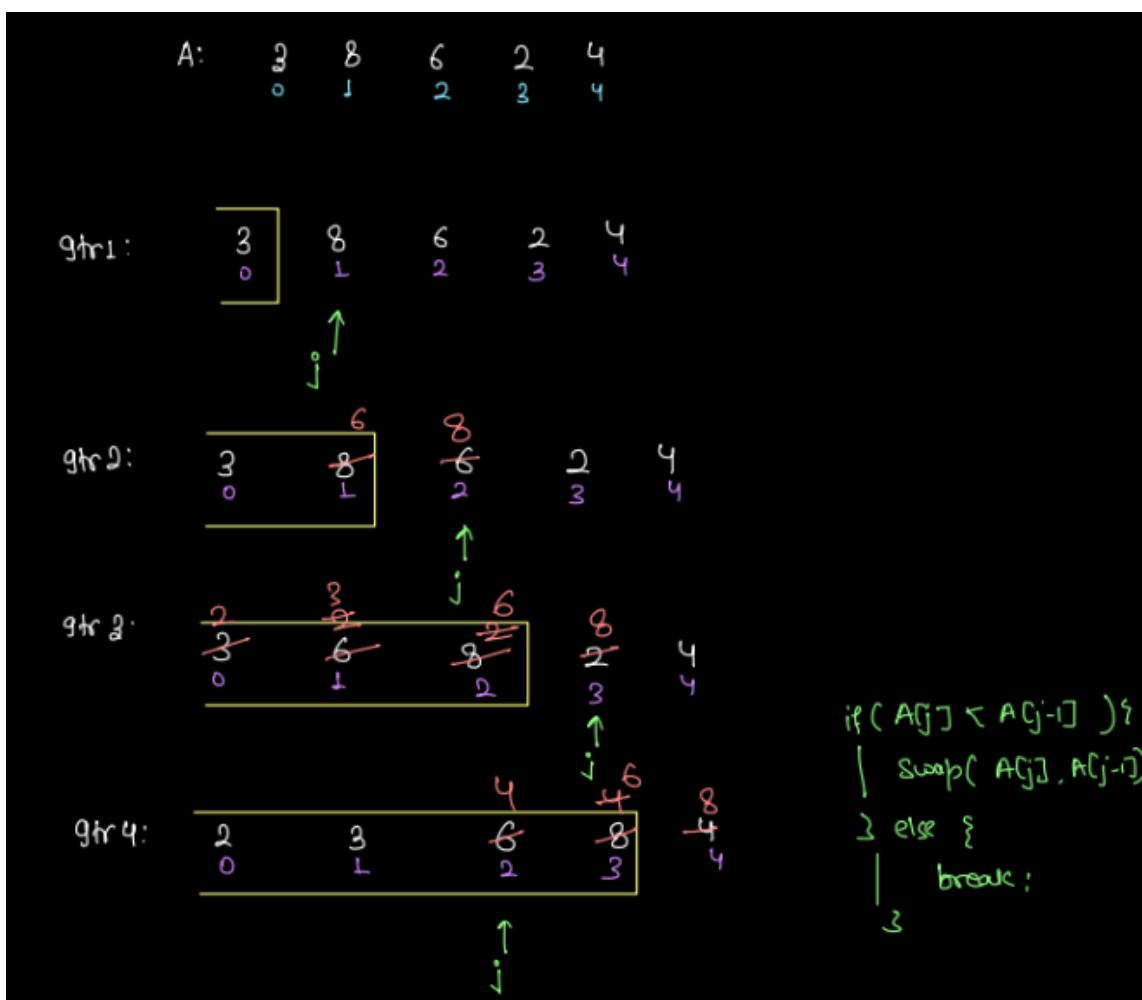
$i = 0; i < n - 1; i++$

- Second loop

$j = i + 1; j > 0; j -$

This algorythem called insertion sort

It has $TC = n^2$



```

    i = 0      to      i < n-1
    [
        j = i+1  to  j > 0
        [
            if ( A[j] < A[j-1] )
                {
                    swap( A[j] , A[j-1] );
                    }
            else
                break;
        ]
    ]
}

```

```

// insertion sort
public static void insertionSort(int[] A) {
    int n = A.length;
    for(int i = 0; i < n - 1; i++) {
        for(int j = i+1; j > 0; j--) {
            if(A[j] < A[j-1]) {
                // swap curr i.e. A[j] with prev. i.e. A[j-1]
                int temp = A[j];
                A[j] = A[j - 1];
                A[j - 1] = temp;
            } else {
                // if curr is greater than prev.
                // that means curr ele is already placed at
                // correct position
                break;
            }
        }
    }
}

T.C: O(n^2)
S.C: O(1)

```

i	j	state of array
0	$1 \rightarrow 0$	$\frac{2}{\cancel{1}} \frac{7}{2} 4 6$
1	$2 \rightarrow 1$	$2 \frac{4}{\cancel{2}} \frac{7}{4} 6$
2	$3 \rightarrow 2$	$2 4 \frac{6}{\cancel{7}} 6$

$$\text{Eq. } \quad \begin{array}{r} 10 \\[-1ex] \times 20 \end{array} \quad \begin{array}{r} 20 \\ 30 \\ 40 \\ 50 \\ \hline \text{final answer} \end{array} \quad \begin{array}{r} 2467 \\ \hline \text{8000} \end{array}$$

i	j	state of array
0	$1 \rightarrow break$	[10] 20 30 40 50 ↑
1	$2 \rightarrow break$	[10] 20 [30] 40 50 ↓ ↑
2	$3 \rightarrow break$	[10] 20 [30] 40 50 ↓ ↑
3	$4 \rightarrow break$	[10] 20 30 [40] 50 ↓ ↑

T.C: $O(n)$ \rightarrow Best case in insertion sort i.e. \rightarrow array is already sorted.

Merge Two Sorted Array

Problem 1: Given two sorted arrays A and B, Merge them.

to make an array which is overall sorted.

Brute force

A: 2 5 9 12 15 → n

① add all elements in

answer array

→ O(n+m)

B: 3 6 8 10 16 18 → m

② sort answer array

Size n+m

Time =
 $\frac{(n+m) \log(n+m)}{n+m}$
 $\approx \log(n+m)$

Ans:

2	3	5	6	8	9	10	12	15	16	18
0	1	2	3	4	5	6	7	8	9	10

Allowed T.C.

O(n+m)

↓
i

A: 2 5 9 12 15

B: 3 6 8 10 16 18

↓ ↓

2	3	5	6	8	9	10	12	15	16	18
0	1	2	3	4	5	6	7	8	9	10

↑
k

A: ~~2~~⁰ ~~5~~¹ ~~9~~² ~~12~~³ ~~15~~⁴

↓ ↓ ↓ ↓ ↓

B: ~~3~~⁰ ~~6~~¹ ~~8~~² ~~10~~³ ~~16~~⁴ ~~18~~⁵

↓ ↓ ↓ ↓ ↓

res[k] = A[i]

res[k] = B[j]

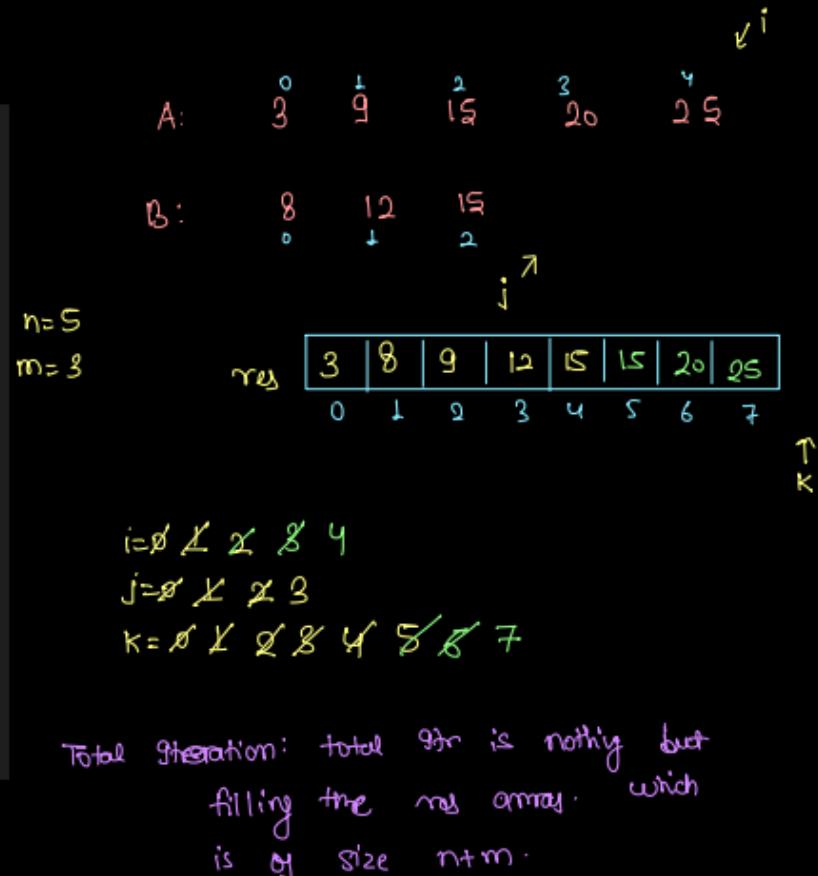
3	8	9	12	15	15	20	25
0	1	2	3	4	5	6	7

k

```

i=0, j=0, k=0
while(i < n && j < m) {
    if(A[i] < B[j]) {
        res[k] = A[i];
        i++;
    } else {
        res[k] = B[j];
        j++;
    }
    k++;
}
// if i is left
while(i < n) {
    res[k] = A[i];
    i++;
    k++;
}
// if j is left
while(j < m) {
    res[k] = B[j];
    j++;
    k++;
}
return res;

```



T.C. : O(n+m)

S.C: O(1) → Because we are not creating result array for our size instead question is asking to return an array.

Merge Sort -> Divide and conquer algorithm (TC: NlogN)

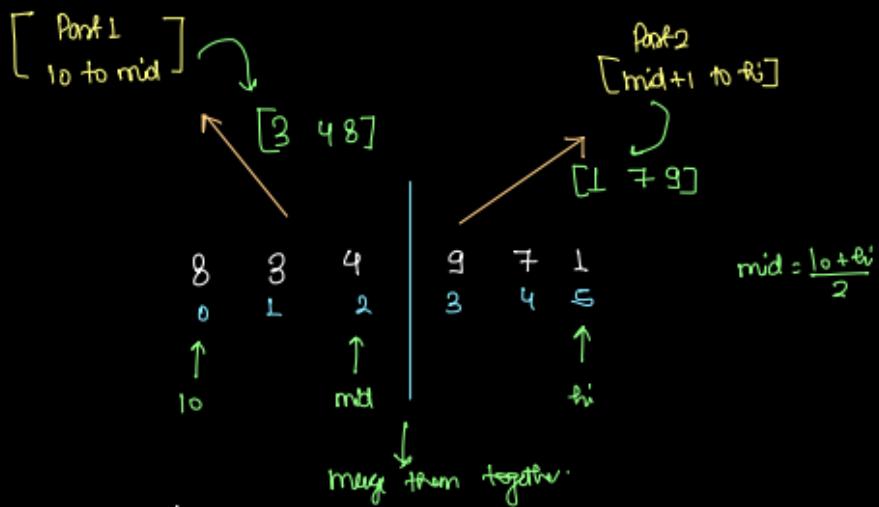
This algorithm based on recursion. In this algorithm We will get the mide point using $lo+hi/2$ like if array has 1,2,3,4,5,6 then lo is 0 index and hi is 5th index and mid point is 2. now we will make it two half of array and then we will merge both the array till we will not reach to base case where $lo == hi$.

Merge Sort:

T.C. $O(n \log n)$

divide and conquer algorithm

→ this algorithm is based on
Recursion.



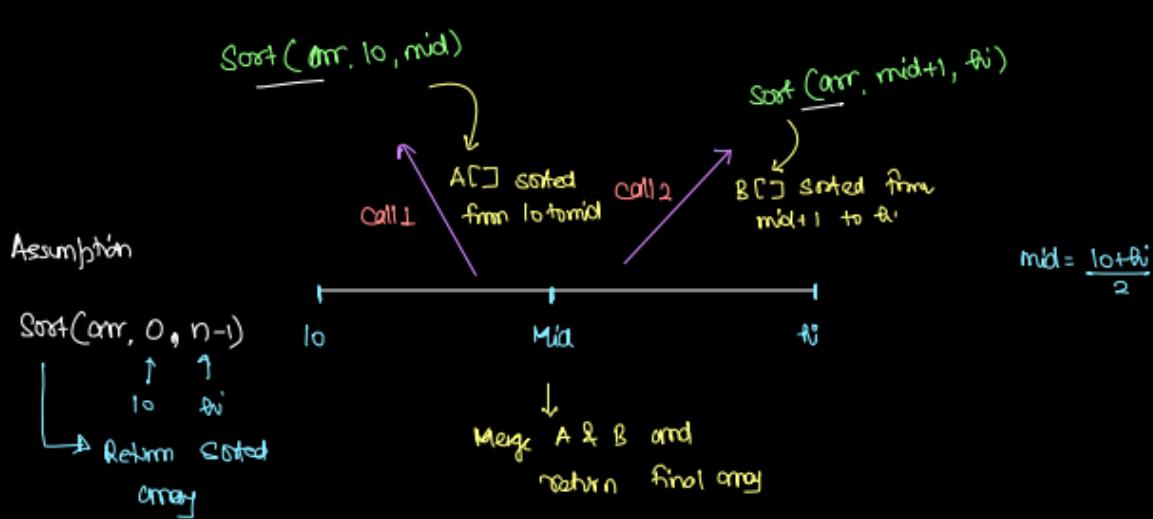
Assumption → arr, lo, hi
Sort array from lo to hi

$[1, 3, 4, 7, 8, 9]$

Main logic: → $\text{sort}(\text{arr}, \text{lo}, \text{mid}) \rightarrow A$

$\text{sort}(\text{arr}, \text{mid}+1, \text{hi}) \rightarrow B$

My task → Merge them together.

