

## TP Sessions-Bit Manipulation and Number Based Problems

### Count Setbits

```
n=12
int count=0
while(n>0)
{
    if(n%2==1)
    {
        count++
    }
    n=n/2
}
print count
```

### Better Approach

```
n=12
while(n>0)
{
    if((n&1)==1)
    {
        count++
    }
    n>>=1 //a=a>>1
}
```

### Optimized Approach

```
while (n>0)
{
    n &= (n - 1)
    count += 1
}
print count
```

### Count sum of set bits from 1 to n

```
int totalCount = 0
for (int i = 1; i <= n; i++) {
    int num = i;
    while (num > 0)
    {
        totalCount += (num & 1)
        num >>= 1
    }
}
```

### Count of Set bits for each number in the range upto N and Store it in array

-----  
for example: if N=5  
count of set bits from 0 to 5 in the array is:

[0,1,1,2,1,2]

```
int[] arr = new int[n + 1] //initialized array to zero
int offset = 1
```

```

        for (int i = 1; i <= n; i++) {
            if (offset * 2 == i) {
                offset = i
            }
            arr[i] = 1 + dp[i - offset]
        }
        print arr elements
    }

```

Swap 2 numbers using Bitwise Operators

-----

```

a = a ^ b
b = a ^ b
a = a ^ b

```

Find Unique element in the given array

```

arr=[2,3,4,2,3]
for num in arr:
    result ^= num
print result

```

Find Missing number in the array

-----

```

N = 6
a = [0,1,2,3,5]

```

```

xor1 = 0
xor2 = 0

```

```

for num in a:
    xor2 ^= num

```

```

for i in range(0, N):
    xor1 ^= i

```

```

print(xor1 ^ xor2)

```

Given x number of 1's followed by y number of 0's, your task is to find the decimal representation of the x and y.

```

int decimal = Math.pow(2, x) - 1
decimal = decimal * Math.pow(2, y)

```

Better approach

-----

```

int res=0
int decimal = (1 << x) - 1
decimal <<= y

```

if number is too large use big integer class

```

import java.math.BigInteger;
BigInteger x = scanner.nextBigInteger();
BigInteger y = scanner.nextBigInteger();

```

```

BigInteger res = BigInteger.ZERO;

```

```

        BigInteger res =
        BigInteger.ONE.shiftLeft(x.intValue()).subtract(BigInteger.ONE).shiftLeft(y.intValue());

```

Given Xth and Yth Bit position. Create a number where X and Yth bit are Set:

```

int result= (1<<x) | (1<<y)

```

check if a given number is power of 2 or not

```

if ((n > 0) and (n & (n - 1)) == 0)
    its power of 2

```

In a given integer - N, check whether the ith bit is set or not.

suppose i=2 and N=10, 1010 in binary: 2nd bit is 0(starting from 0,1,2)

```

if ((n >> i) & 1) then
    print true
else
    print false

```

Calculate number of bits required to represent an integer value:

```

while n>0 do
    count++
    n>>=1
    print count

```

instead: `int bits = (int)(Math.log10(num) / Math.log10(2)) + 1;`

For example Take num=4

this is equal to  $\log_{10}(4)/\log_{10}(2) \rightarrow \log_2(4)=2+1=3$

Reverse the Bits of an integer number and print the value in decimal.

```

while N>0 Do
    res=0
    res = (res << 1) | (n & 1);
    n >>= 1;
}
print res

```

N&1 to extract LSB bit of original number.

Res<<1 to shift the result bit to the left side so that we can reverse the original number.

OR operation to combine the shifted res with the extracted bit.

n>>=1 to move to the next bit in the number

Sum of 2 numbers using bitwise operators

```

int x=5, y=3
while y != 0 Do
    carry = x & y

```

```

        x = x ^ y
        y = carry << 1
    print(x)

```

- $x \& y$  calculates the carry bits.
- $x \wedge y$  calculates the sum bits without carry.
- `carry << 1` shifts the carry bit left
- Repeat until there's no carry left.

