Data Stoucturses
And Algosithms
(-Java)

Printing in Java:

System.out.print(" ");

for cursor moving to the next line System.out.println(" ");

Datatypes:

int, long → integer -[1, 3 ...]

double, float → decimal -[24.03]

char -['a', 'A']

boolean-[true | false]

Conditionals:

```
if (condition){
}
```

```
if (a>b)
    System.out.println("a is large");
else
    System.out.println("b is large");
```

If else → mutually exclusive

Operators:

Arithmetic : +, -, *, /, %
(/ quotient, % modulo [remainder])

Comparision : ==, !=

Logical : &&, ||

Input:

```
Scanner sc = new Scanner(System.in);
Eg: int x = sc.nextInt();
```

Note: java.util.* package must be imported in order to use Scanner.

Loops:

① while (condition)
{
}
increment

② for (int i=0; condition; i++)
{
}

Pre Increment - ++a
Past Increment - a++

i=10 present future

Postinc 10 11
Preinc 11 11

Eg:
```
int i = 10;
if (i++ == i)
    print (i is good)
else
    print (i is bald)
```
o/p: 11 is bad

```
int i=10;
if (++i == i)
    print (i is good)
else
    print (i is bad)
```
o/p: 11 is good

Note: Don't use pre/post increments in Recursion.

Getting Started:

1. **Print Z**

Input: There is no Input

Output:

```
* * * * *
    *
  *
*
* * * * *
```

[size-5]

Pseudo Code: [Steps]

1. As the required size is '5', we need to have println statements
2. In each println statement, print the stars at required positions.

Program:

```
public class Main {
    public static void main(String args[]) {
        System.out.println("*****");
        System.out.println("    *");
        System.out.println("  *");
        System.out.println("*");
        System.out.println("*****");
    }
}
```

2. Grading System

Input: 92 [marks]

Output: Excellent

Steps:

1. Take marks as input [integer]
2. Use else-if ladder to print the grade.

Program:

```
int marks = sc.nextInt();
if (marks > 90)
        print("excellent");
else if(marks > 80)
        print("  ")
else if (marks > 70)
        print(" ");
else
        print(" ");
```

3. Is A Number Prime

Input: 2 —— no of tests
11
6 } —— Numbers

Output: prime
not prime

Constraints: $1 <= t <= 10000$

$2 <= n <= 10^9$

Steps:

1. Take input as test cases.
2. Get the number 'n' through input.

3. Traverse from 2 to $\sqrt{n}$ and check whether any number between 2 to $\sqrt{n}$ is a factor to n;

4. If we get a factor, we can conclude the number is not prime, otherwise number is prime.

Why up to $\sqrt{n}$;

Consider n is not prime

$n = a * b$ (a and b are factors)

Now a and b can't be both greater than the square root of n, since then the product of $a*b$ would be greater than $sqrt(n) * sqrt(n) = n$

So, in any factorization of n, atleast one of the factors must be smaller than the square root of n, and if we can't find any factor less than or equal to the square root, n must be prime.

Program:

```
int t = sc.nextInt();          // no of test cases
for (int i = 0; i < t; i++)
{
    int n = sc.nextInt();      // n-inputNo
    if (isPrime(n))            // passing n to a function to check prime or not
        print (Prime)
    else
        not Prime
}
```

```
Public static boolean isPrime(int n){
    for(int i=2; i*i<=n; i++){        // iterating from 2 to √n
        if(n%i==0)                    // if factor found return false
            return false;
    }
    return true;                      // if no factor found return true
}
```

4. Print All Primes Till N

Input: 6 24 } Both low and high should be included
Output: 7
11
13
17
19
23

Constraints: $2 <= low < high < 10^6$

Steps:

1. Take two inputs low and High
2. Traverse from low to high
3. For each number in the range, check whether number is prime or not.
4. If number is prime print the number.

Program:

```
int low=sc.nextInt();
int high=sc.nextInt();
for(int i=low; i<=high; i++){
    if(isPrime(i))
        print(i);
}
```

-> isPrime() is same as 3rd ⇒ check for the Reference.

5. Print Fibonacci Numbers Till N

Input: 5 [n]

Output: 0
1
1
2
3

Constraints: 1 <= n <= 40

Steps:

1. Take input number n
2. Take two variables a=0, b=1
3. Print the values
4. From i=2 ⇒ n, peoform the addition of a and b and print the result
5. Now, Reassign the values a=b and b=c

0+1=1

0 1 1 ⇒ Then a=1 [After Reassining]
a b c b=1
⇒ c=2 ⇒ 0 1 1 2
a b c

Program:

```
int n = sc.nextInt();
int a=0, b=1;
int c;
  println(a)
  println(b)
for(int i=2; i<n; i++){
    c = a+b;          // Adding previous two terms
    println(c);
    a=b;              // Modifying a and b values
    b=c;
}
```

6. Count Digits in a Number;

Input; 34256 | (n)
Output; 6

Constraints; 1 <= n < 10^9

Steps;

Method-1

1. Intialize count -> 0
2. Take a loop with condition n != 0
3. Increase the count, once it enters while loop.
4. Decrement / Modify the n with n/10.

Method-2

1. Convert Integer to string using Integer.tostring
2. Return the length of string.

Method Program

```
int n = sc.nextInt();
int count = 0;
while(n != 0)
  {
  count++;
  n = n/10;
  }
return count;
```

Program

```
int n = sc.nextInt();
String a = Integer.toString(n);
return a.length();
```

7. Digits of a Number;

Input; 123
Output; 1
2
3

Constraints; 1 <= n < 10^9

Steps:

1. Take the input number
2. Find the length of the number
3. Calculate the divisor by the 10 power of n-1
4. Print the quotient
5. Assign the Remainder to n
6. Decrement the div by 10.
7. Run the entire steps till div > 0

Dry Run:

n = 123
div = 10^2 = 100
div > 0 ✓
q = 123/100
q = 1 - print
n = 123 - 1 * 100 = 23
div = 10
div > 0 ✓
q = 23/10 = 2 - print
n = 23 - 2 * 10 = 3
div = 10/10 = 1
div > 0 ✓
q = 3/1 = 3 - print
n = 3 - 3 * 1 = 3
div = 1/10 = 0
div > 0 ✗

Program:

```
int n = sc.nextInt();
String a = Integer.toString(n);
int leg = a.length();
int div = (int) Math.pow(10, leg-1);
while (div > 0){
    int q = n/div;
    print(q);
    n = n - q * div;
    div /= 10;
}
```

8. Reverse Of a Number:

Input: 123

Output: 3
2
1

Constraints: $1 <= n < 10^9$

Steps:

1. Take the input number.
2. Calculate the remainder by n%10
3. Print remainder
4. modify the number with n=n/10.
5. Continue all the steps till n>0.