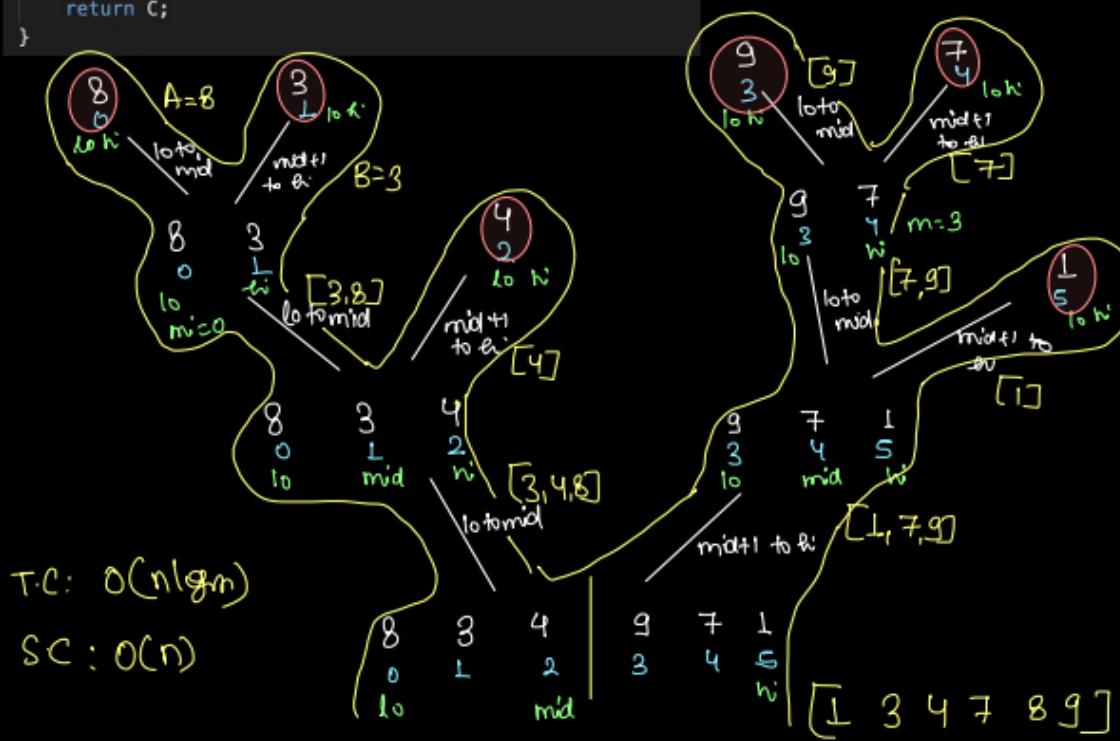


```
public static int[] mergeSort(int[] arr, int lo, int hi) {
    // base case
    if (lo == hi) {
        int[] bres = new int[1];
        bres[0] = arr[lo];
        return bres;
    }
    int mid = (lo + hi) / 2;
    int[] A = mergeSort(arr, lo, mid);
    int[] B = mergeSort(arr, mid + 1, hi);
    int[] C = mergeTwoSortedArray(A, B);
    return C;
}
```

Recurrence Relation

$$T(n) = T(n/2) + T(n/2) + n$$

$\downarrow \quad \downarrow$
1st call 2nd call merge
two sorted array



DSA: Sorting 2 - 8 - June 2023

- Inversion Count
- Custom Comparison

- Factor Sort
- Sort Coordinates
- Largest Number

Inversion Count -> if $A[i] > A[j]$ then all pairs with $A[i]$ called inversions like below. It will solved using recursion shorting function like merge short where we will divid array in 2 parts and then call same function using $\text{mergeSort}(A, \text{lo}, \text{mid})$, $\text{merge Sort}(A, \text{mid}+1, \text{hi})$ and mid will calculate $n+m/2$

Problem 1 Inversion Count

Given an array of integers A . If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an inversion of A . Find total number of inversions in A .

NOTE: Return count % $(10^9 + 7)$

Eg1. $A[] = \begin{bmatrix} 2 & 3 & 0 & 1 \\ 0 & 1 & 2 & 3 \end{bmatrix}$

$$\begin{array}{ll} A[i], A[j] & A[i], A[j] \\ [2, 0] & [2, 1] \\ [3, 0] & [3, 1] \end{array}$$

Inversion count = 4 Ans

Eg2: $A[] = \begin{bmatrix} 8 & 5 & 3 & 4 & 1 & 6 & 2 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}$

$$\begin{array}{lllll} A[i], A[j] & & & & \\ (8, 5) & (5, 3) & (3, 1) & (4, 1) & (6, 2) \\ (8, 3) & (5, 4) & (3, 2) & (4, 2) & \\ (8, 4) & (5, 1) & & & \\ (8, 1) & (5, 2) & & & \\ (8, 6) & & & & \text{Inversion Count} = 15 \text{ Ans} \\ (8, 2) & & & & \end{array}$$

Brute force idea:

for value of $A[i]$, we will count value of $A[j]$ in which $A[i] > A[j]$ for $j = i+1$ to $n-1$.

```
int inversionCount(int[] A) {
    int n = A.length;
    long count = 0;
    for(int i=0; i<n; i++) {
        for(int j=i+1; j<n; j++) {
            if(A[i] > A[j]) {
                count = (count + 1)%mod
            }
        }
    }
    return (int)(count);
}
```

T.C: $O(n^2)$

S.C: $O(1)$

Time complexity

$$\begin{array}{lll} i=0, j=1 \text{ to } n-1 & \longrightarrow & n-1 \\ i=1, j=2 \text{ to } n-1 & \longrightarrow & n-2 \\ i=2, j=3 \text{ to } n-1 & \longrightarrow & n-3 \\ \vdots & & \vdots \\ i=n-2, j=n-1 \text{ to } n-1 & \longrightarrow & 1 \end{array}$$

Total time: $1+2+3+\dots+n-2+n-1$

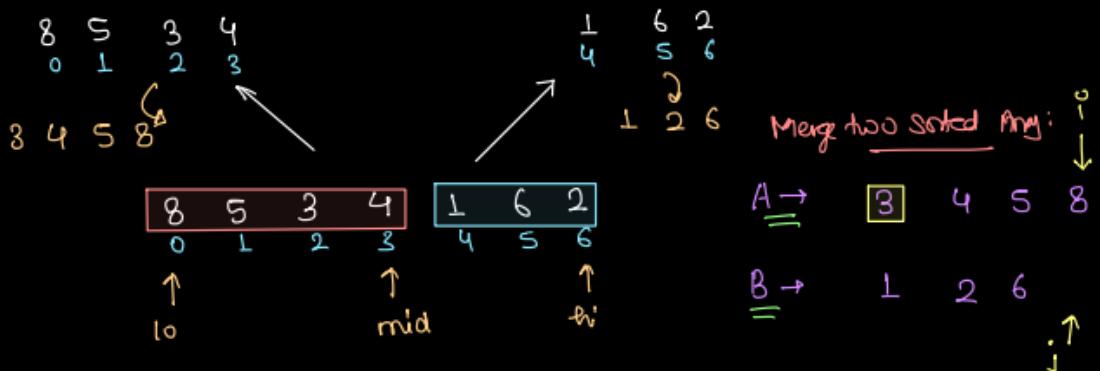
$$= \frac{n(n-1)}{2}$$

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

Expected T.C: $O(n \log n)$

$$A[] = \boxed{8 | 5 | 3 | 4 | 1 | 6 | 2}$$

→ sort array (using Merge Sort)



$$\text{mid} = \frac{l_0 + h_1}{2}$$

$$= \frac{0+6}{2} = 3$$

```

if (A[i] > B[j]) {
    ans[k] = B[j];
    j++;
    [Count inversion pair]
}

```

3 else {

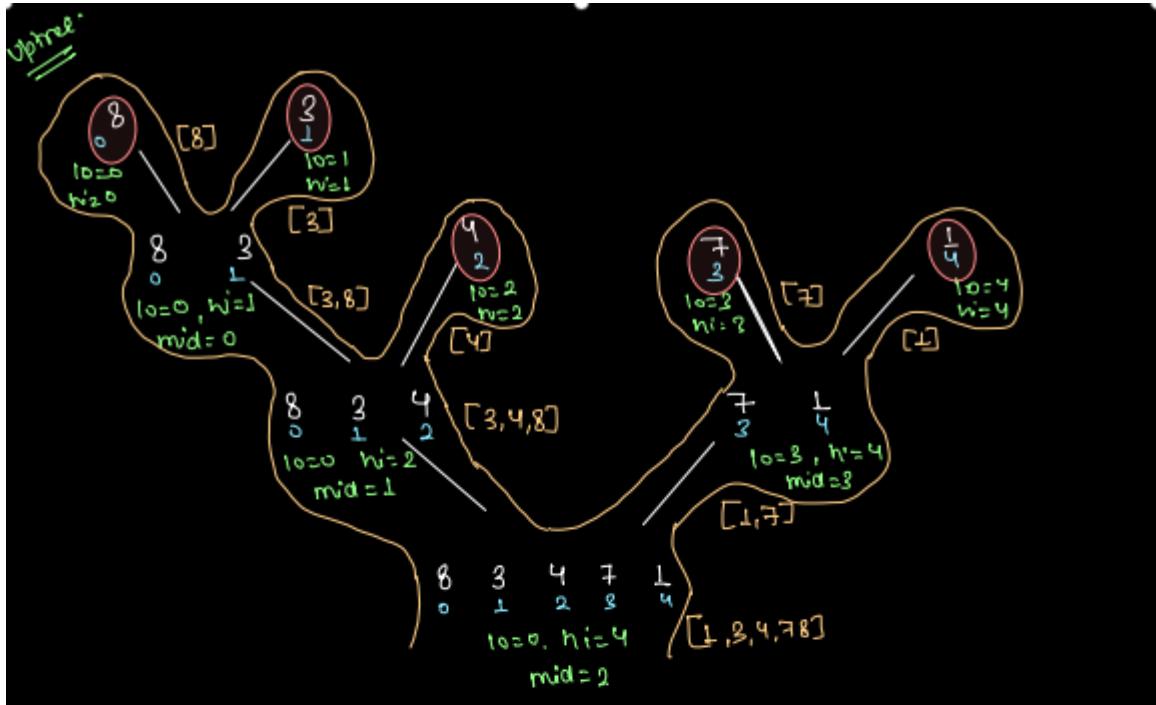
```

    ans[k] = A[i];
    i++;
}

```

3
k++:

3,1 4,1 5,1 8,1
3,2 4,2 5,2 8,2
8,6



$A[] \rightarrow 3 \ 4 \ 8$

$B[] \rightarrow 1 \ 7$

\uparrow

$C[] \rightarrow 1, 3, 4, 7, 8$

$A[i] > B[j]$

Count += (n-i)

Count = 0 + 1 + 1 + 1 + 3 + 1

= 7 Ans

(8, 3), (8, 4), (7, 1)

(3, 1) (4, 1) (7, 1)

(8, 7)

We are just adding one
single line in merge two sorted array

8:32 - 8:45
Break time

→ Rest Every code is same

T.C. : $O(n \log n)$

S.C. : $O(n)$

Custom Comparison -> We will write custom short function for custom comparison like if we want to sort car by price or by feature then we have to write it because default sort feature only provide chat, int sorting. for custom sorting function we will use sort method and pass second argument as comparator function like below.

We will use Integer class reference type not primitive type int and array also we have to convert in int to Integer. We will use @Override it is optional. If we want to return first parameter

```
Arrays.sort(arr, new Comparator<Integer>() {
    @Override
    public int compare(Integer a, Integer b) {
        // Possibility 1 : want to keep a before b
        // return negative value
        // Possibility 2 : want to keep b before a
        // return positive value
        // Possibility 3 : Doesnot matter
        // return 0

        // noraml Order
        /*
        if(a < b) {
            // want to keep a before b
            return -1;
        } else if(a > b) {
            // want to keep b before a
            return 1;
        } else {
            // Doesnot matter
            return 0;
        }
        */
        // reverse the order
        if(a < b) {
            // want to keep b before a
            return 1;
        } else if(a > b) {
            // want to keep a before b
            return -1;
        } else {
            // Doesnot matter
            return 0;
        }
    }
}
```

```

// sort only use this instead above
return a - b;
// for order reversal
// return b - a;
}
});

```

Factor Sort

Problem 2: Factor Sort

Given an array A. Sort the array in increasing order of number of distinct factors.

- * Least number of factor will come first
- * If two numbers have same factor the less value will come first.

Eg → [4 ↓ 7 ↓ 6 ↓ 9 ↓ 8 ↓ 2 ↓ 10 ↓]
 ↓ ↓ ↓ ↓ ↓ ↓ ↓
 3 2 4 3 4 2 4
 factor count

Sort on the basis of number of factors

Sorted: 2 7 4 9 6 8 10
 array

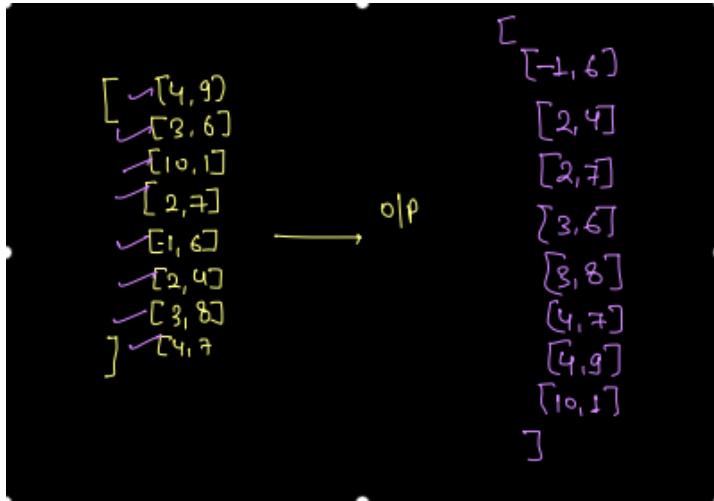
on basis of count of factors.

Problem: Sort coordinates:

Hint: { * first decide on the basis of x-coordinate.
 { * If x-coordinate is same for two different coordinates, then decide on the basis of y.