Largest Power of 3 less than or equal to given number N:

```

    initialize res to 1
    initialize power to 1
    until power < N Do
        res=power
        power =power * 3
    print(res)

```

If the number is too large use BigInteger Class

```

    String input = scanner.nextLine();
    BigInteger N = new BigInteger(input);
    // or read directly as: BigInteger x = scanner.nextBigInteger();

```

```

    BigInteger res = BigInteger.ONE;
    BigInteger power = BigInteger.ONE;

```

```

    while (power.compareTo(N) < 0) {
        res = power;
        power = power.multiply(BigInteger.valueOf(3));
    }

```

```

    print(res)

```

Print Number of trailing zeroes in the factorial of a given number N

GENERAL APPROACH

```

mainfunction()
{
    read long int value
    long fact=factorial(n)
    long count=counttrailingzeroes(fact)
    print(count)
}

```

```

factorial(long n)
{
    initialize long res=1
    for i=1 to n (i=1;i<=n;i++)
        res=res*i
    return res
}

```

```

counttrailingzeroes(long n)
{
    int count = 0;
    while n > 0 Do
        if n mod 10 == 0 then
            count++
        else
            break the loop
        n= n/10;
    END WHILE
    return count
}

```

### BETTER APPROACH

every multiple of 5 contributes to number of trailing zeroes in N!

use formula:  $n/5 + n/25 + n/125$  and so on!!

initialize count=0

```

while n > 0 Do
    count = count + (n/5)
    n = n/5
END WHILE

```

```

print(count)

```

Check if a given number is Prime Number or Not

```

int flag=1
if n<=1
    print not Prime
    return

for i=2 to Math.sqrt(n) DO
    if n mod i==0 then
        set flag to 0
        break the loop

if flag==1 then
    print "prime"
else
    print "not Prime"

```

Instead of using Math.sqrt the better efficient way could be:

```

for i=2 to i*i<=n
    if n mod i==0
        set flag=0
        break the loop

```

### Print Prime numbers from 1 to given range N

```
mainfunction()
{
    read n
    for i=1 to n+1 Do
        if(checkprime(i)):
            System.out.print(i+" ")
}

boolean checkprime(int n)
{
    if n is less than or equal to 1
        return False
    for i=2 to Math.sqrt(n) Do
        if n%i==0
            return false
    return true
}
```

### Print Prime Numbers upto given Count

```
read N
set count=0
set i=1
while count<n Do:
    res=checkprime(i)
    if res==true
        System.out.print(i+" ")
        increment count++
    increment i++
```

```
boolean checkprime(int n)
{
    if n is less than or equal to 1
        return False
    for i=2 to Math.sqrt(n) Do
        if n%i==0 then
            return false
    return true
}
```

### SIEVE OF ERATOSTHENES: THE MOST OPTIMIZED PRIME NUMBER LOGIC

```
boolean[] prime = new boolean[n + 1]
```

```
for int i=0 to n Do
    set prime values to true
```

```
OuterLoop for int i=2 to i*i<=n Do
    check if prime[i] is true if so then goto inner loop
        for int j=p*p to n (Note: increment innerloop by j=j+p)
            set prime[j] to false
```

```

for int i=2 to n Do
    if prime[i] is true then
        print(prime[i]+" ")

```

Print prime numbers upto given range N for T test cases each on new line

```

Take input number of test cases T
iterate from i=1 to T Do
    read integer number n
    call printupton(n) function

```

```

function printupton(n):
    iterate from i to n Do
        if(checkforprime(i)==true):
            print(i+" ")
    println()
End Function

```

```

function boolean checkforprime(int n)
    if n<=1 then
        return false
    iterate from i=2 to sqrt(n) D0
        if n%i==0 then
            return false
    return true
End function

```

**for example:**

```

T=3
10
2 3 5 7
20
2 3 5 7 11 13 17 19
100
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

```

compute a power b where b value can be as large as  $10^9$ .  
don't use inbuilt function pow.

Basic Approach

```

-----
Read a value (base)
Read b value (power)
long result = 1;
while (b != 0) {
    result = result * a;
    b--;
}

```

For large values of power ( $10^9$ ) this approach will take too long to compute. i.e  $10^9$  multiplications, which is computationally expensive and will take a significant amount of time to complete,  
Time Complexity is  $O(b)$ , which is not efficient for large  $b$ .