Dry Run

n=123

>0

r=123%10=3

Print-3

n=n/10=123/10=12

>0

r=12%10=2

print-2

n=n/10=12/10=1

>0

r=1%10=1

print-1

n=1/10=0. x

Program

```
int n=sc.nextInt();
while(n>0)
{int r=n%10;
 Print(r);
 n=n/10;
}
```

9. Inverse of a Number:

Input: 6 2 4 1 3 5

Output: 6 1 4 2 5 3

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

Put 1 in 5th position

Constraints:

n is 5 digit long, it should contain all digits from 1 to 5 without repeating any digit from 1 to 5.

$1 <= n <= 10^8$

Steps:

1. Take the input number and ans=0, i=1
2. Get the last digit of a number.
3. Now multiply the ~~last digit~~ i with power of 10 to remainder -1
4. Increament i and modify n to n=n/10
5. continue till n!=0.

Program:

```
int n=sc.nextInt();
int r=0;
int i=1;
int ans=0;
while(n!=0){
    r=n%10;
  ans+=i*(int)Math.pow(10,r-1);
   i++;
   n=n/10;
   }
   print(ans);
```

Dry Run: i=1, ans=0

n=624135, !=0

r=5 [n%10]

ans=0+1×10^(5-1) = 10000, i=2

n=62413, !=0

r=3

ans=10000+2×10^2 = 10,200, i=3

n=624, !=0

r=1

ans=10,200+3×10^0 = 10,203, i=4

n=624, !=0

r=4

ans=10,203+4×10^3 = 14203, i=5

n=62, !=0

r=2

ans=~~10063+5×10^5~~ 14203+5×10^1 = 14,253, i=6

n=6, !=0

r=6

ans=14,253+6×10^5 = 614253, i=7

n=0 !=0 ✓

10. Rotate a Number

Input: n = 562984
k = 2

Output: 845629

Constraints:

$1 <= n < 10^9$

$-10^9 < k < 10^9$

562984 k=2

845629

562984 k=-2

298456

Steps:

1. First take the inputs n and k.
2. For ~~k~~ Eg: consider n=123, k=1 → 312
   ~~then~~ , k=4 → 312
   so, the cycle repeats

   so, modify k = k % (number length)
3. First we concentrate for k=+ve - Right Rotation

   5 6 2 9 (84) k=2

   So, in order to get 84, we divide the number with 10^k

   $r = 562984 \% 10^2 = 84$

   $q = 562984 / 10^2 = 5629$

   so, we need to multiply the r with 10^{leng-k} and add the q

   $84 \times 10^{6-2} = 840000$

   $q = 5629$

   845629
4. For k=-ve, left Rotation

n = 5 6 2 9 8 4 , k = -2

Instead of doing the same thing, we can do achieve the left Rotation with Right Rotation

k = -2 ⇒ 5 6 2 9 8 4

k = 4 ⇒ 2 9 8 4 5 6

-2 + 6 = 4 (Right Rotation)

So, leftRotation = Numberlength + k

k = numberlength + k [k = 6 - 2 = 4, -2 == 4]

Program:

```
int n = sc.nextInt();
int k = sc.nextInt();
String a = Integer.toString(n);
int leng = a.length();          // length of number
k = k % leng;                   // For cycle repetition
if(k < 0){                      // For handling left rotation and expressing in terms of right rotation
    k = k + leng;
}
int div = (int)Math.pow(10,k)   // For Divisor
int q = n / div;                // quotient
int r = n % div;                // remainder
int mult = (int)Math.pow(10,leng-k);  // Multiplication factor to multiply with r
int ans = mult*r + q;
System.out.println(ans);
```

11. Pythagorean Triplet:

Input 5 3 4 [a b c] | Pythagorean form: $c^2 = a^2 + b^2$

Output: true

Constraints: $1 <= a <= 10^9$
$1 <= b <= 10^9$
$1 <= c <= 10^9$

Steps:

1. Take the three numbers as input.
2. Calculate $a^2 = b^2 + c^2$ ~~and~~ if yes print true else
   $b^2 = c^2 + a^2$ if yes print true else
   $c^2 = a^2 + b^2$ if yes print true else false

Program:

```
int a = sc.nextInt();
int b = sc.nextInt();
int c = sc.nextInt();
if (axa == bxb+cxc || bxb == cxc+axa || cxc == axa+bxb)
        print(true)
else
        print(false)
```

12. The Curious Case of Benjamin Bulbs

Input: 6 [no of bulbs | fluctuations]
Output: 1
4 [no of bulbs toggled on]

Constraints: $2 <= n <= 10^9$

Steps:

1. Take the input number n.
2. Assume number as 6.
   - Initially all are off
     1st toggle on
     2nd every 2nd bulb toggles (either on/off) [If on → off, off → on]
     3rd fluctuation every 3rd bulb toggles

Bulbs	Fluctuations 1	2	3	4	5	6
1	(on)					
2	on	off				
3	on		off			
4	on	off		(on)		
5	on				off	
6	on	off	on			off

After all fluctuations, we observed only 1 and 4 bulbs are on.

Bulbs 1 and 4 have odd number of factors
(1) (1, 2, 4)

→ So only Bulbs with odd numbers of fluctuation will be turned on,

→ Numbers whose number of factors are odd as same as perfect squares.

→ Find perfect squares from 1 to √n.