→ [4, 9] → val1
 → [3, 6] → val2
 → [10, 1] → val2
 → [2, 7] → val1 → Integer[][]
 → [1, 6] → two values val1 → Integer[] a
 → [2, 4] val2 → Integer[] b
 → [3, 8]
] [4, 7]

TODO
write logic



```

class Main {
/* Problem : factor Sort */
public static int factorCount(int val) {
// todo : write optimise code
int fcount = 0;
for(int i = 1; i <= val; i++) {
if(val % i == 0) {
fcount++;
}
return fcount;
}
public static int[] factorSort(int[] arr) {
int n = arr.length;
// convert int array into Integer array
Integer[] A = new Integer[n];
for(int i = 0; i < n; i++) {
A[i] = arr[i];
}
Arrays.sort(A, new Comparator<Integer>() {
@Override
public int compare(Integer a, Integer b) {
int fa = factorCount(a);
int fb = factorCount(b);
if(fa < fb) {
// a come first
return -1;
} else if(fa > fb) {
// b come first
return 1;
} else {
}
}
}

```

```

// if both are equal
// decide according to value of a and b
return a - b;
}
};

// fill arr from A
for(int i = 0; i < n; i++) {
arr[i] = A[i];
}
return arr;
}

public static void main(String args[]) {
// Your code goes here
int[] arr = {4, 7, 6, 9, 8, 2, 10};
System.out.println(Arrays.toString(arr));
arr = factorSort(arr);
System.out.println(Arrays.toString(arr));
}
}

```

Largest Number

Problem : Largest Number

Given an array of non-negative integers, arrange them such that they form the longest number.

NOTE: Result may long so return String.

→ Next Session

doubt session:

Sort →

val 1
val 2

int res = compare(val1, val2);
+ve ⇒ val 2 is greater.

-ve ⇒ val 1 is smaller &
will come first

o

[gbuilt sort → pre defined function]

Comparison in
manual sort if (arr[i] < arr[j]) {
function []
}

comparison
→ Comparable
pass lambda
function

→ Comparable
it allows to
modify user
defined class.

DSA: Sorting 3 - 10 - June 2023

- Sort 0 1
- Partition of Array
- Quick Sort Algorithm
- TC and SC of quickSort
- Worst case of quickSort
- Largest Number
- Sort Coordinates

Sort 0 1 → sort all 0's first then 1's after. We can use any sorting algorithm but we have to sort this in $O(n)$. we will use Aproch swap in single loop. We will define 2 initial variable i and j to 0 and then we will apply condition if $j == 1$ then $j++$ go to next element and if $j != 1$ then we will swap in same array j to i and i to j.

~~~~~

**Sort 0 1**

~~~~~

You are given an array of 0s and 1s in random order. Segregate 0s on left side and 1s on right side of the array [Basically you have to sort the array].

array A → 0 1 0 0 1 1 0 1 0
0|p ! 0 0 0 0 0 1 1 1 1

Idea 1: Sort the entire array

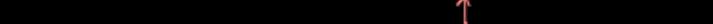
1. Bubble Sort → $O(n^2)$
2. Insertion Sort → $O(n^2)$
3. Merge Sort → $O(n \log n)$
4. Inbuilt sort → $O(n \log n)$

Expected T.C → $O(n)$

Idea 2: With the help of count strategy.

Step 1: Count number of 0s
Step 2: Count number of 1s] → First Iteration
Step 3 → Set 0's first & then 1's] → Second Iteration.

T.C: $O(n)$, Generate 2 times the array.

array → 

$i \rightarrow$ First index of L
 $j \rightarrow$ First index of unsorted part

```

if( arr[j] == 1) {
    j++;
}
else {
    swap(arr[i], arr[j]);
    i++;
    j++;
}

```

وَالْمُؤْمِنُونَ

```

int i=0;
int j=0;
while(j < arr.length) {
    if(arr[j] == 1) {
        j++;
    } else {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
        i++;
    }
}
return arr;

```

T.C. $O(n)$
in single generation

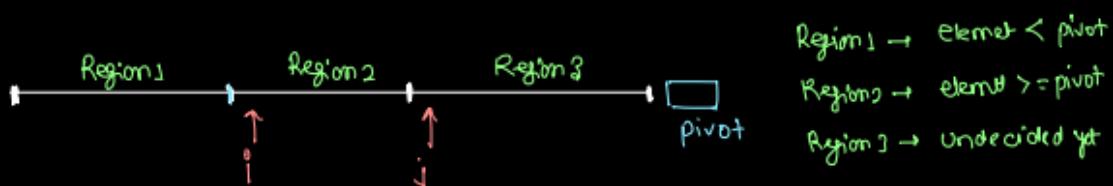
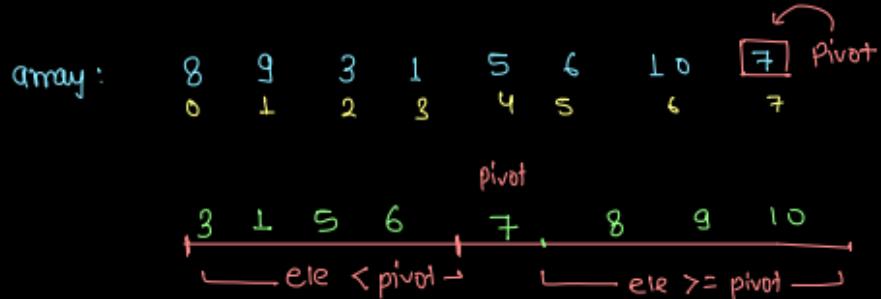
-1000 Soft 0 ± 2

A horizontal line representing a binary sequence. The first segment is labeled "0's", the second segment is labeled "1's", and the third segment is labeled "2's". A bracket below the "1's" and "2's" segments is labeled "Unsorted Region".

Partition of Array

Partition An Array

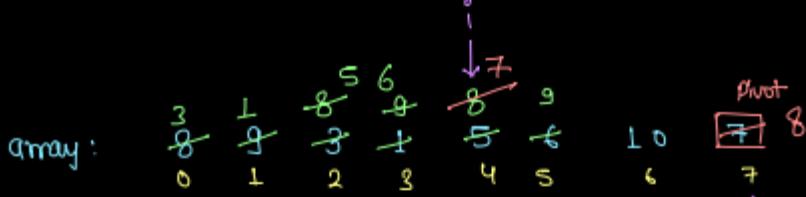
Given an array, partition it based on last element(Pivot Element), Such that all the elements < pivot are coming on left of it and all elements \geq pivot are coming on right of it.



$i \rightarrow$ first index
of Region 2

$j \rightarrow$ first index
of Region 3

$\text{arr}[j] < \text{pivot}$	$\text{arr}[j] \geq \text{pivot}$
Swap ($\text{arr}[i], \text{arr}[j]$)	// increase Region 2
$i++$	$j++$
$j++$	



final array $\underbrace{3}_{\text{ele} < \text{pivot}} \underbrace{1}_{\text{pivot}} \underbrace{5}_{\text{ele} > \text{pivot}} \underbrace{6}_{\text{pivot}} \underbrace{8}_{\text{pivot}}$

NOTE: for last element, move it in middle of partition

when $j = n-1$ (i.e. index of pivot)
swap arr[i], arr[j]

pivot = arr[n-1]

```

if (arr[j] >= pivot) {
    j++;
}
else {
    swap(arr[i], arr[j]);
    i++;
    j++;
}

```

int partitionIndex (int[] arr) {

```

int i=0;
int j=0;
int pivot = arr[arr.length-1];
while(j < arr.length-1) {
    if( arr[j] >= pivot) {
        j++;
    }
}

```

```

else {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
    i++;
    j++;
}

```

// j is pointing to last elem (i.e. pivot)

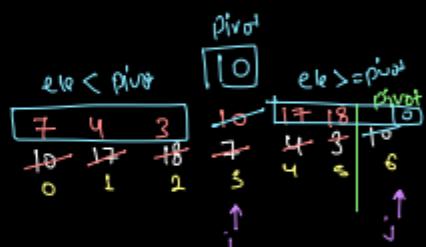
```

int temp = arr[i];
arr[i] = arr[j];
arr[j] = temp;

```

return i; → partition arr

3



$i=0$
 $j=0 \neq 2$
 $pivot=0$

Quick Sort Algorithm

Quick Sort Algorithm:

↳ Divide and Conquer Sorting Algorithm
↳ Recursion

Steps: → ① array from lo to hi index

② assume last element is pivot element

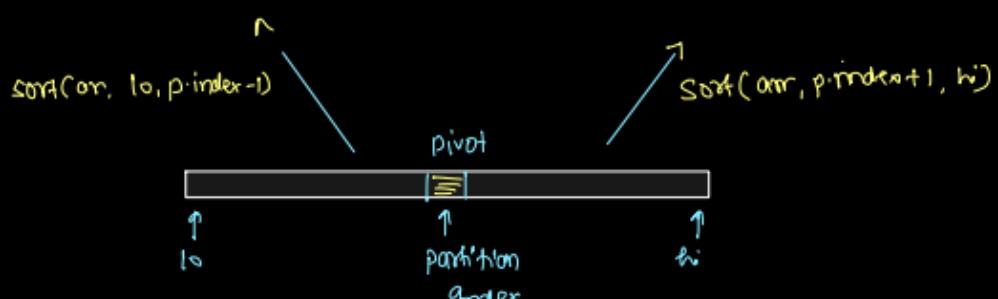
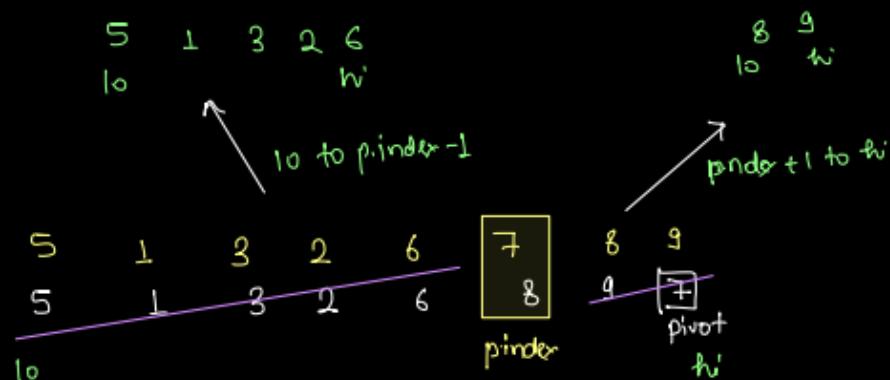
find position smaller

(After finding $p.i$, we are sure that $p.i$ is correctly placed at its position)

③ Ask from recursion to sort

$\text{sort}(\text{arr}, lo \text{ to } p.\text{index}-1)$

$\text{sort}(\text{arr}, p.\text{index}+1 \text{ to } hi)$



Assume function that mechanism works

```
void quickSort(int arr[], int lo, int hi) {
```

```
    if (lo >= hi) {
        return;
    }
```

```
    int pi = partitionIndex(arr, lo, hi);
    quickSort(arr, lo, pi - 1);
    quickSort(arr, pi + 1, hi);
```

3

$lo = hi$

$\frac{1}{0}$
 $lo \nearrow hi$

$\perp \quad \boxed{2} \quad 5 \quad 3 \quad 4$
5 3 \perp 4 ②
0 ① 2 3 1
 $lo \nearrow hi$

$lo = hi$
 $3 \quad 2 \quad \boxed{4} \quad 5$
5 3 ③ 4
 $lo \nearrow hi$
 $lo + pi - 1 \nearrow$
 $pi + 1 \rightarrow hi \nearrow$
 $lo \nearrow hi$
 $pi \nearrow$
 $lo = hi$

Assumption Dry Run:

Single
Element
is selected

$lo = hi$
 $\boxed{0, 0} \nearrow$
 $2, 3$
 $1, 4, pi=1$
 $2, 4, pi=4$
 $5, 4 \nearrow$
 $lo \nearrow hi$
 $@ yearf$
 $lo \nearrow hi$

Not a valid scenario
Array is empty

In this particular scenario, the initial array is [0, 1, 2, 3, 4], which has been sorted. Now, we are examining the second condition. The first base condition is when the "lo" index is equal to the "hi" index. However, upon further consideration of different scenarios, including the one

mentioned earlier, we observe that when we encounter a pivot (in this case, 4), we increment the pivot value by 1 and compare it to the elements in the array. Since there is no element equivalent to "pivot + 1" (i.e., 5) in the array, we can conclude that if we don't have an element greater than our pivot element, we should return from the function. This leads to the introduction of a new condition: "lo > hi". Consequently, we combine both conditions and establish a new base case: "lo >= hi".

```
public class Solution {  
    public static int partitionIndex(int[] arr, int lo, int hi){  
        int pivot = arr[hi];  
        int i = lo;  
        int j = lo;  
  
        while(j < hi){  
            if(arr[j] >= pivot){  
                j++;  
            }else{  
                int temp = arr[i];  
                arr[i] = arr[j];  
                arr[j] = temp;  
            }  
        }  
  
        int temp = arr[i];  
        arr[i] = arr[j];  
        arr[j] = temp;  
  
        return i;  
    }  
    public static void quickSort(int[] arr, int lo, int hi){  
        if(lo >= hi){  
            return ;  
        }  
  
        int pi = partitionIndex(arr, lo, hi);  
        quickSort(arr, lo, pi-1);  
        quickSort(arr, pi + 1, hi);  
    }  
    public int[] solve(int[] A) {  
        quickSort(A, 0, A.length-1);  
        return A;  
    }  
}
```

TC and SC of quickSort

Time Complexity and Space Complexity: →

```
public static void quickSort(int[] arr, int lo, int hi) {
    if(lo >= hi) {
        return;
    }
    int pi = partitionIndex(arr, lo, hi);
    quickSort(arr, lo, pi - 1);
    quickSort(arr, pi + 1, hi);
}
```