## Optimized Approach

---

**Efficient Reduction:** Instead of iterating  $b$  times, we reduce  $b$  exponentially by halving it, achieving  $O(\log b)$  time complexity.  
**Handles Large Powers:** The method works even for very large powers like  $10^9$  where direct multiplication would take too long.  
**Modulo Operation:** Keeps intermediate results within limits, preventing overflow.

formula:  $a^b$  if  $b$  is odd for example  $2^3$ : formula is:  $a^b = a * a^{(b-1)}$   
 if  $b$  is even:  $a^b = (a * a)^{b/2}$  for example:  $2^4 = (2 * 2)^{4/2} = 4^2 = 16$

```
int t = sc.nextInt(); // Number of test cases
int MOD = 1000000007;

while (t-- > 0) {
    Read a value
    Read b value
    declare long result=1
    assign a to long base i.e long base =a

    while (b > 0) {
        if (b % 2 == 1) {
            result = (result * base) % MOD;
        }
        base = (base * base) % MOD;
        b /= 2;
    }
    print(result)
}
```

can be solved using BigInteger class (but wont work if constraints are mentioned)

---

```
BigInteger mod = new BigInteger("1000000007");
int t=scanner.nextInt();
while(t-->0)
{
    BigInteger result = BigInteger.ONE;
    BigInteger a = scanner.nextBigInteger();
    BigInteger b = scanner.nextBigInteger();
    while (!b.equals(BigInteger.ZERO))
    {
        result = (result.multiply(a)).mod(mod);

        b = b.subtract(BigInteger.ONE);
    }

    System.out.println(result);
}
```



## Arrays 1D and 2D

Beautiful NUmber

-----

input=1223433444

output=1223433444 is a beautiful number

Function boolean check\_beautiful(int num)

```
    String str = String.valueOf(num)
    int[] count = new int[10]
    iterate from i=0 to i<str.length() Do
        int digit = str.charAt(i) - '0'
        count[digit]++
    end For
    iterate from i=0 to i<str.length() Do
        int digit = str.charAt(i) - '0'
        if (count[digit] != digit)
            return False
        End For
    end For

    return true
End Function
```

Find Duplicate Number using Array Count Approach

-----

input=5

2 1 4 3 2

output=2

```
Function int findDuplicate(int[] nums)
    int len = nums.length;
    int[] cnt = new int[len]
    For iterate from i=0 to i<len Do
        cnt[nums[i]]++
        if cnt[nums[i]] > 1
            return nums[i]
        End For
    end For
    return -1
End Function
```

Duplicate Number Using ArrayList

-----

```
Function int findDuplicate(int[] nums)
    ArrayList<Integer> arr = new ArrayList<>()
    for (int num : nums)
        if (arr.contains(num))
            return num
        else
            arr.add(num)
        }
    return -1
End Function
```

## Duplicate Number Using HashSet

---

```
Function int findDuplicate(int[] nums)
    HashSet<Integer> arr = new HashSet<>()
    for (int num : nums)
        if (!arr.add(num))
            return num
    return -1
End Function1
```

hashset takes  $O(1)$  for lookup  
whereas arraylist uses  $O(n)$

## Print Unique Elements in Array Basic Approach

---

input=5  
1 2 1 1 3  
output=  
2 3

```
findAndPrintUniqueElements(int[] array, int N)
    int[] frequency = new int[N]

    for (int i = 0; i < N; i++)
        for (int j = 0; j < N; j++)
            if (array[i] == array[j])
                frequency[i]++

    for (int i = 0; i < N; i++)
        if (frequency[i] == 1)
            print(array[i])
```

## Print Unique Elements in Array Optimized Approach

---

```
HashSet<Integer> unique = new HashSet<>()
HashSet<Integer> duplicate = new HashSet<>()
for (int num : array)
    if (!unique.add(num))
        duplicate.add(num)

unique.removeAll(duplicate)
for (int i : unique)
    print(i)
```

## Count Pairs with given Sum K Basic Approach

---

input=  
7  
4 3 2 6 5 1 3  
6  
output=  
Number of pairs with sum 6: 3

```

int countPairs(int[] nums, int K)
    int count = 0
    int n = nums.length

    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (nums[i] + nums[j] == K)
                count++
        }
    }
}
print(count)

```

### Count Pairs with Sum K using Compliment approach

---

```

int countPairs(int[] nums, int K)
    int count = 0
    int[] res = new int[10000] //assume max value upto 10000
    for (int num : nums)
        int complement = target - num
        count += res[complement]
        res[num]++

    print count