Why √n: Numbers having perfect squares less than n will always be less than √n. Eg: for 6: [1, 4] For √6: we get approx 2....
1 to 2 → we get 1 and 4

[1×1 = 1, 2×2 = 4]

Program:

```
int n = sc.nextInt();                    // No of bulbs/ fluctuations
for(int i=1; i<=Math.sqrt(n); i++)       // ~~bulb~~ i to √n fluctuation traversing
{
    System.out.println(i*i);
}
```

13. GCD and LCM:

Input: 36 24 [a b]

Output: 12 [gcd]
72 [lcm]

Constraints: $0 < a <= 10^9$
$0 < b <= 10^9$

Steps:

1. We can calculate L.C.M by calculating G.C.D

~~LCM = GCD = t.~~
n

$$LCM = \frac{n1 \times n2}{GCD}$$

2. So first we will calculate GCD

Assume n1 = 12
n2 = 54

12) 54 (4
48
6) 12 (2
12
0

GCD

3. Take divisor = 12
dividend = 54

4. Perform division, then assign divisor to dividend and remainder to ~~divisor~~.

5. Continue the entire steps till
dividend % div != 0

6. we get div = gcd

7. Now calculate lcm = $\frac{n1 \times n2}{gcd}$

Dry Run;

n1 = 36
n = 24

divisor = 24
dividend = 36

36 % 24 != 0

r = 12
dividend = 24
divisor = 12

24 % 12 != 0 [36 % 12 == 0]
↓ break
~~r = 2~~
~~dividend = 8~~
~~divisor = 2~~

gcd = 12

lcm = $\frac{n1 \times n2}{gcd} = \frac{36 \times 24}{12} = 72$

Program;

```
int n1 = sc.nextInt();
int n2 = sc.nextInt();
int dividend = n1;
int divisor = n2;
while (dividend % divisor != 0)
{ int r = dividend % divisor;
  dividend = divisor;
  divisor = r;
}
```

println(divisor) // gives the gcd

int lcm = (n1 x n2)/divisor; // Gives the lcm

println(lcm);

14. Prime Factorization of a Number

Input: 1440

Output: 2 2 2 2 2 3 3 5

Constraints: 2 <= n <= 10^9

Steps:

1. Take the input Number
2. Traverse from i = 2 to √n
3. Divide the number with i
4. Print i
5. Continue till n%i == 0. [Step 3 to 5]
6. Check at last n is equal to 1 or not
7. If n is not equal to 1 then print n.

Why √n and checking n != 1

Factors of a number will always be less than √n and only one factor will be present greater than √n [atmost].

Dry Run:

① $n = 43$ $\sqrt{n} = 6$ appox greater

$i = 2$ to $\sqrt{n}$: 2 to 6

n%i

43 % 2 != 0 ✓

n = n/i , n = 23

print(2)

23 % 2 != 0 ✗

i++, i = 3

23 % 3 != 0 ✗

i++, i = 4

23 % 4 != 0 ✗

i++, i = 5

23 % 5 != 0 ✗

i++, i = 6

23 % 6 != 0

n != 1

23 != 1

print(23)

② $n = 36$, $\sqrt{n} = 6$

$i = 2$ to $\sqrt{n} = 6$

36 % 2 == 0

print(2)

n = 18

18 % 2 == 0

print(2)

n = 9

9 % 2 != 0

i++

9 % 3 == 0

print(3)

n = 9/3 = 3

3 % 3 == 0

print(3)

n = 1

~~if (n != 1), 0 !=~~

1 % 3 != 0

i++

1 % 4 != 0

i++

1 % 5 != 0

i++

1 % 6 != 0

↳ n = 1, skips the last if

Program:

```
int n = sc.nextInt();
for (int i = 2; i <= Math.sqrt(n); i++) {
    while (n % i == 0) {
        n = n / i;
        print (i);
    }
}
if (n != 1)
    print(n);
```

// If there is only one prime factor greater than √n (atmost)

Patterns

1. Input: 2 (n)

Output:
```
*
* *
```

Constraints: $1 <= n <= 100$

```
   3
   *
   * *
   * * *
```

Steps:

1. Number n represents the number of rows. So, i=1 to n [Take loop]
2. For each row, print the corresponding number of stars with tab space
3. After finishing each row move the cursor to next line by using println.

Program:

```
for(int i=1; i<=n; i++){          // rows traversing
    for(int j=1; j<=i; j++) {     // traversing through corresponding no of row times
        print("*\t");
    }
    println();                    // Move cursor to next line
}
```

2.

Input: 5 (n)

Output:

```
* * * * *
* * * *
* * *
* *
*
```

Constraints: $1 < n <= 100$

Steps:

1. Take the input number
2. Traverse from i=n to 0 for rows
3. Traverse from j=i to 0 and print stars with tab space till j loop ends.
4. After completing the row, move to next line

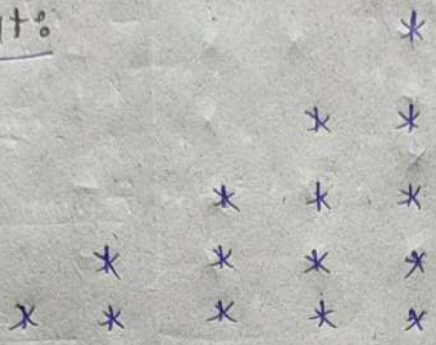
Program:

```
for (int i=n; i>0; i--) {
    for (int j=i; j>0; j--) {
        print("* ");
    }
    println();
}
```

3.

Input: 5

Output:



Constraints: $1 <= n <= 10$

Steps:

1. Take the input number.
2. We will use spaces and stars approch.
3. We observe for n=5, spaces=4, stars=1
4. Print 4 space with a loop and print stars k=0 to stars print stars.
5. After each row println
6. Modify now stars++; and spaces--;
7. Continue the same steps (3 to 6) till all rows are covered.

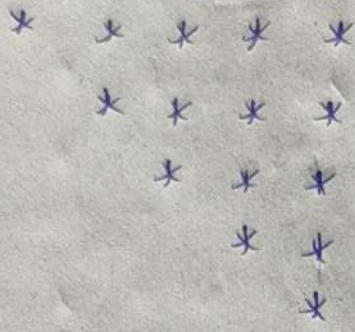
Program;

```
int n=sc.nextInt(); int stars=1; int spaces=n-1;
for(int i=0; i<n; i++){                    // Through rows
    for(int j=0; j<spaces; j++)            // Through spaces and printing
        print(" \t");
    for(int k=0; k<stars; k++)             // Through stars and printing
        print("*\t");
    println();
    stars++;
    spaces--;
}
```

4.

Input: 5 (n)

Output :



Constraints: $1 <= n <= 100$

Steps:

1. Take the input number n and stars = n and spaces = 0
2. Iterate through rows and print no of spaces and no of stars by using respective loops.
3. Now After each row, move cursor to next line
4. Modify stars--; and spaces++;

Program:

```
int n = sc.nextInt();
int stars = n;
int spaces = 0;
for(int i=0; i<n; i++) {
    for(int j=0; j<spaces; j++)
        Print(" ");
    for(int k=0; k<stars; k++)
        Print("* ");
    println();
    spaces++;
    stars--;
}
```

5.