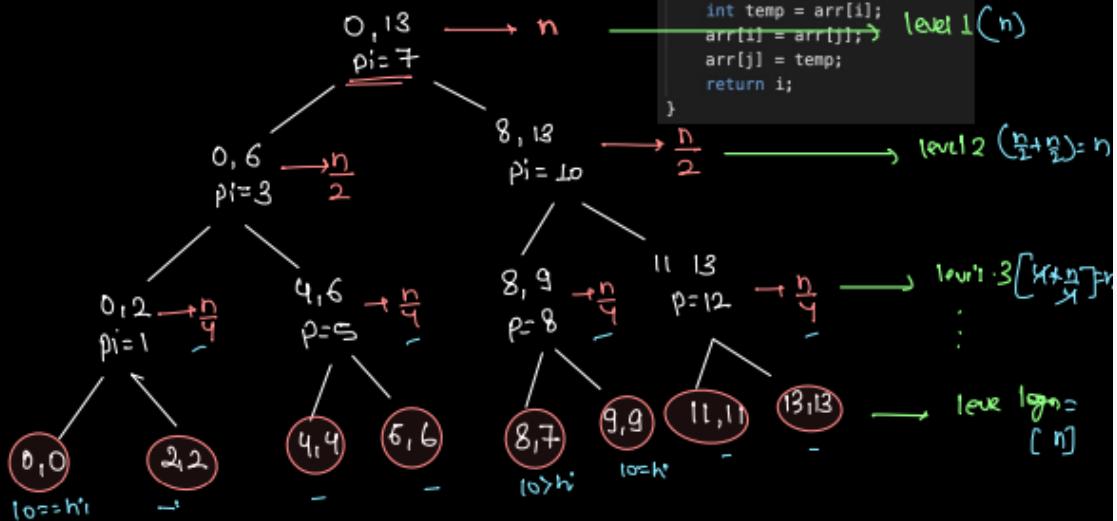
Assumption:

[Partition index is always middle index]

$$T(n) = n + T(n_1) + T(n_2) + k$$

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

```
public static int partitionIndex(int[], int lo, int hi) {
    int pivot = arr[hi];
    int i = lo;
    int j = lo;
    while(j < hi) {
        if(arr[j] >= pivot) {
            j++;
        } else {
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
            i++;
            j++;
        }
    }
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
    return i;
}
```



$$\begin{aligned} \text{Total gtr} &= gtr(\text{level 1}) + gtr(\text{level 2}) + gtr(\text{level 3}) + \dots + gtr(\text{level } \lg n) \\ &= n + \underbrace{n + n + \dots}_{\lg n \text{ time}} + n \end{aligned}$$

$$\text{Total Sto} = n \lg n$$

$$\text{T.C. : } O(n \lg n)$$

$$\text{S.C. : } O(\lg n)$$

Worst case of quickSort

Worst case scenario:

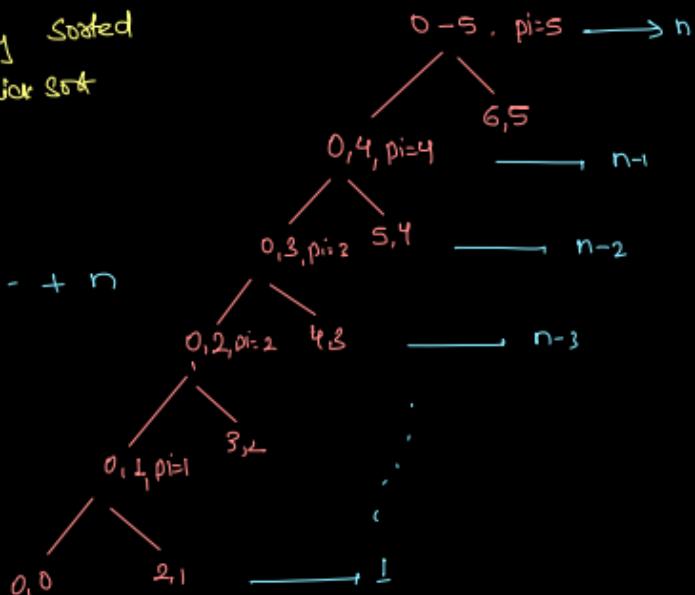
If array is already sorted & we are using quick sort on it.

Total gtr: $1+2+3+4+\dots+n$

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

T.C.: $O(n^2)$

S.C.: $O(n)$



Scenario	T.C.	S.C.
Best	$O(n \log n)$	$O(\log n)$
Worst	$O(n^2)$	$O(n)$

But in Merge Sort → we explicitly find middle element ↗

$$\text{mid} = \frac{l+u}{2}$$

that means merge sort is always $O(n \log n)$
S.C. $O(n)$

Largest Number

`long.parseLong('str')` is a function which convert string to long. Why it required because we can not compare to string directly then we have to convert it in long.

`String.valueOf` -> It is used for convert int to String.

left over:

Problem: Largest Number

Given an array of non-negative integers, arrange them such that they form the longest number.

NOTE: Result may long so return string.

$$\text{arr[]} \rightarrow \{2, 3, 9, 0\} \rightarrow \{9, 3, 2, 0\} \rightarrow 9320$$

Sort it in decr. order -? ~~X~~ Not work

$$\text{arr[]} \rightarrow \{99, 90, 98\} \rightarrow \{99, 98, 90\} \Rightarrow 999890$$
$$\quad \quad \quad \{99, 90, 98\} \Rightarrow 999098$$

$$\text{arr[]} \rightarrow \{998, 9\} \rightarrow \{998, 9\} \Rightarrow 9989$$
$$\quad \quad \quad \{9, 998\} \rightarrow 9998$$

Sort on the basis of digit's \rightarrow lexicographical order. ~~Not work~~

$$\text{arr[]} \rightarrow \{30, 3\} \rightarrow \{30, 3\} \rightarrow 303$$
$$\quad \quad \quad \{3, 30\} \rightarrow 330$$

$$\text{arr[]} \quad \{34, 3\} \rightarrow \{34, 3\} \rightarrow 343$$
$$\quad \quad \quad \{3, 34\} \rightarrow 334$$

Conclusion:

① Sorting on decr. order will not work

② Sorting on digit will also not work

num1 → a
 num2 → b
 possibility \Rightarrow → ab
 └ ba
 ab > ba → a first in array
 ab < ba → b come first

Steps → ① Convert Integer array into String array.

② Arrays.sort (strArray , new Comparator<String> () {

```

public int compare( String a , String b ) {
  String n1= a+b;
  String n2= b+a;
  long val1= Long.parseLong(n1);
  long val2 = Long.parseLong(n2);
  if (val1 > val2) {
    // ab is best → a comes first
    return -1;
  } else if (val1 < val2) {
    // ba is best → b comes first
    return 1;
  } else {
    return 0;
  }
}
  
```

Coordinates

```
import java.util.*;

class Main {
    public static void sortCoordinate() {
        int[][] arr = {
            {1, 1},
            {4, 9},
            {-1, 5},
            {4, 4},
            {3, 0},
            {4, 1},
            {3, 0}
        };
        // convert it into Integer
        Integer[][] A = new Integer[arr.length][2];
        for(int i = 0; i < arr.length; i++) {
            A[i][0] = arr[i][0];
            A[i][1] = arr[i][1];
        }
        for(Integer[] ele : A) {
            System.out.println(ele[0] + " " + ele[1]);
        }
        // sort
        Arrays.sort(A, new Comparator<Integer[]>() {
            public int compare(Integer[] a, Integer[] b) {
                int x1 = a[0];
                int y1 = a[1];
                int x2 = b[0];
                int y2 = b[1];
                // first compare on the basis of x
                if(x1 != x2) {
                    return x1 - x2;
                }
                // if they are same compare on the basis of y
                return y1 - y2;
            }
        });
        System.out.println("~~~~~");
        for(Integer[] ele : A) {
            System.out.println(ele[0] + " " + ele[1]);
        }
    }
}
```

```

    }
    public static void main(String args[]) {
        sortCoordinate();
    }
}

```

DSA: Sorting 1 - 6 - June 2023

- What is sorting
- Bubble Sort
- Insertion Sort
- Merge Two Sorted Array
- Merge Sort

What is sorting -> Arranging data to some specific order is sorting. Arrays.sort(arr), Collections.sort(ArrayList) this is the in build function which has complexity O(NlogN).

Bubble Sort ->

In this method we have to check the max and set it to end. Set end part called sorted reason then on every steps we will check if current > next element then we will swap it. We will use 2 loop to iterate and check every element. It has time complexity $O(n^2)$ and best time complexity is $O(n)$ if no. of swapping 0. It means it is already ordered.

1. **Bubble Sort:**

3	8	6	2	4
0	1	2	3	4

Sorted Region

After gtr 1: 3 6 2 4 8

After gtr 2: 3 2 4 6 8

After gtr 3: 3 2 4 6 8

After gtr 4: 2 3 4 6 8

if n elements is there
(n-1) gtr is required.

Steps:
iterate and swap max data in end and make end part sorted.

3	8	6	2	4
0	1	2	3	4

gtr 1 : 3 ~~6~~ ~~2~~ ~~8~~ ~~4~~ 8
 ↓ ↑ ↓ ↑
 j j+1

gtr 2: 3 ~~2~~ ~~6~~ ~~4~~ 6 8
 ↓ ↑ ↓ ↑
 j j+1

if ($A[j] > A[j+1]$) {
 | swap($A[j], A[j+1]$);
 }
 }

gtr 3: ~~2~~ ~~3~~ 4 6 8
 ↓ ↑ ↑
 j j+1

i = 0 to i < n - 1

[j = 0 to j < n - 1 - i
 if ($A[j] > A[j+1]$) {
 | swap($A[j], A[j+1]$);
 }
]

SCALER ⚡

```

// bubble sort algorithm
public static void bubbleSort(int[] A) {
    int n = A.length;
    for(int i = 0; i < n - 1; i++) {
        for(int j = 0; j < n - 1 - i; j++) {
            if(A[j] > A[j + 1]) {
                // swap the data of A[j] and A[j+1]
                int temp = A[j];
                A[j] = A[j + 1];
                A[j + 1] = temp;
            }
        }
    }
}

```

T.C.: $O(n^2)$

S.C.: $O(1)$

A: 8 1 1 2
0 1 2 3

i	j	
0	0, 1, 2	8 1 1 2 0 1 2 3
1	0, 1	1 8 1 2 0 1 2 3
2	0	1 2 8 0 1 2 3

↓ ↓ ↓ ↓

10 20 30 40 50

Bubble Sort:

Bubble sort not depend on
Best Case T.C. \Rightarrow order of data.

But if no. of swapping is 0

that means data is already sorted.

→ If data is already sorted stop the algorithm.

```

// bubble sort algorithm
public static void bubbleSort(int[] A) {
    int n = A.length;
    for(int i = 0; i < n - 1; i++) {
        boolean swapping = false;
        for(int j = 0; j < n - 1 - i; j++) {
            if(A[j] > A[j + 1]) {
                // swap the data of A[j] and A[j+1]
                int temp = A[j];
                A[j] = A[j + 1];
                A[j + 1] = temp;
                // mark swapping as true for
                // identification of step
                swapping = true;
            }
        }
        if(swapping == false) {
            return;
        }
    }
}

```

Best Case: → data is sorted

T.C. $\rightarrow O(n)$

Worst Case → data is not sorted

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

S.C. $\rightarrow O(1)$

8:19 - 8:30 → Break Time

Insertion Sort -> In this algo we will keep first unchanged and assume it is max and from next element we will check if previous element > to current Element if condition true then swap it either break the loop. We will use 2 loop for this. first loop will start from 0 and second from i+1;

A:	3	8	6	2	4
	0	1	2	3	4
gtr1:	3	8	6	2	4
	0	1	2	3	4
	j				
gtr2:	6	8	2	4	
	3	8	2	3	4
	0	1	2	3	4
	j				
gtr2:	2	3	6	8	
	3	6	8	2	4
	0	1	2	3	4
	j				
gtr4:	4	6	8	2	8
	2	3	6	8	4
	0	1	2	3	4
	j				
	i = 0	to	i < n-1		
	j = i+1	to	j > 0		
	if (A[j] < A[j-1]) {				
	swap(A[j], A[j-1]);				
	}				
	else {				
	break;				
	}				

```
// insertion sort
public static void insertionSort(int[] A) {
    int n = A.length;
    for(int i = 0; i < n - 1; i++) {
        for(int j = i+1; j > 0; j--) {
            if(A[j] < A[j-1]) {
                // swap curr i.e. A[j] with prev. i.e. A[j-1]
                int temp = A[j];
                A[j] = A[j - 1];
                A[j - 1] = temp;
            } else {
                // if curr is greater than prev.
                //that means curr ele is already placed at
                // correct position
                break;
            }
        }
    }
}
```

$T.C = \Theta(n^2)$

§.C. o(1)

i	j	state of array
0	$1 \rightarrow 0$	$\begin{bmatrix} 2 \\ 1 \end{bmatrix}$ 4 6
1	$2 \rightarrow 1$ break	$\begin{bmatrix} 4 \\ 2 \end{bmatrix}$ 4 6
2	$3 \rightarrow 2$ break	$\begin{bmatrix} 6 \\ 4 \end{bmatrix}$ 4 6

Eq.

10

Final answer

2467

Sorted

i	j	state of array
0	$1 \rightarrow \text{break}$	[10] 20 30 40 50
1	$2 \rightarrow \text{break}$	[10] [20] 30 40 50
2	$3 \rightarrow \text{break}$	[10] 20 [30] 40 50
3	$4 \rightarrow \text{break}$	[10] 20 30 [40] 50

T.C: $O(n)$ → Best case in

insertion sort i.e. → array is already sorted.

Merge Two Sorted Array

Problem 1: Given two sorted arrays A and B, Merge them

to make an array which is overall sorted.

Brute force

A: 2 5 9 12 15 → n

① add all elements in

answer array

→ O(n+m)

B: 3 6 8 10 16 18 → m

② sort answer array

Size n+m

Time =
 $\frac{(n+m) \log(n+m)}{2}$
 $\approx \log_2 n$

Ans:

2	3	5	6	8	9	10	12	15	16	18
0	1	2	3	4	5	6	7	8	9	10

Allowed T.C.