```

### Use ArrayList approach for space complexity

---

```

List<Integer> res = new ArrayList<>()
int count=0
for (int num : nums)
    int complement = K - num;
    count += Collections.frequency(res, complement)
    res.add(num)

```

### Using HashMap to improve time and space complexity

---

```

Map<Integer, Integer> res = new HashMap<>()
for (int num : nums)
    int complement = K - num
    if (res.containsKey(complement))
        count += res.get(complement)
    res.put(num, res.getOrDefault(num, 0) + 1);
End For

```

### Kth Smallest Element in an Array Basic Approach

---

```

3 4 66
34 5 7
8 77 22
Enter the value of k:
3
[3, 4, 66, 34, 5, 7, 8, 77, 22]
The 3rd smallest element is: 5

```

```

int rows = matrix.length
int cols = matrix[0].length
int[] flatArray = new int[rows * cols]
int index = 0
for (int i = 0; i < rows; i++)
    for (int j = 0; j < cols; j++)
        flatArray[index++] = matrix[i][j]

Arrays.sort(flatArray)
print (flatArray[k - 1])

```

### Kth Smallest Element in an Array using Priority Queue

---

```

PriorityQueue<Integer> minHeap = new PriorityQueue<>()
for (int i = 0; i < rows; i++)
    for (int j = 0; j < cols; j++)
        minHeap.offer(matrix[i][j]);

int k=sc.nextInt()

int kthSmallest = -1
for (int i = 0; i < k; i++)
    kthSmallest = minHeap.poll()

print kthSmallest

```

### Transpose Matrix

---

```

{1, 2}
{4, 5}  3*2 matrix
{7, 8}
after transpose becomes 2*3 matrix
{1,4,7}
{2,5,8}

```

```

int[][] transpose1(int[][] matrix)
{
    int rows = matrix.length;
    int cols = matrix[0].length;
    int[][] transpose = new int[cols][rows];

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            transpose[j][i] = matrix[i][j];
        }
    }

    public static void printMatrix(int[][] matrix)
    {
        for (int[] row : matrix)
            for (int value : row)
                System.out.print(value + " ")
            println()
    }
}

```

## Zero Matrix

```
-----
input=
Enter the size of the square matrix: 4
Enter matrix elements (0s and 1s):
0 1 0 1
0 1 0 1
1 1 1 1
1 1 1 1
output=
Modified Matrix:
0 0 0 0
0 0 0 0
0 1 0 1
0 1 0 1
```

```
setZeroMatrix(int[][] matrix, int n)
    boolean[] row = new boolean[n]
    boolean[] col = new boolean[n]

    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (matrix[i][j] == 0)
                row[i] = true
                col[j] = true

    for (int i = 0; i < n; i++)
        if (row[i])
            for (int j = 0; j < n; j++)
                matrix[i][j] = 0

    for (int j = 0; j < n; j++)
        if (col[j])
            for (int i = 0; i < n; i++)
                matrix[i][j] = 0
```

## MATRIX MULTIPLICATION

-----

For matrix multiplication, the number of columns in the first matrix must be equal to the number of rows in the second matrix. The resulting matrix, longstreakehas the number of rows of the first and the number of columns of the second matrix.

```
int[][] multiply(int[][] a, int[][] b)
    if (a[0].length != b.length)
        print "Cant Multiply bcz Matrix A columns must match Matrix B"
        EXIT

    int rowsA = a.length
    int colsA = a[0].length
    int rowsB = b.length
    int colsB = b[0].length
    int[][] result = new int[rowsA][colsB]
```

```
for (int i = 0; i < rowsA; i++)  
    for (int j = 0; j < colsB; j++)  
        result[i][j] = 0  
        for (int k = 0; k < colsA; k++)  
            result[i][j] += a[i][k] * b[k][j]
```

## Big Factorial

Constraints:

-----  
Input length: within integer range  
Output length:  $1 \leq \text{length}(n) < 1000$

Test Case examples:

-----  
input=45  
output=  
Factorial of 45 is:  
119622220865480194561963161495657715064383733760000000000

input=124  
output=  
Factorial of 124 is:  
1506141741511140879795014161993280686076322918971939407100785852066825250652908790935063463  
1159673850691712435674404619250412953547310447825510676604683764441946110045200570541670400  
00000000000000000000000000000000

input= -5853  
output=  
Factorial is not defined for negative numbers.