Input: n [5]

Output:

```
    *
  * * *
* * * * *
  * * *
    *
```

stars	spaces
1	2
3	1
5	0
3	1
1	2

Constraints: $1 \le n \le 100$

Also n is odd

Steps:

1. Take number n and we observe stars are incrementing by 2 in the first half and decrementing by 2 in the second half. So, stars = 1 initially.
2. We observe spaces are decrementing by 1 in the first half and incrementing by 1 in the second half. So, spaces = n/2 [initially]
3. Iterate through rows and print corresponding number of spaces and stars w.r.t loops.
4. Now, move to new line modify for the first Half: stars+=2; spaces--; second half: stars-=2; spaces++;

Program:

```
int n = sc.nextInt();
int stars = 1;
int spaces = n/2;
for(int i=0; i<n; i++) {
    for(int j=0; j<spaces; j++)
        print(" ");
    for(int k=0; k<stars; k++)
        print("*");
    println();
    if i<n/2 { spaces--; stars=stars+2; } else { spaces++; stars=stars-2; }
}
```

6.

Input: 5

Output:

```
*  *  *        x  x  x
*  *              x  x
*                    x
*  *              x  x
*  x  x        x  x  x
```

stars	space	stars
3	1	3
2	3	2
1	5	1
2	3	2
3	1	3

Constraints: $1 <= n <= 100$

n is odd

Steps:

1. Take the input n and by observing the pattern stars = n/2+1 and spaces = 1
2. ~~Spaces~~ stars are decrementing by 1 in 1st half and incrementing by 1 in 2nd half s|ly ~~stars~~ spaces ↑2 and ↓2 respectively.
3. After each row, modify star s and spaces

Program:

```
int n = scn.nextInt();
int stars = n/2+1;
int spaces = 1;
for(int i=0; i<n; i++) {
    for(int j=0; j<stars; j++)
        print("*");
    for(int k=0; k<spaces; k++)
        print(" ");
    for(int j=0; j<stars; j++)
        print("*");
    println();
    if(i<n/2){
        stars--;
        spaces+=2;
    } else {
        stars++;
        spaces-=2;
    }
}
```

7. Input: 5

Output: *

Constraints: 1 <= n <= 100

Steps:

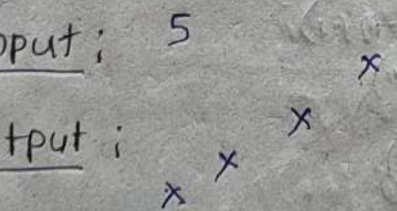
1. Take the input number n
2. Iterate through i -> 1 to n, j 1 to n, print * if (i==j) else "\t"
3. Print New Line After Each Row.

Program:

```
for(int i=1; i<=n; i++){
    for(int j=1; j<=n; j++){
        if (i==j)
            print("*");
        else
            print("\t");
    }
    println();
}
```

8. Input: 5

Output:



Constraints: 1 <= n <= 100

Steps:

1. Take the input number n
2. Iterate through i->1 to n, j->1 to n, Print * if i+j==n+1 else "\t"
3. Print New Line After Each Row

Program:

```
int n = sc.nextInt();
for(int i=1; i<=n; i++){
    for(int j=1; j<=n; j++){
        if(i+j==n+1)
            print("*")
        else
            print("\t");
    }
    println();
}
```

9. Input: n

Output:

```
*       *
  *   *
    *
  *   *
*       *
```

Constraints: $1 <= n <= 100$

Steps: Combine Pattern 7 and Pattern 8

Program:

```
int n = sc.nextInt();
for(int i=1; i<=n; i++){
    for(int j=1; j<=n; j++){
        if(i==j || i+j==n+1)
            print("*");
        else
            print("\t");
    }
    println();
}
```

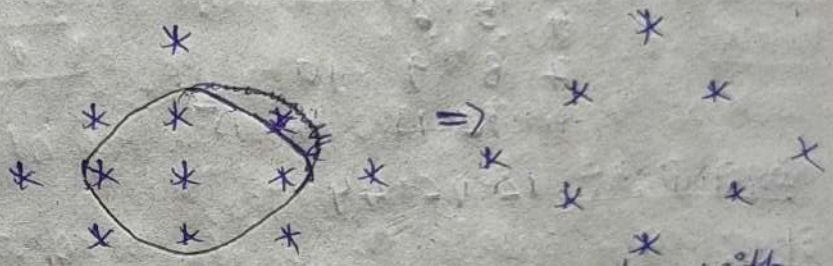
10. Pattern

Input: 5

Output:

Constraints: $1 <= n <= 100$

Steps: 1. Build the pattern 5



2. Stars inside the figure replace with \t.
3. We can observe that stars are present only in first and last of each row and fill it with space.

Program:

```
int n=sc.nextInt();
int stars=1;
int spaces=n/2;
for(int i=1; i<=n; i++){
    for(int j=1; j<=spaces; j++)
        print("\t")
    for(int j=1; j<=stars; j++){
        if(j==1 || j==stars)
            print("*\t");
        else
            print("\t");
}
```

println();
if(i<=n/2){
spaces--;
stars+=2;
}else{
spaces++;
stars-=2;
}
}

11. Input 5

Output:

```
1
2 3
4 5 6
7 8 9 10
11 12 13 14 15
```

Constraints: $1 <= n <= 44$

Steps:

1. Take input number and count = 1
2. Take i -> 1 to n and j -> 1 to i, for each j print count and increment count
3. Print New Line After Each Row

Program:

```
int n = sc.nextInt();
int count = 1;
for(int i=1; i<=n; i++){
    for(int j=1; j<=i; j++){
        print(count+" ");
        count++;
    }
println();
}
```

12. Input: 5

Output:

```
0
1 1
2 3 5
8 13 21 34
55 89 144 233 377
```

Constraints: $1 <= n <= 5$

Steps:

1. Take input Number n
2. First Construct

```
      *                    0
n=4   * *        ⟹        1 1
      * * *                2 3 5
      * * * *              8 13 21 34
```

3. Then Replace with Numbers
4. By looking at the numbers, we can tell they are fibonacci numbers [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