O(n+m)

↓
i

A: 2 5 9 12 15

B: 3 6 8 10 16 18

↓ ↓

2	3	5	6	8	9	10	12	15	16	18
0	1	2	3	4	5	6	7	8	9	10

↑
k

A: ~~2~~⁰ ~~5~~¹ ~~9~~² ~~12~~³ ~~15~~⁴

↓ ↓ ↓ ↓ ↓

B: ~~3~~⁰ ~~6~~¹ ~~8~~² ~~10~~³ ~~16~~⁴ ~~18~~⁵

↓ ↓ ↓ ↓ ↓

res[k] = A[i]

res[k] = B[j]

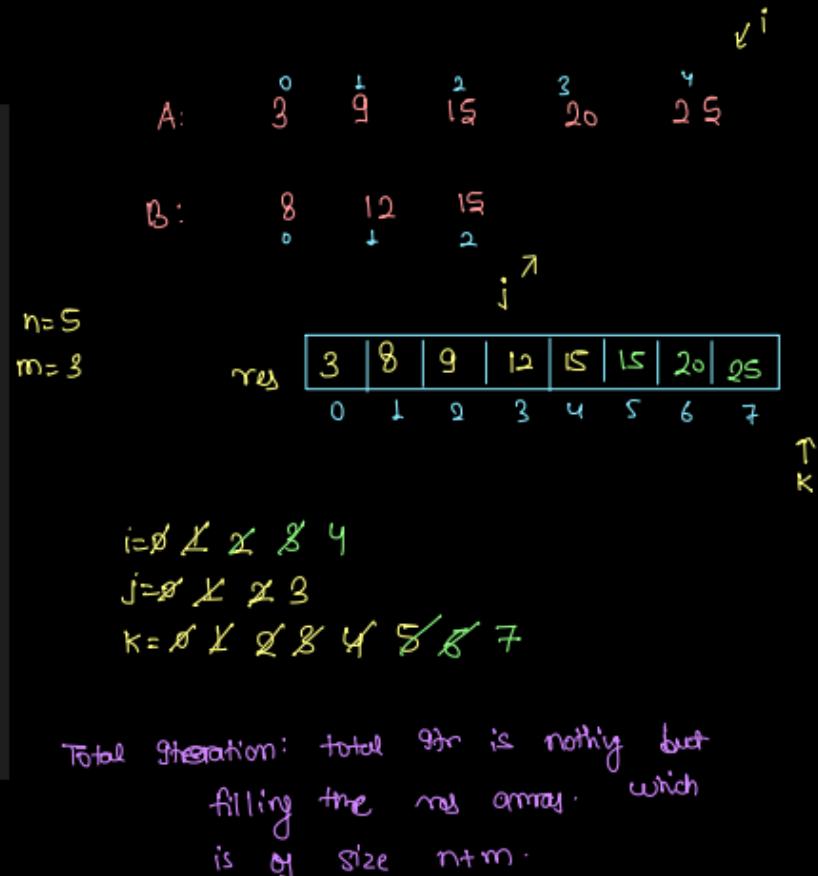
3	8	9	12	15	15	20	25
0	1	2	3	4	5	6	7

k

```

i=0, j=0, k=0
while(i < n && j < m) {
    if(A[i] < B[j]) {
        res[k] = A[i];
        i++;
    } else {
        res[k] = B[j];
        j++;
    }
    k++;
}
// if i is left
while(i < n) {
    res[k] = A[i];
    i++;
    k++;
}
// if j is left
while(j < m) {
    res[k] = B[j];
    j++;
    k++;
}
return res;

```



T.C. : O(n+m)

S.C: O(1) → Because we are not creating result array for our size instead question is asking to return arr.

Merge Sort -> Divide and conquer algorithm (TC: NlogN)

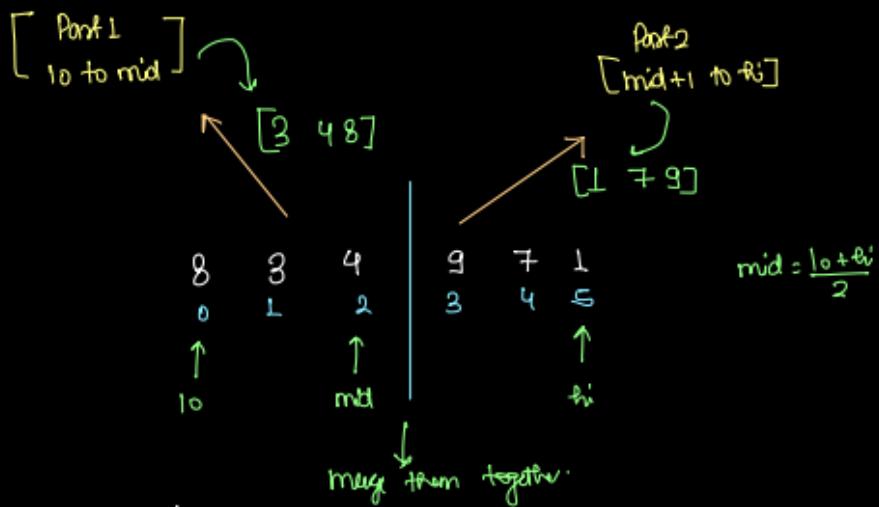
This algorithm based on recursion. In this algorithm We will get the mide point using $lo+hi/2$ like if array has 1,2,3,4,5,6 then lo is 0 index and hi is 5th index and mid point is 2. now we will make it two half of array and then we will merge both the array till we will not reach to base case where $lo == hi$.

Merge Sort:

T.C. $O(n \log n)$

divide and conquer algorithm

→ this algorithm is based on
Recursion.



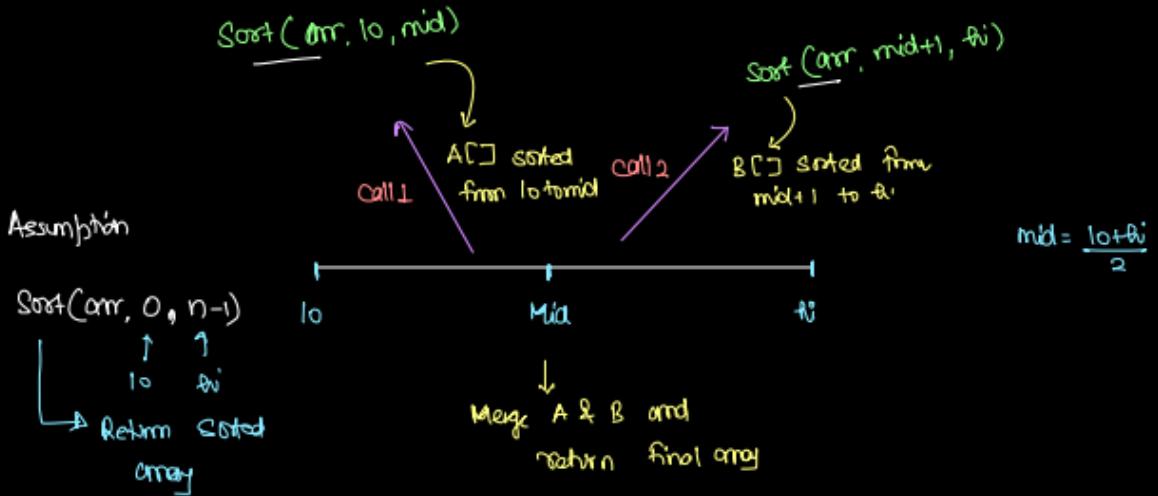
Assumption → arr, lo, hi
Sort array from lo to hi

$[1, 3, 4, 7, 8, 9]$

Main logic: → $\text{sort}(\text{arr}, \text{lo}, \text{mid}) \rightarrow A$

$\text{sort}(\text{arr}, \text{mid}+1, \text{hi}) \rightarrow B$

My task → Merge them together.



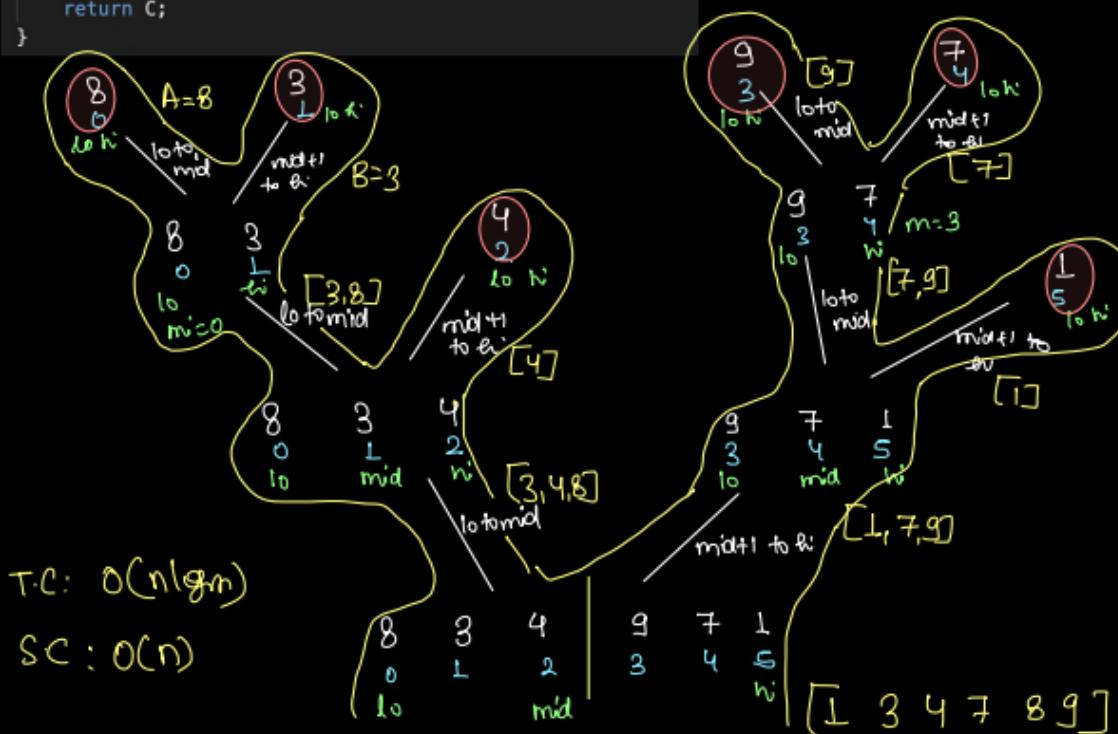
```
public static int[] mergeSort(int[] arr, int lo, int hi) {
    // base case
    if (lo == hi) {
        int[] bres = new int[1];
        bres[0] = arr[lo];
        return bres;
    }
    int mid = (lo + hi) / 2;
    int[] A = mergeSort(arr, lo, mid);
    int[] B = mergeSort(arr, mid + 1, hi);
    int[] C = mergeTwoSortedArray(A, B);
    return C;
}
```

$$\text{Recurrence Relation}$$

$$T(n) = T(n_1) + T(n_2) + n$$

\downarrow \downarrow

1st call 2nd call merge two sorted array



DSA: Sorting 1 - 8 - June 2023

- Inversion Count
- Custom Comparison
- Factor Sort
- Sort Coordinates
- Largest Number

Inversion Count -> if $A[i] > A[j]$ then all pairs with $A[i]$ called inversions like below. It will solved using recursion shorting function like merge sort where we will divid array in 2 parts and then call same function using $\text{mergeSort}(A, \text{lo}, \text{mid})$, $\text{mergeSort}(A, \text{mid}+1, \text{hi})$ and mid will calculate $n+m/2$

Problem 1 Inversion Count

Given an array of integers A . If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an inversion of A . Find total number of inversions in A .

NOTE: Return count $\% (10^9 + 7)$

$A[i], A[j]$	$A[i], A[j]$
$[2, 0]$	$[2, 1]$
$[3, 0]$	$[3, 1]$

Inversion count = 4 Ans

$A[i]$	$A[j]$
8	5
0	1
3	4
2	3
4	1
5	6
6	2

$A[i], A[j]$	$(8, 5)$	$(5, 3)$	$(3, 1)$	$(4, 1)$	$(6, 2)$
	$(8, 3)$	$(5, 4)$	$(3, 2)$	$(4, 2)$	
	$(8, 4)$	$(5, 1)$			
	$(8, 1)$	$(5, 2)$			
	$(8, 6)$				
	$(8, 2)$				

Inversion Count = 15 Ans

Brute force idea:

for value of $A[i]$, we will count value of $A[j]$ in which $A[i] > A[j]$ for $j = i+1$ to $n-1$.

```
int inversionCount(int[] A) {
    int n = A.length;
    long count = 0;
    for(int i=0; i<n; i++) {
        for(int j=i+1; j<n; j++) {
            if(A[i] > A[j]) {
                count = (count + 1)%mod
            }
        }
    }
    return (int)(count);
}
```

T.C: $O(n^2)$

S.C: $O(1)$

Time complexity

$$\begin{array}{lll} i=0, j=1 \text{ to } n-1 & \longrightarrow & n-1 \\ i=1, j=2 \text{ to } n-1 & \longrightarrow & n-2 \\ i=2, j=3 \text{ to } n-1 & \longrightarrow & n-3 \\ \vdots & & \vdots \\ i=n-2, j=n-1 \text{ to } n-1 & \longrightarrow & 1 \end{array}$$

Total time: $1+2+3+\dots+n-2+n-1$

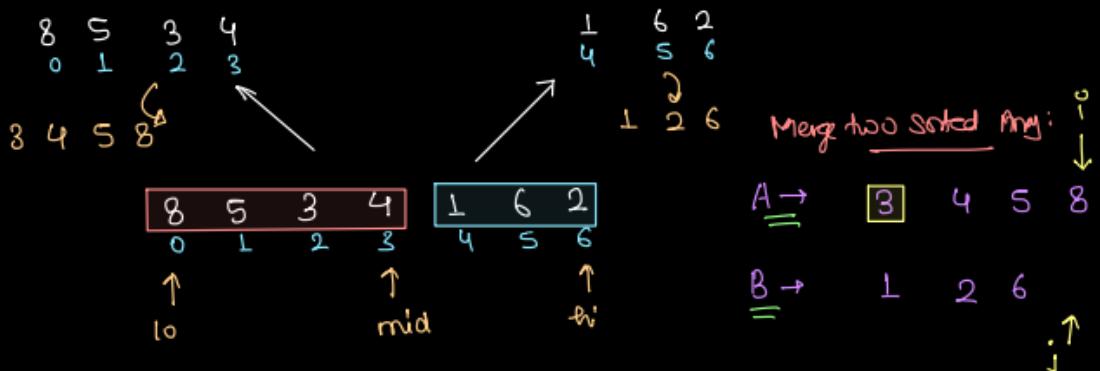
$$= \frac{n(n-1)}{2}$$

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

Expected T.C: $O(n \log n)$

$$A[] = \boxed{8 | 5 | 3 | 4 | 1 | 6 | 2}$$

→ sort array (using Merge Sort)



$$\text{mid} = \frac{l_0 + h_1}{2}$$

$$= \frac{0+6}{2} = 3$$

```

if (A[i] > B[j]) {
    ans[k] = B[j];
    j++;
    [Count inversion pair]
}