input= !@#\$%^&\*)fgfg(\*&+\_  
output=  
Special Characters and Alphabets are not allowed.

```
*/      try{
        String input = scanner.nextLine().trim();
        int number = Integer.parseInt(input);

        if (number < 0)
            throw new GrietException("Factorial is not defined for negative numbers.")

        if (!input.matches("\\d+"))
            print "Special Characters and Alphabets are not allowed."
            return;

        BigInteger factorial = calculateFactorial(number)
        print "factorial"

    } catch (GrietException e) {
        print(e.getMessage())
    } catch (NumberFormatException e) {
        print("Special Characters and Alphabets are not allowed.")
    }
}

private static BigInteger calculateFactorial(int n) {
    BigInteger result = BigInteger.ONE
    for (int i = 2; i <= n; i++) Do
        result = result.multiply(BigInteger.valueOf(i))
    end for
    return result
}
```

## Welcome Back Armstrong

An Armstrong number: is a number that is the sum of its own digits each raised to the power of the number of digits.

Input/Output Constraint:

-----  
 $1 \leq N < 10^{17}$

Output format: return true if its armstrong number, return false if its not an armstrong number.  
Don't print any output message in ur code.

Test Case Examples:

-----  
input= 153  
output=  
153 is an Armstrong number.

Explanation:  $1^3 + 5^3 + 3^3 = 153$  Hence its an Armstrong number.

input= 9474  
output=  
9474 is an Armstrong number.

Explanation:  $9^4 + 4^4 + 7^4 + 4^4 = 9474$  hence its an Armstrong number.

input= 24678050  
output= 24678050 is an Armstrong number.

input=35641594208964132  
output=35641594208964132 is an Armstrong number.

input=4929273885928088826  
output=4929273885928088826 is not an Armstrong number.

\*/  
// Start writing your code from here

```
public static boolean Met(String numStr)
    int length = numStr.length()
    long sum = 0

    for (int i = 0; i < length; i++) Do
        int digit = Character.getNumericValue(numStr.charAt(i));
        sum += Math.pow(digit, length);
    END for

    return sum == Long.parseLong(numStr);
```

In Python u can take a limit variable assigned to  $10^{17}$ , at any point if sum exceeds limit value u can return false or else return true.



## Next Smallest Palindrome

$1 \leq \text{number} \leq 10^{17}$

Test Case Example:

-----

input=8496395839536

output=

Next Smallest Palindrome: 8496395936948

input=1268

output=

Next Smallest Palindrome: 1331

input=9775577457

output=

Next Smallest Palindrome: 9775665779

```
    long num = scanner.nextLong()  
    long res = PalindromeChecker(num)  
  
    print res as next Smallest Palindrome  
}
```

```
private static long PalindromeChecker(long num)  
    while (true) Do  
        num++  
        if (isPalindrome(num))  
            return num  
    end While  
Close function
```

```
private static boolean isPalindrome(long num)  
    long originalNum = num  
    long rev = 0
```

```
    while (num > 0) Do  
        long digit = num % 10  
        rev = rev * 10 + digit
```

```
        num /= 10
    end while
```

```
    return originalNum == rev
End Function
```

Universal Palindrome

```
input=1221
output=palindrome
```

```
input=ar77ra
output=palindrome
```

```
input=*&*
ouptut=palindrome
```

```
public static boolean isPalindrome(String str) {
    String rev = new StringBuilder(str).reverse().toString();
    return str.equals(rev);
}
```

## Big Digit Sum

## Big Digit Sum

Constraints:

```
input= 1 <= length(N) <= 10^3
output= result should be within integer range.
```

Note:

Print appropriate error messages if user enters any other characters apart from digits in the input. As given in the test cases below. Remove if there are any extra whitespaces in input.

test case examples:

```
input=1234
output=
10
```

```
input=56787654324567876543456765434567654345676543
output=
228
```