Program:

```
int n=scn.nextInt();
int a=0, b=1;
for(int i=1; i<=n; i++){
    for(int j=1; j<=n; j++) {
        print(a+" ");
        c=a+b;
        a=b;
        b=c;
    }
print();
}
```

13. Input: n [5]

Output:

```
    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
```

Constraints: $1 <= n <= 10$

Steps:

1. First construct

```
*
* *
* * *
* * * *
* * * * *
```

Then =>

```
    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
```

2. We can clearly observe the numbers at each row are Binomial coefficient of corresponding Row No. Eg: Row 3: ${}^3C_0$ ${}^3C_1$ ${}^3C_2$ ${}^3C_3$ [1 3 3 1]

3. We know the starting Binomial coefficient of each row is 1 $[{}^1C_0, {}^2C_0, {}^3C_0, {}^4C_0]$.

4. We can calculate ${}^nC_{r+1}$ if we know ${}^nC_r$

$${}^nC_{r+1} = \frac{n!}{(n-r-1)!(r+1)!}$$

$$= \frac{n!(n-r)}{(n-r)(n-r-1)!(r+1)!}$$

$$= \frac{n!(n-r)}{(n-r)(n-r-1)!(r+1)r!}$$

$$= \left(\frac{n!}{(n-r)!r!}\right) \times \frac{n-r}{r+1}$$

$${}^nC_{r+1} = {}^nC_r \frac{n-r}{r+1}$$

5. Replace i with n, j with r

$$= {}^iC_j \frac{(i-j)}{j+1}$$

6. Initially For Every Row $i_{C_j} = 1$

↓ value

DryRun:

n = 4 i = 0; i < n; i++

int value = 1;

j = 0 to i

print value

value = (value x (i - j)) / (j + 1);

i = 0, < 4 ✓

i = 0

value = 1

j = 0 ≤ 0 ✓

print value (1) ✓

value = 1 x 0 / 1 = 0

j = 1 ≤ 0 ✗

i = 1

value = 1

j = 0 ≤ 1

print (value) ✓

value = 1 x (1 - 0) / 0 + 1

value = 1

j = 1 ≤ 1 ✓

print (value) ✓

value = 1 x (1 - 1) / 1 + 1

= 0

j = 2 ≤ 1 ✗

i = 2

value = 1

j = 0 ≤ 2 ✓

print (value) ✓

value = 1 x (2 - 0) / 1

= 2

j = 1 ≤ 2

print (value) ✓

value = 2 x (2 - 1) / 2

value = 1

j = 2 ≤ 2 ✓

print (value) ✓

j = 3 ≤ 2 ✗

i = 3

value = 1

j = 0 ≤ 3 ✓

print (value)

value = 1 x (3 - 0) / 1

= 3

j = 1 ≤ 3 ✓

print (value)

value = 3 x (3 - 1) / 2

= 3

j = 2 ≤ 3 ✓

print (value) ✓

value = 3 x (3 - 2) / 2 + 1 = 3

j = 3 ≤ 3 ✓

print (value)

value = 3 x (3 - 3) / 3 + 1 = 0

j = 4 ≤ 3 ✗

Program:

```
int n = scn.nextInt();
for(int i=0; i<n; i++){
    int value=1;
    for(int j=0; j<=i; j++){
        print(value);
        value=(value*(i-j))/(j+1);
    }
    println();
}
```

14.

Input: 3

Output: 3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
3 * 10 = 30

Constraints: $1 <= n <= 10$

Steps:

1. Take the input number
2. Iterate from i: 1 to 10
3. Multiply n with i and print in required format
4. After each line give enter

Program;

```
int n = scn.nextInt();
for (int i=1; i<=10; i++)
    {int k = i*n;
     System.out.println(n+"*"+i+" = "+k);
    }
```

15.

Input; 5 (n)

Output;

```
        1
      2 3 2
    3 4 5 4 3
      2 3 2
        1
```

Constraints; $1 <= n <= 10$ [Also n is odd]

Steps;

1. Construct by looking at Patterns

```
      *
    * * *
  * * * * *
    * * *
      *
```

2. Transform to

```
        1
      2 2 2
    3 3 3 3 3
      2 2 2
        1
```

inside for loop
take temp = i
print temp instead of *

3. Then change

```
        1
      2 3 4
    3 4 5 6 7
      2 3 4
        1
```

Take count = 1;
and assign temp = count
. print temp now and increment temp in the stays loop
-> Modify the count in 1st half and 2nd half

4) Now change to

```
    1
  2 3 4
3 4 5 4 3
  2 3 4
    1
```

we observe in ~~eve~~ previous step that all rows are correct and in middle rows change the temp
↳ in first mid → temp++
second half → temp--;

Program:

```
int n = sc.nextInt();
int spaces = n/2;
int stars = 1;
int count = 1;
for(int i=1; i<=n; i++) {
    int temp = count;
    for(int j=0; j<spaces; j++)
        print("\t");
    for(int k=0; k<stars; k++) {
        print(temp + "\t");
        if(k<stars/2)
        temp++;
        else
        temp--;
    }
    println();
    if(i<=n/2) {
        spaces--;
        stars+=2;
        count++;
    }
    else {
        spaces++;
        stars-=2;
        count--;
    }
}
```

16.

Input: n

Output:

```
        1
    1   2       2   1
1   2   3       3   2   1
1   2   3   4   3   2   1
```

Constraints: $1 <= n <= 10$

Steps:

1. Construct with stars

```
*       *
* *     * *
* * *   * * *
* * * * * * *
```

2. We can observe that initially spaces are $2n-3$ [As total No of columns in each row is $2n-1$].
   ↳ After subtracting 2 stars in 1st row.

3. So, stars = 1, spaces = $2n-3$, stars are incrementing by 2 and spaces are decrementing by 1 we will get 4

4. In last row, As one star is extra, we simply performing the operation of last stars loop, decrement star then 3 2 1 will be achieved.

5. Now replace * with j variable with forward and in last [second] star for loop with backward.

Program:

```
int n = scn.nextInt();
int spaces = 2*n-3;
int stars = 1;
for(int i=1; i<=n; i++) {
    for(int j=1; j<=stars; j++)
        print(j+"\t");
    for(int j=1; j<=spaces; j++)
        print("\t");
    // last row [mid half]
    if(i==n)
        stars--;
    for(int j=stars; j>=1; j--)
        print(j+"\t");
    println();
    stars++;
    space = space-2;
}
```

17.