```

3 else {

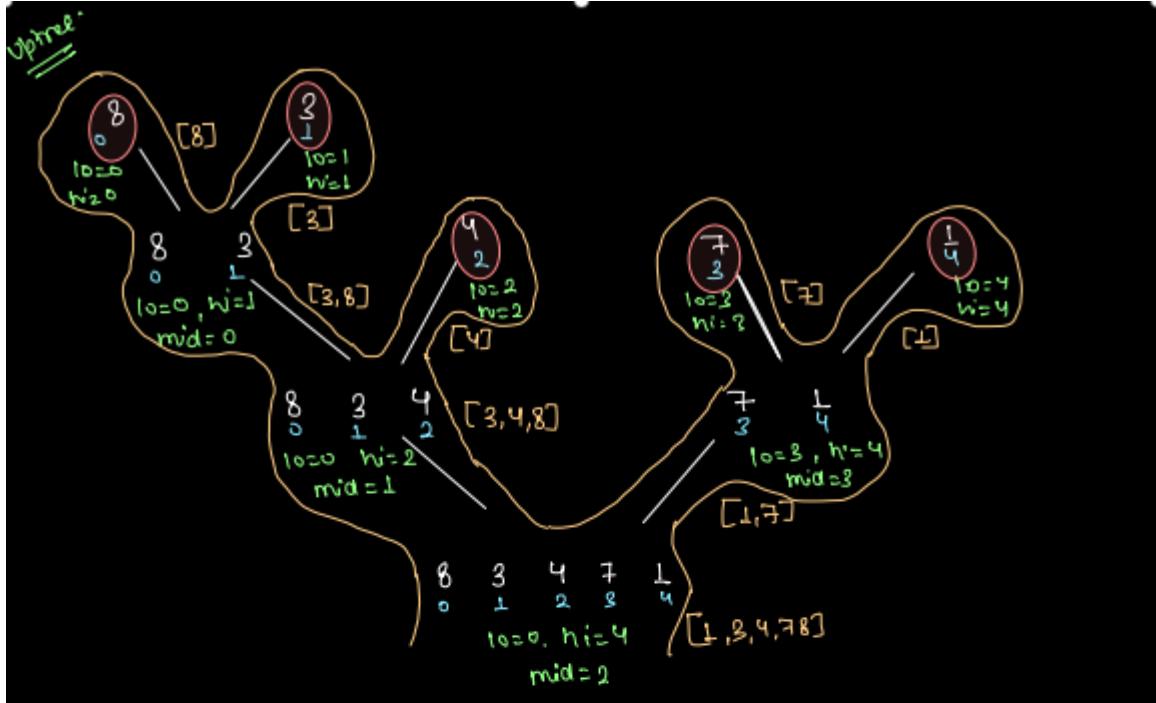
```

    ans[k] = A[i];
    i++;
}

```

3
k++:

3,1 4,1 5,1 8,1
3,2 4,2 5,2 8,2
8,6



$A[i] \rightarrow 3 \ 4 \ 8$

$B[j] \rightarrow 1 \ 7$

\uparrow

$A[i] > B[j]$

Count += (n-i)

Count = $0 + 1 + 1 + 1 + 3 + 1$

= 7 Ans

(8,3), (8,4), (7,1)

(3,1) (4,1) (7,1)

(8,7)

We are just adding one
single line in merge two sorted array

8:32 - 8:45
Break time

→ Rest Every code is same

T.C. : $O(n \log n)$

S.C. : $O(n)$

Custom Comparison -> We will write custom short function for custom comparison like if we want to sort car by price or by feature then we have to write it because default sort feature only provide chat, int sorting. for custom sorting function we will use sort method and pass second argument as comparator function like below.

We will use Integer class reference type not primitive type int and array also we have to convert in int to Integer. We will use @Override it is optional. If we want to return first parameter

```
Arrays.sort(arr, new Comparator<Integer>() {
    @Override
    public int compare(Integer a, Integer b) {
        // Possibility 1 : want to keep a before b
        // return negative value
        // Possibility 2 : want to keep b before a
        // return positive value
        // Possibility 3 : Doesnot matter
        // return 0

        // noraml Order
        /*
        if(a < b) {
            // want to keep a before b
            return -1;
        } else if(a > b) {
            // want to keep b before a
            return 1;
        } else {
            // Doesnot matter
            return 0;
        }
        */
        // reverse the order
        if(a < b) {
            // want to keep b before a
            return 1;
        } else if(a > b) {
            // want to keep a before b
            return -1;
        } else {
            // Doesnot matter
            return 0;
        }
    }
}
```

```

// sort only use this instead above
return a - b;
// for order reversal
// return b - a;
}
});

```

Factor Sort

Problem 2: Factor Sort

Given an array A. Sort the array in increasing order of number of distinct factors.

- * Least number of factor will come first
- * If two numbers have same factor the less value will come first.

Eg → [4 ↓ 7 ↓ 6 ↓ 9 ↓ 8 ↓ 2 ↓ 10 ↓]
 ↓ ↓ ↓ ↓ ↓ ↓ ↓
 3 2 4 3 4 2 4
 factor count

Sort on the basis of number of factors

Sorted: 2 7 4 9 6 8 10
 array

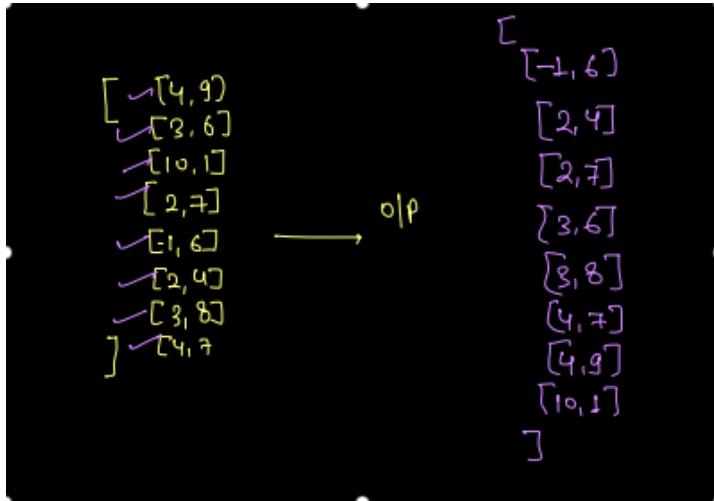
on basis of count of factors.

Problem: Sort coordinates:

Hint: { * first decide on the basis of x-coordinate.
 { * If x-coordinate is same for two different coordinates, then decide on the basis of y.

→ [4, 9] → val1
 → [3, 6] → val2
 → [10, 1] → val2
 → [2, 7] → val1 → Integer[][]
 → [1, 6] → two values val1 → Integer[] a
 → [2, 4] val2 → Integer[] b
 → [3, 8]
] [4, 7]

TODO
write logic



```

class Main {
/* Problem : factor Sort */
public static int factorCount(int val) {
// todo : write optimise code
int fcount = 0;
for(int i = 1; i <= val; i++) {
if(val % i == 0) {
fcount++;
}
return fcount;
}
public static int[] factorSort(int[] arr) {
int n = arr.length;
// convert int array into Integer array
Integer[] A = new Integer[n];
for(int i = 0; i < n; i++) {
A[i] = arr[i];
}
Arrays.sort(A, new Comparator<Integer>() {
@Override
public int compare(Integer a, Integer b) {
int fa = factorCount(a);
int fb = factorCount(b);
if(fa < fb) {
// a come first
return -1;
} else if(fa > fb) {
// b come first
return 1;
} else {
}
}
}

```

```

// if both are equal
// decide according to value of a and b
return a - b;
}
};

// fill arr from A
for(int i = 0; i < n; i++) {
arr[i] = A[i];
}
return arr;
}

public static void main(String args[]) {
// Your code goes here
int[] arr = {4, 7, 6, 9, 8, 2, 10};
System.out.println(Arrays.toString(arr));
arr = factorSort(arr);
System.out.println(Arrays.toString(arr));
}
}

```

Largest Number

Problem : Largest Number

Given an array of non-negative integers, arrange them such that they form the longest number.

NOTE: Result may long so return String.

→ Next Session

doubt session:

Sort →

val 1
val 2

int res = compare(val1, val2);
+ve ⇒ val 2 is greater.
-ve ⇒ val 1 is smaller &
will come first

O

[gmbuilt sort → pre defined function]

comparison in
manual sort if(arr[i] < arr[j])
function [+] }

comparison
→ Comparator
pass lambda
function
→ Comparable
it allows to
modify user
defined class.

DSA: Sorting 3 - 10 - June 2023

- Sort 0 1
- Partition of Array
- Quick Sort Algorithm
- TC and SC of quickSort
- Worst case of quickSort
- Largest Number
- Sort Coordinates

Sort 0 1 → sort all 0's first then 1's after. We can use any sorting algorithm but we have to sort this in $O(n)$. we will use Aproch swap in single loop. We will define 2 initial variable i and j to 0 and then we will apply condition if $j == 1$ then $j++$ go to next element and if $j != 1$ then we will swap in same array j to i and i to j.

~~~~~

**Sort 0 1**

~~~~~

You are given an array of 0s and 1s in random order. Segregate 0s on left side and 1s on right side of the array [Basically you have to sort the array].

array A → 0 1 0 0 1 1 0 1 0
0|p ! 0 0 0 0 0 1 1 1 1

Idea 1: Sort the entire array

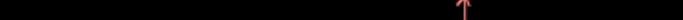
1. Bubble Sort → $O(n^2)$
2. Insertion Sort → $O(n^2)$
3. Merge Sort → $O(n \log n)$
4. Inbuilt sort → $O(n \log n)$

Expected T.C → $O(n)$

Idea 2: With the help of count strategy.

Step 1: Count number of 0s
Step 2: Count number of 1s] → First Iteration
Step 3 → Set 0's first & then 1's] → Second Iteration.

T.C: $O(n)$, Generate 2 times the array.

array → 

$i \rightarrow$ First index of L
 $j \rightarrow$ First index of unsorted part

```

if( arr[j] == 1) {
    j++;
}
else {
    swap(arr[i], arr[j]);
    i++;
    j++;
}

```

• **ANSWER**

```

int i=0;
int j=0;
while(j < arr.length) {
    if(arr[j] == 1) {
        j++;
    } else {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
        i++;
    }
}
return arr;

```

T.C. $O(n)$
in single iteration

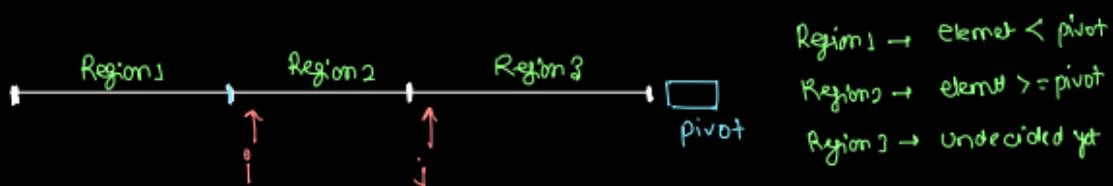
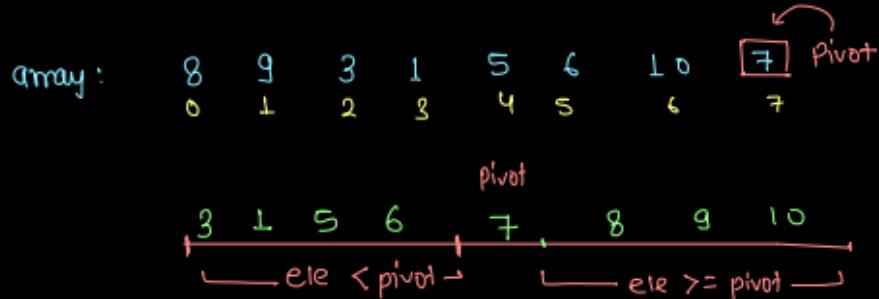
-1000 Soft 0 ± 2

A horizontal line representing a binary search tree. The left part is labeled '0's' and the right part is labeled '1's'. A bracket under the line between these two regions is labeled 'Unsorted Region'.

Partition of Array

Partition An Array

Given an array, partition it based on last element(Pivot Element), Such that all the elements < pivot are coming on left of it and all elements \geq pivot are coming on right of it.