[illegible]

```
input=      Griet1661
output=
Only digits are allowed
```

input=\$6\*(((6!#\$+\_)))  
Special characters are not allowed

```
String input = scanner.nextLine().trim()
    sumOfDigits(input)
```

```
public static void sumOfDigits(String N)
    boolean hasAlphabet = false
    boolean hasSpecialChar = false
```

```
for (char ch : N.toCharArray())
    if (Character.isLetter(ch))
        hasAlphabet = true
        break;
    else if (!Character.isDigit(ch))
        hasSpecialChar = true
        break

if (hasAlphabet)
    print "Only digits are allowed"
    return;
else if (hasSpecialChar)
    print "Special characters are not allowed"
    return
```

```
int digitSum = 0
for (char digit : N.toCharArray())
    digitSum += Character.getNumericValue(digit)
print digitSum
```

## Sum Between the largest and second largest in the arraylist

```
input=2 1 6 7 8 9 5 4 10
quit
output=
Sum between the largest and second-largest numbers: 9
```

STEPS:

-----

```
INITIALIZE ArrayList named as numbers
```

```
WHILE (sc.hasNextInt())
    READ number and ADD to numbers
```

```
IF (numbers.size() < 2)
    PRINT "At least two distinct numbers are required."
    RETURN
```

```
INITIALIZE Bigindex = 0
INITIALIZE secondBig index= -1
```

```
Iterate from i=1 to i less than numbers.size() Do
```

```
    IF (numbers[i] > numbers[Bigindex])
        secondBigindex = Bigindex
        Bigindex = i
    ELSE IF (secondBigindex == -1 OR numbers[i] > numbers[secondBigindex])
        secondBigindex = i
```

```
INITIALIZE sum = 0
k = MIN(Big, secondBig)
```

```
Iterate from j=k+1 TO k < (Big + secondBig - k)
    sum = sum + numbers[j]
```

```
PRINT "Sum between the largest and second-largest numbers: " + sum
```

## Harmony in Array

input=5  
3 2 5 3 2  
output=  
Harmony Index: 2

```
int harmonyindex(ArrayList<Integer> arr, int n)
    int totalSum = 0
    int leftSum = 0

    for (int i = 0; i < n; i++)
        totalSum += arr.get(i)

    for (int i = 0; i < n; i++)
        totalSum -= arr.get(i)

    if (leftSum == totalSum)
        return i

    leftSum += arr.get(i)
}

return -1
}
```

## Unique Pairs and Count in 2D array of Strings

```
input=5
virat kohli
rohit sharma
ishan kishan
virat kohli
KL rahul
output=
1
2
3
3
4
```

```
int n = scanner.nextInt()
scanner.nextLine()
```

```
String[][] input = new String[n][2]
```

```
for (int i = 0; i < n; i++)
    String line = scanner.nextLine()
    input[i] = line.split(" ")
```

```
uniquePairs(input)
```

```
void uniquePairs(String[][] pairs)
```

```
    HashSet<String> set = new HashSet<>()
    int count = 0
```

```
    for (String[] i : pairs)
        String merged = i[0] + " " + i[1]
```

```
        if (set.contains(merged))
            print(count)
```

```
        else
```

```
            set.add(merged)
            count++;
            print(count)
```

## Winning Candidate

input=5  
3 1 3 3 2  
output=  
Winning Candidate: 3

STEPS:

-----

```
winningCandidate(ArrayList<Integer> list)
```

```
    int count = 0
```

```
    int candidate = -1
```

```
    for (int num : list)
```

```
        if (count == 0)
```

```
            candidate = num
```

```
        if(num==candidate)
```

```
            count++
```

```
        else count--
```

```
    int finalCount = 0
```

```
    for (int num : list)
```

```
        if (num == candidate)
```

```
            finalCount++
```

```
    if (finalCount > list.size() / 2)
```

```
        return candidate
```

```
    return -1
```

## Longest Consecutive Subsequence

```
input=2 1 0 3 quit
output=
Length of the longest consecutive subsequence: 4
```

```
input=44 45 2 6 47 90 48 12 56 49 100 50
quit
output=
Length of the longest consecutive subsequence: 4
```

Explanation: 47,48,49,50 is the longest consecutive sequence hence output is 4.