Input: 5 [n]

Output:

```
        *
      * *
    * * * * *
      * *
        *
```

os	star	os
2	1	2
2	2	1
0	5	0
2	2	1
2	1	2

Constraints: $1 <= n <= 10$

Also, n is odd

steps;

1. from the pattern we can observe os are os are constant and stars are incrementing by 1 in first half and decrementing by 1 in second half.

oos are decrementing by 1 in first half and incrementing by 1 in second half.

2. But if we observe in the middle row, no os space, but we will adjust with stars by checking with condition by keep os is constant so 3rd row os star oos 2 3 0

Program;

```
int n = scn.nextInt();
int os = n/2;
int stars = 1;
int oos = n/2;
for(int i=1; i<=n; i++){
    for(int j=0; j<os; j++){
        if(i==n/2+1)
            print("*\t");
        else
            print("\t");
    }
    for(int k=0; k<stars; k++)
        print("*\t");
    for(int j=0; j<oos; j++)
        print("\t");
    println();
    if(i<=n/2){
        stars++;
        oos--;
    }else{
        stars--;
        oos++;
    }
}
```

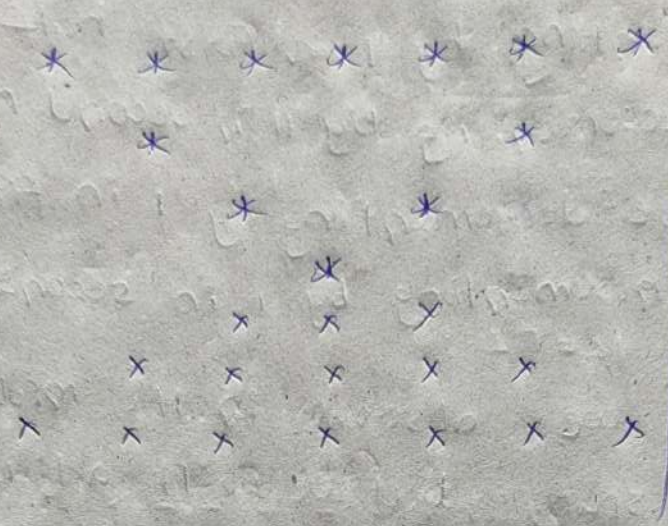
// modifying middle row (reference endpoint)

18.

Input; 7

Constraint: $1 <= n <= 10$
n is odd

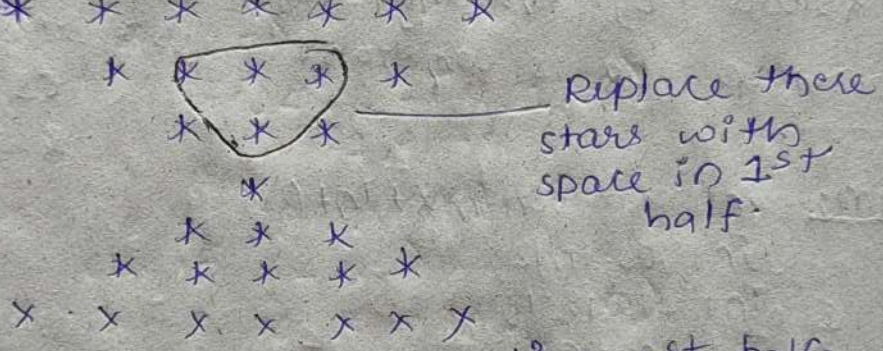
Output;



spaces	stars
0	7
1	5
2	3
3	1
2	3
1	5
0	7

Steps;

1. Construct



2. We can observe space↑, stars↓^{2} in 1st half
space↓, stars↑^{2} in 2nd half.

Program;

```
int n=sc.nextInt();
int spaces=0;
int stars=n;
for(int i=1;i<=n;i++){
    for(int j=0;j<spaces;j++)
        print("\t");
    for(int j=0;j<stars;j++){
```

```
if(i<=n/2 && i>1){
    if(j>1 && j<stars-1)
        print(" ");
    else print("*");
else
    print("*");
println();
if(i<=n/2){
    spaces++;
    stars=stars-2;
}
else{
    spaces--;
    stars=stars+2;
}
```

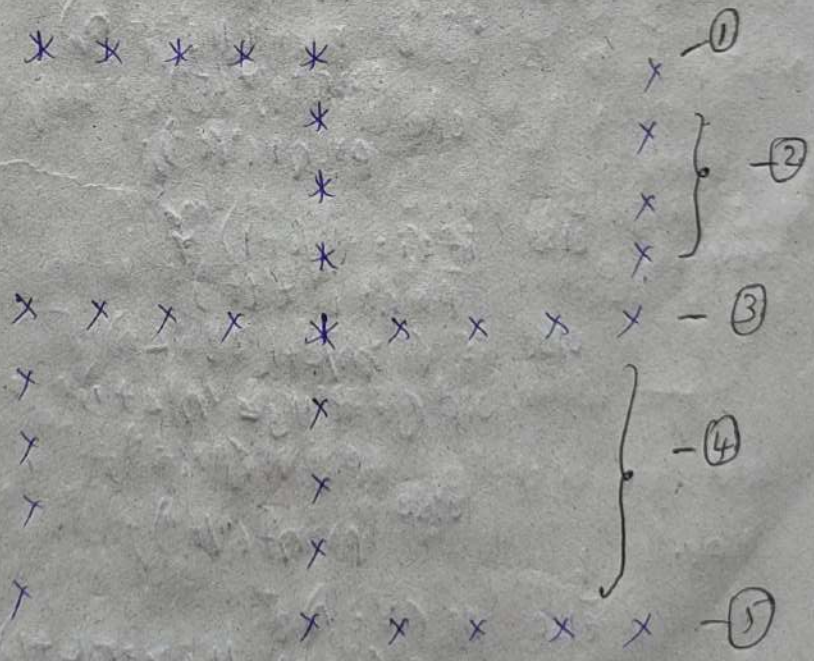
// checking for 1st half with 2nd row

printing only first last element star other wise " ";

19. Input: 9

Constraints: $1 <= n <= 10$, n is odd

Output:



Steps:

1) We can divide the pattern into 5 partition, where common pattern exists in each row

```
i == 1
i <= n/2
i == n/2+1
i > n/2+1 && i < n
i == n
```

} In the 5 partitions print stars and spaces at required positions.