$i \rightarrow$ first index
of Region 2

$j \rightarrow$ first index
of Region 2

$arr[j] < \text{pivot}$	$arr[j] \geq \text{pivot}$
Swap ($arr[i], arr[j]$)	// increase Region 2
$i++$	$j++$
$j++$	



final array $\underbrace{3}_{\text{ele} < \text{pivot}} \underbrace{1}_{\text{pivot}} \underbrace{5}_{\text{ele} > \text{pivot}} \underbrace{6}_{\text{pivot}} \underbrace{8}_{\text{pivot}}$

NOTE: for last element, move it in middle of partition

when $j = n-1$ (i.e. index of pivot)
swap arr[i], arr[j]

pivot = arr[n-1]

```

if (arr[j] >= pivot) {
    j++;
}
else {
    swap(arr[i], arr[j]);
    i++;
    j++;
}

```

int partitionIndex (int[] arr) {

```

int i=0;
int j=0;
int pivot = arr[arr.length-1];
while(j < arr.length-1) {
    if (arr[j] >= pivot) {
        j++;
    }
}

```

```

else {
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
    i++;
    j++;
}

```

// j is pointing to last elem (i.e. pivot)

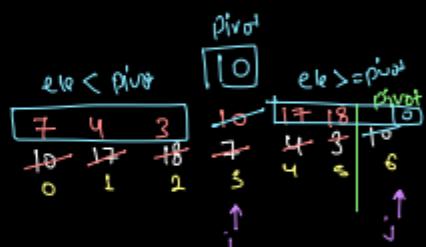
```

int temp = arr[i];
arr[i] = arr[j];
arr[j] = temp;

```

return i; → partition arr

3



$i=0$
 $j=0 \neq 2$
 $pivot=0$

Quick Sort Algorithm

Quick Sort Algorithm:

↳ Divide and Conquer Sorting Algorithm
↳ Recursion

Steps: → ① array from lo to hi index

② assume last element is pivot element

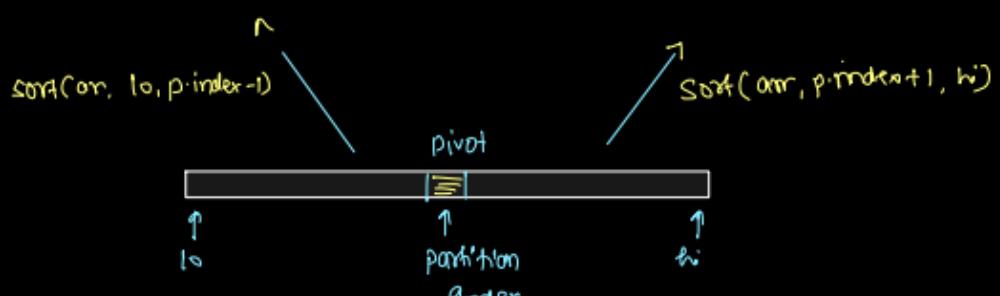
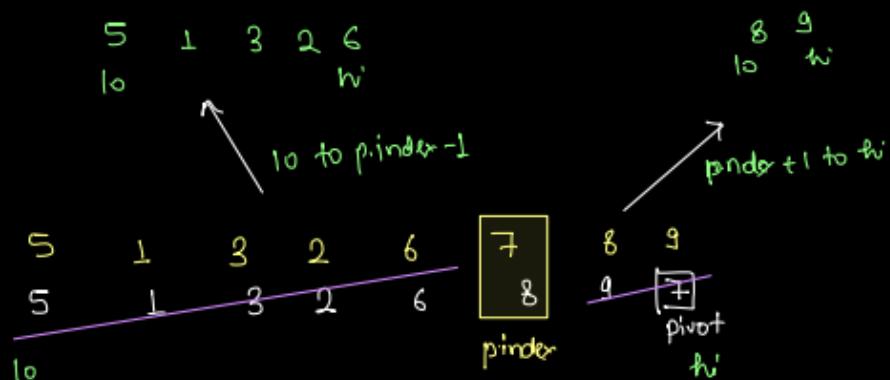
find position smaller

(After finding $p.i$, we are sure that $p.i$ is correctly placed at its position)

③ Ask from recursion to sort

$\text{sort}(\text{arr}, lo \text{ to } p.\text{index}-1)$

$\text{sort}(\text{arr}, p.\text{index}+1 \text{ to } hi)$



Assume function that mechanism works

```
void quickSort(int arr[], int lo, int hi) {
```

```
    if (lo >= hi) {
        return;
    }
```

```
    int pi = partitionIndex(arr, lo, hi);
    quickSort(arr, lo, pi - 1);
    quickSort(arr, pi + 1, hi);
```

3

$lo = hi$

$\frac{1}{0}$
 $lo \nearrow hi$

$\perp \quad [2] \quad 5 \quad 3 \quad 4$
5 3 \perp 4 ②
0 ① 2 3 1
 $lo \nearrow hi$

$lo = hi$
 $3 \quad 2$
 $lo \nearrow hi$ $lo + pi - 1$
 $5 \quad 4$
 $lo \nearrow hi$ $pi + 1 \rightarrow hi$
 $3 \quad 4 \quad 5$
5 3 ④
2 ③ 4
① ② 1
 $lo \nearrow hi$ $pi \nearrow hi$
 $lo \nearrow hi$

Assumption Dry Run:

Single
Element
is selected

$lo = hi$
 $[0, 0]$
 $lo \nearrow hi$

$2, 3$
 $lo \nearrow hi$

$2, 4, pi=4$
 $lo \nearrow hi$

$0, 4, pi=1$
 $lo \nearrow hi$

$lo \nearrow hi$

$5, 4$
 $lo \nearrow hi$

$Not a valid scenario$
 $Array is empty$

@ yearf

In this particular scenario, the initial array is $[0, 1, 2, 3, 4]$, which has been sorted. Now, we are examining the second condition. The first base condition is when the "lo" index is equal to the "hi" index. However, upon further consideration of different scenarios, including the one

mentioned earlier, we observe that when we encounter a pivot (in this case, 4), we increment the pivot value by 1 and compare it to the elements in the array. Since there is no element equivalent to "pivot + 1" (i.e., 5) in the array, we can conclude that if we don't have an element greater than our pivot element, we should return from the function. This leads to the introduction of a new condition: "lo > hi". Consequently, we combine both conditions and establish a new base case: "lo >= hi".

```
public class Solution {  
    public static int partitionIndex(int[] arr, int lo, int hi){  
        int pivot = arr[hi];  
        int i = lo;  
        int j = lo;  
  
        while(j < hi){  
            if(arr[j] >= pivot){  
                j++;  
            }else{  
                int temp = arr[i];  
                arr[i] = arr[j];  
                arr[j] = temp;  
            }  
        }  
  
        int temp = arr[i];  
        arr[i] = arr[j];  
        arr[j] = temp;  
  
        return i;  
    }  
    public static void quickSort(int[] arr, int lo, int hi){  
        if(lo >= hi){  
            return ;  
        }  
  
        int pi = partitionIndex(arr, lo, hi);  
        quickSort(arr, lo, pi-1);  
        quickSort(arr, pi + 1, hi);  
    }  
    public int[] solve(int[] A) {  
        quickSort(A, 0, A.length-1);  
        return A;  
    }  
}
```

TC and SC of quickSort

Time complexity and Space complexity: →

```
public static void quickSort(int[] arr, int lo, int hi) {
    if(lo >= hi) {
        return;
    }
    int pi = partitionIndex(arr, lo, hi);
    quickSort(arr, lo, pi - 1);
    quickSort(arr, pi + 1, hi);
}
```

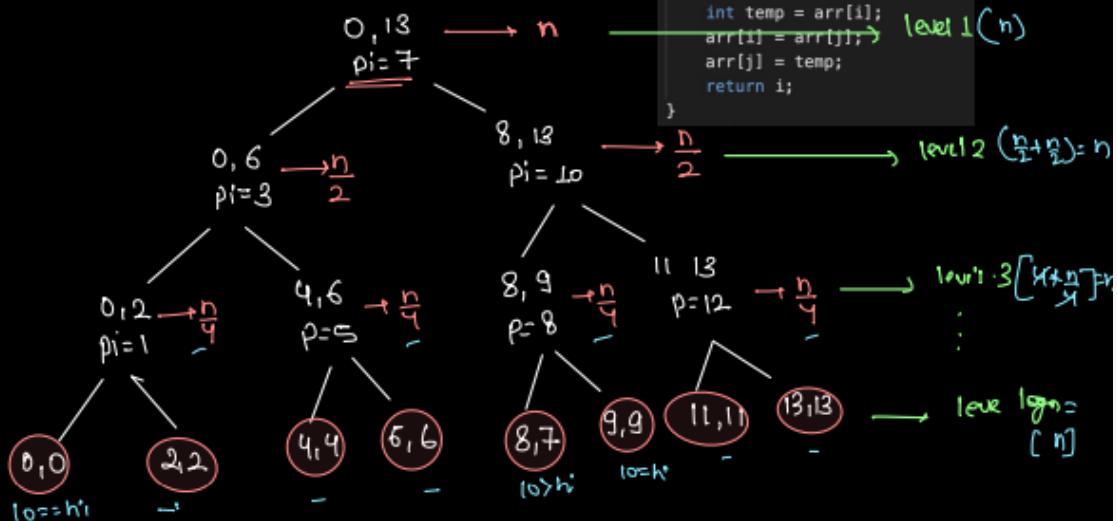
Assumption:

[Partition index is always middle index]

$$T(n) = n + T(n_1) + T(n_2) + k$$

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

```
public static int partitionIndex(int[], int lo, int hi) {
    int pivot = arr[hi];
    int i = lo;
    int j = lo;
    while(j < hi) {
        if(arr[j] >= pivot) {
            j++;
        } else {
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
            i++;
            j++;
        }
    }
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
    return i;
}
```



$$\begin{aligned} \text{Total gtr} &= gtr(\text{level 1}) + gtr(\text{level 2}) + gtr(\text{level 3}) + \dots + gtr(\text{level } \lg n) \\ &= n + \underbrace{n + n + \dots + n}_{\text{lg n time}} \end{aligned}$$

$$\text{Total Sto} = n \lg n$$

$$\text{T.C. : } O(n \lg n)$$

$$\text{S.C. : } O(\lg n)$$

Worst case of quickSort

Worst case scenario:

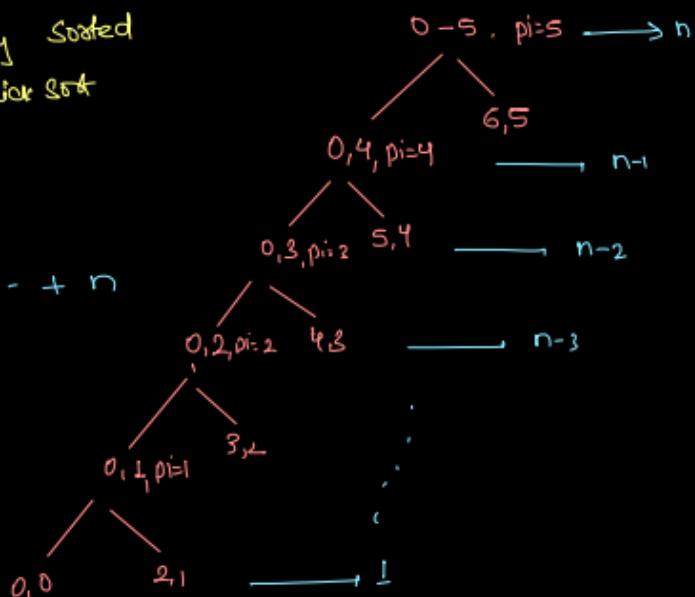
If array is already sorted & we are using quick sort on it.

Total gtr: $1+2+3+4+\dots+n$

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

T.C.: $O(n^2)$

S.C.: $O(n)$



Scenario	T.C.	S.C.
Best	$O(n \log n)$	$O(\log n)$
Worst	$O(n^2)$	$O(n)$

But in Merge Sort → we explicitly find middle element ↗

$$\text{mid} = \frac{l+u}{2}$$

that means merge sort is always $O(n \log n)$
S.C. $\Theta(n)$

Largest Number

`long.parseLong('str')` is a function which convert string to long. Why it required because we can not compare to string directly then we have to convert it in long.

`String.valueOf` -> It is used for convert int to String.

left over:

Problem: Largest Number

Given an array of non-negative integers, arrange them such that they form the longest number.

NOTE: Result may long so return string.

$$\text{arr[]} \rightarrow \{2, 3, 9, 0\} \rightarrow \{9, 3, 2, 0\} \rightarrow 9320$$

sort it in decr. order -? ~~X~~ Not work

$$\text{arr[]} \rightarrow \{99, 90, 98\} \rightarrow \begin{cases} \{99, 98, 90\} \Rightarrow 999890 \\ \{99, 90, 98\} \Rightarrow 999098 \end{cases}$$

$$\text{arr[]} \rightarrow \{998, 9\} \rightarrow \begin{cases} \{998, 9\} \Rightarrow 9989 \\ \{9, 998\} \Rightarrow 9998 \end{cases}$$

Sort on the basis of digits \rightarrow lexicographical order. ~~Not work~~

$$\text{arr[]} \rightarrow \{30, 3\} \rightarrow \begin{cases} \{30, 3\} \rightarrow 303 \\ \{3, 30\} \rightarrow 330 \end{cases}$$

$$\text{arr[]} \rightarrow \{34, 3\} \rightarrow \begin{cases} \{34, 3\} \rightarrow 343 \\ \{3, 34\} \rightarrow 334 \end{cases}$$

Conclusion:

① Sorting on decr. order will not work

② Sorting on digit will also not work

num1 → a
 num2 → b
 possibility \Rightarrow → ab
 └ ba
 ab > ba → a first in array
 ab < ba → b come first

Steps → ① Convert Integer array into String array.