STEPS:

-----

```
Function Subsequence(HashSet<Integer> arr)
    int longstreak = 0

    for (int num : numberSet) {
        if (!numberSet.contains(num - 1))
            int curnum = num
            int curstreak = 1

            while (numberSet.contains(curnum + 1))
                curnum++
                curstreak++

            longstreak = Math.max( longstreak, streak)

    }

    print(longstreak)
```



## Searching, sorting

### Array, Key, Segment, Search

-----

```
input=
9
3 6 2 2 5 7 4 5 2
3 //key
3 //segment
output=No Key not found in every segment
```

Explanation: segment size is 3 so. segments are {3,6,2}, {2,5,7} and {4,5,2}  
key value is 3 which is present in every segment.

```
input=
9
6 7 2 5 2 9 4 3 2
2 //key
3 //segment
output=Yes the key found in every segment
```

```
Function int KeyPresentInSegments(int arr[], int n, int x, int k)
    for (int i = 0; i < n; i += k)
        int found = 0
        for (int j = i; j < i + k && j < n; j++)
            if (arr[j] == x)
                found = 1
                break

        if (found == 0)
            return 0

    return 1                //i.e key present in every segment
```

### Smaller < X < Greater

-----

```
input=10
2 1 3 4 5 7 12 45 78 10
output=
Index of the element: 2

input=5
10 2 2 3 1 0
output=No such element found.
```

```
Function int findElement(int[] arr, int n)
    for (int i = 1; i < n - 1; i++)
        if (check(arr, n, i) == 1)
            return i

    return -1

Function int check(int[] arr, int n, int ind)
    int i = ind - 1
    int j = ind + 1

    while (i >= 0)
        if (arr[i] > arr[ind])
```

```

        return 0
    i--

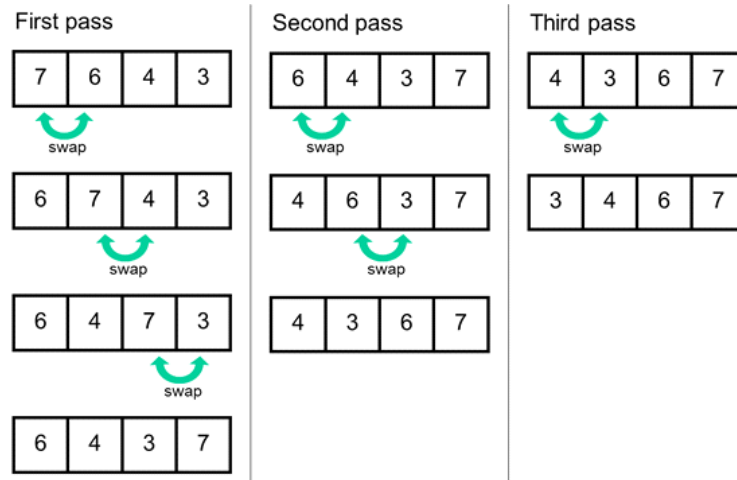
    while (j < n)
        if (arr[j] < arr[ind])
            return 0
        j++

    return 1

```

### Efficient Bubble Sort

---



```

input=7
34 21 67 45 123 43 89
output=
Pass 1: 21 34 45 67 43 89 123
Pass 2: 21 34 45 43 67 89 123
Pass 3: 21 34 43 45 67 89 123
Sorted array is:
21 34 43 45 67 89 123

```

```

for i from 0 to n-1
    flag = false

    for j from 0 to n-i-1
        if arr[j] > arr[j+1]
            swap arr[j] and arr[j+1]
            flag = true

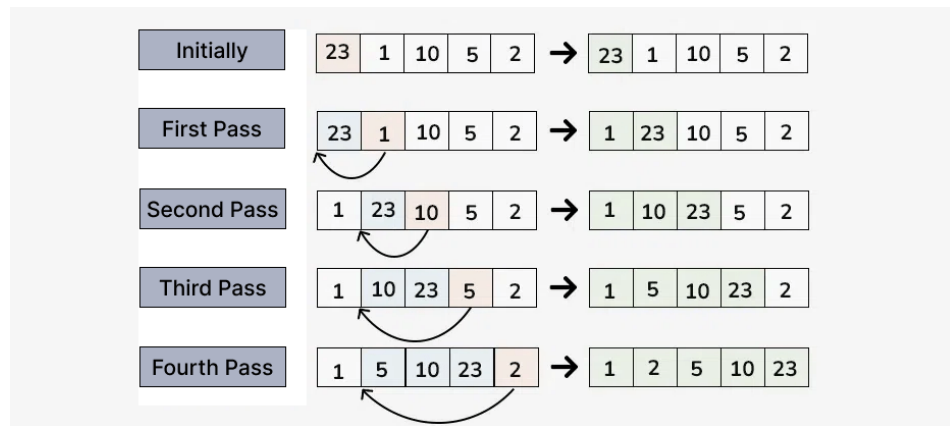
    if flag is false
        break from the outerloop
    else
        for j from 0 to n:
            print pass output arr[j]

    back to main loop