Program:

```
int n = sc.nextInt();
for(int i=1; i<=n; i++) {
    for(int j=1; j<=n; j++) {
        if (i==1) {
            if (j<=n/2+1 || j==n)
                print("*");
            else
                print(" ");
        }
        else if (i<=n/2) {
            if (j==n/2+1 || j==n)
                print("*");
            else
                print(" ");
        }
        else if (i==n/2+1) {
            print("*");
        }
        else if (i>n/2+1 && i<n) {
            if (j==1 || j==n/2+1)
                print("*");
            else
                print(" ");
        }
        else if (i==n) {
            if (j==1 || j>=n/2+1)
                print("*");
            else print(" ");
        }
    }println();
```

20.

Input: 7

Constraints: i<=n<=10
n is odd.

Output:

Steps:

1) We can observe in every first and last column star is present

2) From mid+1 row, we see stars present in i==j and i+j == n+1

3) In Rest places print tab spaces

Program:

```
int n= sc.nextInt();
for(int i=1; i<=n; i++){
    for(int j=1; j<=n; j++){
        if(j==1 || j==n)
            print("*\t")
        else if(i>n/2 && (i==j || i+j==n+1))
            print("*\t");
        else
            print("\t");
    }
    println();
}
```

21. Bonus Question-1

Input: 5

Output:

1	2	3	4	5
11	12	13	14	15
21	22	23	24	25
16	17	18	19	20
6	7	8	9	10

Input: 4

Output:

1	2	3	4
9	10	11	12
13	14	15	16
5	6	7	8

Steps:

1. Take the input number and assign it to temp.
2. First consider the first half,
   i -> 1 to n with i=i+2
   and
   take counter = (i-1)xn+1
   with j loop print values and increment counter
   for every row print enter.
3. Now, if n is odd, decrement one number to the temp otherwise keep it same.
4. Now same as above perform in reverse order i -> tempNum to i with i=i-1
   counter = (i-1)xn+1
   ~~and~~ with j loop print values, increment counter
   and for each row print new line.

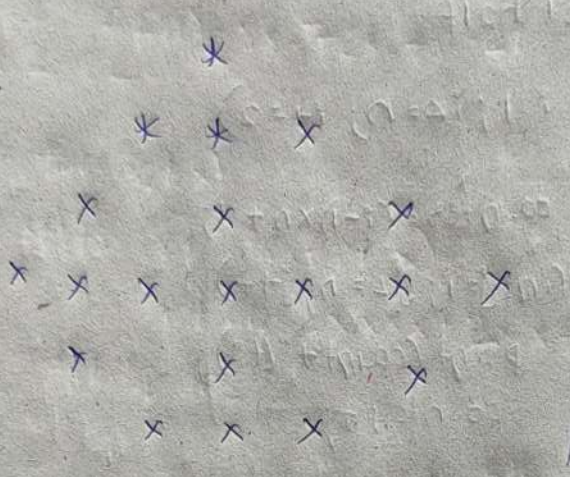
Program;

```
int n= sc.nextInt();
int tempNum=n;
// first half
for(int i=1; i<=n; i+=2)
{
  int counter=(i-1)*n+1;
  for(int j=1; j<=n; j++) {
      print(counter+"\t");
      counter++;
  }
  println();
}
// Now second half, If n is odd
  if(n%2!=0)
    {tempNum=n-1;
    }
// second half
for(int i=tempNum; i>=1; i-=2)
  {
    int counter=(i-1)*n+1;
    for(int j=1; j<=n; j++) {
        print(counter+"\t");
        counter++;
    }
    println();
}
```

22. Bonus Question

Input: 7

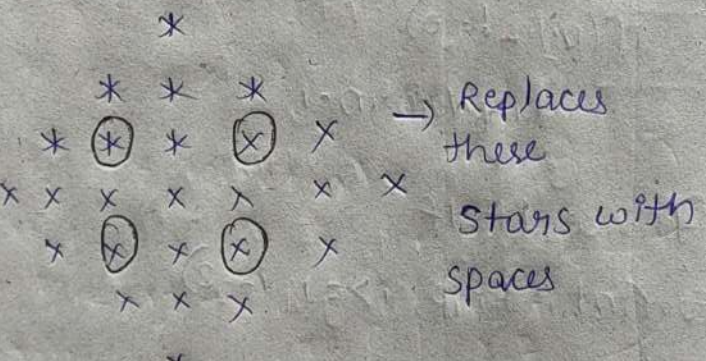
Output:



Stars	Spaces
1	3
3	2
5 (3)	1
7	0
5 (3)	1
3	2
1	3

Steps:

First Construct initially stars = 1
spaces = n/2



→ we observe stars ↑ 2 in first Half
spaces ↓ 1 in first Half and

stars ↓ 2 in second Half
spaces ↑ 1 in second Half

Program:

```
int n = sc.nextInt();
for(int i=1; i<=n; i++)
    {
    for(int j=0; j<spaces; j++)
        print("\t");
    for(int k=0; k<stars; k++){
        if(i==n/2 || i==n/2+2) {
            if(k==0 || k==stars-1 || k==stars/2)
                print("*\t");
            else
                print("\t");
        }
        else
            print("*\t");
    }
    println();
    if(i<=n/2){
        spaces--;
        stars+=2;
    }
    else{
        spaces++;
        stars-=2;
    }
}
```

Functions And Arrays

Function

In Mathematical, $f(x) = 3x^2 + 2x + 5$

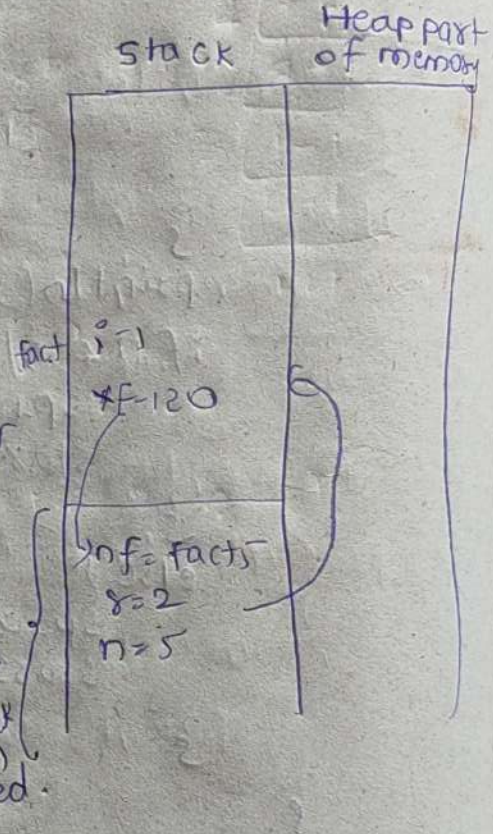
$f(2) = 3 \times 2^2 + 2 \times 2 + 5 = 21$

In Java, func(int x)

```
public static int func(int x){
        int ans = 3*x*x + 2*x + 5
        return ans;
    }
```

Eg: for $^nC_r = \frac{n!}{(n-r)! \times r!}$

```
int nf = factorial(n);
int rf = factorial(r);
int nrf = factorial(n-r);
public static int factorial(int x){
    int xf = 1;
    for(int i = 1; i <= x; i++){
        xf *= i;
    }
    return xf;
}
```



→ Code can be re-used.