② Arrays.sort (strArray , new Comparator<String> () {

```

public int compare( String a , String b ) {
  String n1= a+b;
  String n2= b+a;
  long val1= Long.parseLong(n1);
  long val2 = Long.parseLong(n2);
  if (val1 > val2) {
    // ab is best → a comes first
    return -1;
  } else if (val1 < val2) {
    // ba is best → b comes first
    return 1;
  } else {
    return 0;
  }
}
  
```

Coordinates

```
import java.util.*;

class Main {
    public static void sortCoordinate() {
        int[][] arr = {
            {1, 1},
            {4, 9},
            {-1, 5},
            {4, 4},
            {3, 0},
            {4, 1},
            {3, 0}
        };
        // convert it into Integer
        Integer[][] A = new Integer[arr.length][2];
        for(int i = 0; i < arr.length; i++) {
            A[i][0] = arr[i][0];
            A[i][1] = arr[i][1];
        }
        for(Integer[] ele : A) {
            System.out.println(ele[0] + " " + ele[1]);
        }
        // sort
        Arrays.sort(A, new Comparator<Integer[]>() {
            public int compare(Integer[] a, Integer[] b) {
                int x1 = a[0];
                int y1 = a[1];
                int x2 = b[0];
                int y2 = b[1];
                // first compare on the basis of x
                if(x1 != x2) {
                    return x1 - x2;
                }
                // if they are same compare on the basis of y
                return y1 - y2;
            }
        });
        System.out.println("~~~~~");
        for(Integer[] ele : A) {
            System.out.println(ele[0] + " " + ele[1]);
        }
    }
}
```

```

    }
public static void main(String args[]) {
sortCoordinate();
}
}

```

DSA: Searching - 13 - June 2023

- Linear Search
- Binary Search
- First Occurrence of K
- Floor Of K in Array
- Local Minima

Linear Search -> It is simple search using loop

1. Linear Search

Given an array A[], find out if K is present in the array or not. If it is present, return the index of K in the array; otherwise, return -1.

array: 2 9 8 17 42 1
0 1 2 3 4 5

K=19 → ans = -1
K=17 → ans = 2

int linearSearch(int[] A, int k) {
 int ans = -1;
 for (int i = 0; i < A.length; i++) {
 if (A[i] == k) {
 ans = i;
 break;
 }
 }
 return ans;
}

T.C: O(n)

i	A[i]
0	2 → x
1	9 → x
2	8 → x
3	17 → x
4	42 → x
5	1 → x
6	loop stop

K=19 ans=-1
K=17 ans=2
K=42, ans=1 (4)
ans[i]
0 2 → x
1 9 → x
2 8 → x
3 17 → x
4 42 → x
4 42 → break the loop
ans=4

Binary Search -> In our search algorithm, we are using the Binary Search method, which has a time complexity of $O(\log N)$. This will work if array is already sorted in ascending order. If it is in random order then we will use linear search.

To avoid integer overflow we can get middle index like

$$hi - (hi - lo) / 2,$$

$$hi - hi / 2 + lo / 2$$

Here's how the algorithm works:

1. We initialize two indices, "lo" and "hi," representing the lower and upper bounds of the search range. We calculate the midIndex as $(hi + lo) / 2$.
2. Inside a loop, we have three conditions to check:
 - If arr[midIndex] is equal to the userValue, we have found a match, so we return true and exit the loop.
 - If arr[midIndex] is greater than the userValue, we discard the right portion of the range by updating hi to midIndex - 1.
 - If arr[midIndex] is less than the userValue, we discard the left portion of the range by updating lo to midIndex + 1.
3. We continue the loop as long as the condition $lo \leq hi$ is true.

This algorithm allows us to efficiently search for a value in a sorted array by continually dividing the search range in half.

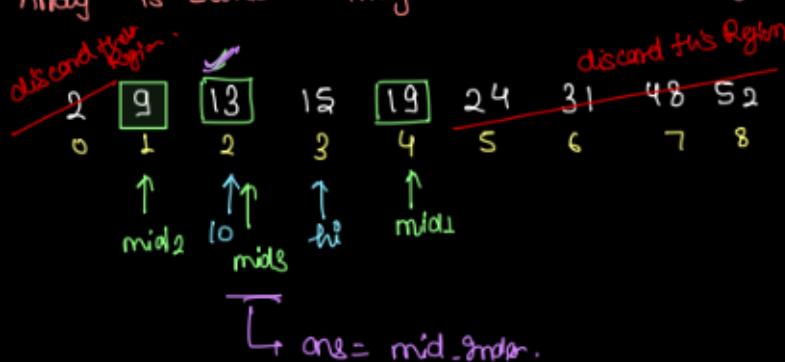
2. Binary Search

Given a sorted array A[], find out if K is present in the array or not. If it is present, return the index of K in the array; otherwise, return -1.

array : 2 9 13 15 19 24 31 48 52
 0 1 2 3 4 5 6 7 8

Apply Linear Search : → ✓ T.C. $O(n)$

* Array is sorted : array is in increasing order



$$K=12$$

$$lo=0, hi=8, mid=4$$

$$lo=0, hi=3, mid=1$$

$$lo=2, hi=9, mid=5$$

$$mid = \frac{lo + hi}{2}$$

① $\text{arr}[\text{mid}] == \text{k}$: found the answer, $\text{end} = \text{mid}$, `break`:

② $\text{arr}[\text{mid}] > \text{k}$: discard right Region
 $h^* = \text{mid} - 1$

③ $arr[mid] < k$; discord left right
 $lo = mid + 1$

$$\frac{(l_0 + h_i)}{2} \Rightarrow \frac{l_0}{2} + \frac{h_i}{2}$$

```
int searching(int[] A, int k){
```

18

```
int lo=0;
```

int Ni = A.length - 1;

```
while( ) {
```

```
int mid = (lo+hi)/2;
```

if(A[mid] == k) {

//data found

```
return mid;
```

se if (A[mid] > K)

✓discard Rig

```
    if (A[mid] < B) {
```

// discard left Region

$lo = mid + 1;$

10 - $\pi \alpha^2 + \epsilon$

3

```
return -1;
```

3

A: 2 4 8 13 19 29 38 42 49

0 1 2 3 4 5 6 7 8

mid2 ↑ 10 ↑ mid3 hv ↑ mid1

lo	hi	mid	Action
0	8	4	$\text{arr}[\text{mid}] > k \&$ $\text{hi} = \text{mid} - 1, \text{hi} = 3$
0	3	1	$\text{arr}[\text{mid}] \leq k$ $\text{lo} = \text{mid} + 1, \text{lo} = 2$
2	3	2	$\text{arr}[\text{mid}] == k$

9 - 1 is only

First Occurrence of K

Given a sorted array $A[]$, find out if K is present in the array or not. If it is present return index of first occurrence of K in the array; otherwise, return -1 .

Expected T.C: $O(\log n)$

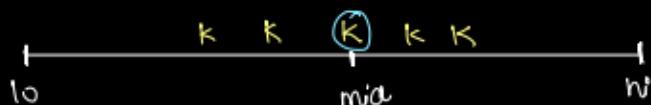
2	2	3	4	5	5	6	7	7	8	9	10	12	19	20
0	1	2	3	4	5	6	7	8	9	10	11	12	13	
		↑												

$K=3 \rightarrow \text{ans} \leftarrow 2$

$K=5 \rightarrow \text{ans} \leftarrow 4$

$K=10 \rightarrow \text{ans} \leftarrow -1$

$K \rightarrow$ data to find



$\text{arr}[mid] == K$

$\text{ans} = \text{mid}$:

keep looking for K in
left side.

$hi = mid - 1$

$K=5$



$\text{ans} = \text{mid2}$

$mid =$

$$\begin{cases} \text{arr}[mid] == K & \text{if } \text{mid} = \text{mid} \\ & \text{Right Region} \\ & \text{else} \\ \text{arr}[mid] > K & \Rightarrow \text{Right Region} \\ \text{arr}[mid] < K & \Rightarrow \text{Left Region} \end{cases}$$

```

3 4 5 5 5 5 7 7
0 1 2 3 4 5 6 7 8 9 ...
int firstIndex (int[] A, int K) {
    int lo = 0, hi = A.length - 1;
    int ans = -1;
    while (lo <= hi) {
        int mid = (lo + hi) / 2;
        if (A[mid] == K) {
            // Data found → move answer &
            // ans = mid; Discard Right Region
            ans = mid;
            hi = mid - 1;
        } else if (A[mid] > K) {
            // Discard Right Region
            hi = mid - 1;
        } else if (A[mid] < K) {
            // Discard Left Region
            lo = mid + 1;
        }
    }
    return ans;
}

```

lo	hi	mid	Action
0	9	4	$A[mid] == K$ $hi = mid - 1$
0	3	1	$A[mid] < K$ $lo = mid + 1$
2	3	2	$A[mid] == K$ $hi = mid - 1$
2	1	-	Loop stops.

$ans = 5 \neq 2$

T.C: $O(\log n)$

To Do:

- How to find last occurrence of K in array
- Using Binary Search

Break time: 8:25 - ~~8:35 AM~~
8:40 AM

2 more problems.

Floor Of K in Array

4. Floor of K in Array

Floor of K

$$\begin{cases} \text{Floor}(\frac{2}{2}) = 2 \\ \text{Floor}(\frac{3}{2}) = 2 \\ \text{Floor}(\frac{9}{2}) = 4 \end{cases}$$

Given a sorted array A[], find out if K is present in the array or not. If it is present return index of floor of K in the array: otherwise, return -1.

Value
 $\text{Floor}(k) \Rightarrow \text{max of all elements which are } \leq k$

Array	A:	12	19	21	25	28	32	35	38	42	51
		0	1	2	3	4	5	6	7	8	9

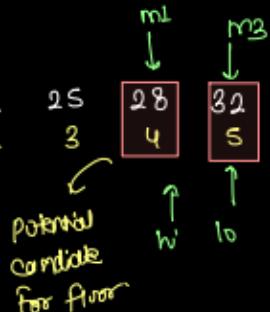
k	Floor
24	21
28	28
40	38
55	51
10	-1

$\text{floor}(k) \leftarrow$

K → if K is present
= smaller than K → if K is not present

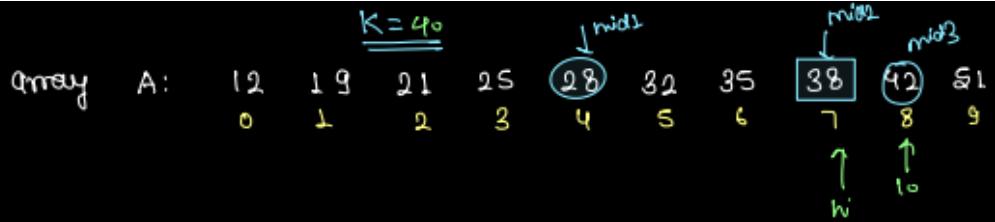
$K = 30$

Array	A:	12	19	21	25	28	32	35	38	42	51
		0	1	2	3	4	5	6	7	8	9



$l_0 = 0 \leq 5$
 $w = 6 \leq 4$ Noop Stop.
 $m_1 = 4 \neq 5$

ans = ~~4~~ 28 $\Rightarrow 28$ is op.



```
if ( arr[mid] == k) {
```

```
    ans = arr[mid];
```

```
    break;
```

```
} else if ( arr[mid] < k) {
```

```
// potential candidate → arr[mid]
```

```
    ans = arr[mid];
```

```
    lo = mid + 1;
```

```
} else {
```

```
    hi = mid - 1;
```

```
}
```

$lo = \cancel{18} 8$] loop stop.

$hi = \cancel{7} \neq 8$

$ans = \cancel{28} 38$] → ans:

TODO: ① Dry Run for floor

array A: 12 19 21 25 28 32 35 38 42 51

K=12

K=50

K=28

K=55

② Try to solve for cell value of K

cell value of K → min from all max.

```

int floorOfK (int[] A, int k) {
    int lo = 0;
    int hi = A.length - 1;
    int ans = -1;
    while (lo <= hi) {
        int mid = (lo + hi) / 2;
        if (A[mid] == k) {
            ans = A[mid];
            break;
        } else if (A[mid] > k) {
            //discard Right Region]
            hi = mid - 1;
        } else if (A[mid] < k) {
            ans = A[mid];
            lo = mid + 1;
        }
    }
    return ans;
}

```

T.C: $O(\log n)$

3

Local Minima

5. Local Minima

- * Given an array, find any local minima.
- * A local minima is defined as an element that is smaller than both of its neighboring elements.
- * Corner elements will have only one neighbor.
- * Array contains distinct elements.

* $\text{len} > 1$

A: 12 10 15 20 ans: 10

A: 12 10 9 7 15 ans: 7

A: 12 15 17 19 8 20 ans: 8 or 12

A: 10 7 3 2 1 ans: 1

Simple Idea:

go on every possible index and check if that element is smaller than both neighbors. that current element is

Answer:

→ $O(n)$

Expected time Comp = $O(\log n)$

→ local minima will always present in the array:

1. Increasing order

local Minima

2. Decreasing order

local min'ma.

3. Random Anagram

multiple local Min'ma.

$\text{arr}[\text{mid}-1] < \text{arr}[\text{mid}] \&$

$\text{arr}[\text{mid}] < \text{arr}[\text{mid}+1]$

$\rightarrow \text{arr}[\text{mid}] \Rightarrow \text{ans}$

this middle element itself
is answer.

$\text{arr}[\text{mid}-1] < \text{arr}[\text{mid}]$ For left

if $\text{arr}[\text{mid}] < \text{arr}[\text{mid}+1]$ 100% sure.

$\rightarrow \underline{\text{left side}}$

For Right side (May be there is ans)
But we are not 100% sure.

$\text{arr}[\text{mid}-1] > \text{arr}[\text{mid}]$

if $\text{arr}[\text{mid}] > \text{arr}[\text{mid}+1]$ For left side
(May ans. avl.)

$\rightarrow \underline{\text{Right side}}$

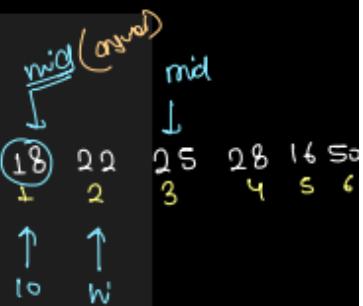
But not 100% sure

mid-1 mid mid+1

```

public static int localMinima(int[] A) {
    int n = A.length;
    // cover all corner cases & Edge cases
    // 1. First element of Array
    if(A[0] < A[1]) {
        return A[0];
    }
    // 2. Last element of Array
    if(A[n - 1] < A[n - 2]) {
        return A[n - 1];
    }
    int lo = 1;
    int hi = n - 2;
    while(lo <= hi) {
        int mid = (lo + hi) / 2;
        if(A[mid - 1] > A[mid] && A[mid] < A[mid + 1]) {
            // middle element is local minima
            return A[mid];
        }
        else if(A[mid - 1] < A[mid] && A[mid] < A[mid + 1]) {
            // left side have answer
            hi = mid - 1;
        }
        else if(A[mid - 1] > A[mid] && A[mid] > A[mid + 1]) {
            // right side have answer
            lo = mid + 1;
        }
    }
    // for satisfaction of java syntax
    // not necessary but requirement of syntax
    return -1;
}

```



T.C: $O(\log \frac{N}{2})$

DSA: Two Pointers - 22 - June 2023