```

### Insertion Sort with Passes

---



```

input = 5
12 78 45 2 18
output =
PASS - 1 :12 78 45 2 18
PASS - 2 :12 45 78 2 18
PASS - 3 :2 12 45 78 18
PASS - 4 :2 12 18 45 78

```

THE SORTED LIST:  
2 12 18 45 78

```

for (int i = 1; i < n; i++)
    int key = arr[i]
    int j = i - 1
    while (j >= 0 && arr[j] > key) {
        arr[j + 1] = arr[j]
        j--
    }
    arr[j + 1] = key
    printArray(arr, n, i)
}

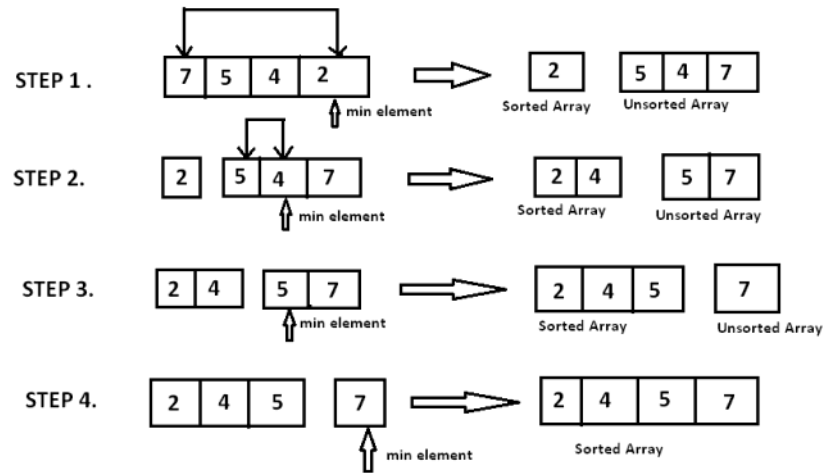
printArray(int[] arr, int n, int pass)
    for (int k = 0; k < n; k++)
        System.out.print(arr[k] + " ")

System.out.println()

```

Selection Sort

-----



```

for (int i = 0; i < n - 1; i++)
    int minIndex = i
    for (int j = i + 1; j < n; j++)
        if (array[j] < array[minIndex])
            minIndex = j

    int temp = array[minIndex]
    array[minIndex] = array[i]
    array[i] = temp;

    print("Array after pass " + (i + 1) + ": ")
    for (int k = 0; k < n; k++)
        System.out.print(array[k] + " ");

end for

print final array

```

Quick Sort with last element as Pivot

```

input=4
5 4 3 1
output=
1 4 3 5
4 3 5
3 4
1 3 4 5

```

```

public static void quicksort(int[] a, int left, int right) //left=0, right=n-1
    if (left < right)
        int pivot = partition(a, left, right)
        quicksort(a, left, pivot - 1)
        quicksort(a, pivot + 1, right)

```

```

public static int partition(int[] a, int left, int right)
    int pivot = a[right]

```

```

    int i = left - 1 //i track the position of the last element that is less than or equal to the pivot
    for (int j = left; j < right; j++)
        if (a[j] <= pivot)
            i++ //This means that all elements up to index i are guaranteed to be less than or equal to the pivot.
            swap(a, i, j)

    swap(a, i + 1, right) //adjust the pivot
    // When we swap the pivot with the element at index i + 1, we are effectively placing the pivot right
    after the last element that is less than or equal to it.

    printArray(a, left, right) // Print the array after partitioning
    return i + 1
}

void swap(int[] a, int i, int j)
{
    int temp = a[i]
    a[i] = a[j]
    a[j] = temp
}

void printArray(int[] a, int left, int right)
{
    for (int i = left; i <= right; i++)
        System.out.print(a[i] + " ")
    }
    System.out.println()
}
}

```