→ Modularity { Optimization of 1 places
Any change reflect in the whole code

1. Digit Frequency

Input: 994543234 = n
4 = d

Output: 3

Constraints; $0 <= n <= 10^9$
$0 <= d <= 9$

Steps:

1. Take the input number n, d and initialize count=0
2. Get the remainder of number by %10, and check remainder is equal to d if yes increment count
3. Modify the n → n/10
4. Continue the whole process till n!=0

Program:

```
int count=0;
while(n!=0)
{
    int r=n%10;
    if(r==d)
        count++;
    n=n/10;
}
return count;
```

2. Decimal to Any Base

Input: 57 [2 / 6]
Output: 111001
Constraints: $0 \le d \le 512$
$2 \le b \le 10$

Steps: Take input n and b (Assumed to be assumed)

1. By looking at the dry, we need to divide n with b
2. Store, rem = n%b, As we observe we need to print the rem in reverse order we will multiply with multiples of 10 [initially multi = 1]
3. Modify the n and multi values
4. Continue the process till n != 0

Dry Run:

2	57
2	28 - 1
2	14 - 0
2	7 - 0
2	3 - 1
2	1 - 1
	0 - 1

111001

Program:

```
public static int getValueInBase(int n, int b){
    int multi = 1;
    int ans = 0;
    while (n != 0) {
        int rem = n % b;
        n = n/b;
        ans += rem * multi;
        multi = multi * 10;
    }
    return ans;
}
```

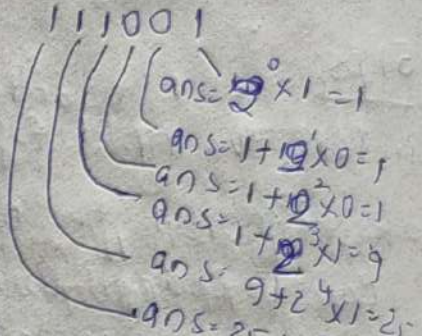
3. Any Base to Decimal

Input: 111001 [n, b]
Output: 2, 57
Constraints: $0 <= n <= 10^9$
$2 <= b <= 10$

Steps:

1. We observe through dry run
2. We need to get the last digit of number and multiply with the increasing powers of base (from 0 to ...)
3. Modify n → n/10
4. Continue the process till n != 0

Dry Run:



Program:

```
~~int po~~
public static int getValueInDecimal(int n,
                                     int b)
{
int pow=0;
int ans=0;
while(n!=0){
    int rem=n%10;
    n=n/10;
    ans+=rem*(int)Math.pow(b,pow);
    pow++;
}
return ans;
}
```

4. Any Base to Any Base

Input: 1100, 2, 3 [n, source base, destination base]

Output: 2010

Constraints: $0 <= n <= 512$
$2 <= b1 <= 10$
$2 <= b2 <= 10$

Steps:

1. Take the inputs and bases from the ~~parameter~~ scanner
2. First convert the number from source Base to Decimal Base.
3. Then convert the number from decimal base to Destination Base.
4. Use the problems 2 and 3 to solve problem 4

Program:

```
int n = sc.nextInt();
int sourceBase = sc.nextInt();
int ~~dest~~Base = sc.nextInt();
int temp = n;
int pow = 0;
int rem = 0;
int ans = 0;
int temp1 = 1;
while (temp != 0){          // sourc base to Decimal Base
    int rem = temp % 10;
    temp = temp / 10;
    ans = (int) Math.pow(sourceBase, pow) * rem;
    pow++;
}
```

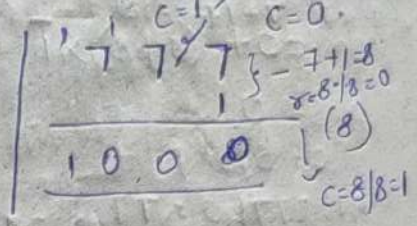
```
while(rev1 != 0){
    int rem = rev1 % destBase;
    rev1 = rev1 / destBase;
    ans = rem * temp1;
    temp1 *= 10;
}
println(ans);
}
```

// Decimal Base to Destination Base

5. Any Base Addition

Input: $\begin{matrix} & 8 & \\ 777 & & (b \\ 1 & & n1 \\ & & n2 \end{matrix}$

Output: 1000



Constraints:
$2 <= b <= 10$
$0 <= n1 <= 10^9$
$0 <= n2 <= 10^9$

Steps:

1. Get the last digit of n1 and n2.
2. Intially carry=0, Now perform Addition of carry of last digit of n1 and last digit of n2.
3. The sum % b gives remainder and sum / b gives the carry which need to be modified.
4. As the remainder, gives out the respective digits of answer they should be multiplied with powers of 10.
5. Now modify $n1 \rightarrow n1/10$, $n2 \rightarrow n2/10$, and increment pow.
6. Continue the process till $c > 0$ ($n1 > 0$ || $n2 > 0$ || $c > 0$)

~~Program:~~

Dry Run:

c=0
ans=0
pow=1
n1!=0, n2!=0, c>0
rem1 = n1 % 10
= 7
rem2 = n2 % 10 = 1
temp = 7+1+0 = 8
ans = 0 + (8 % 8) * 1
ans = 0
c = 8/8 = 1
n1 = 777/10 = 77
n2 = 1/10 = 0, pow = 10
77!=0 || 0==0 || c>0
rem1 = 77 % 10 = 7
rem2 = 0 % 10 = 0
temp = 7+0+1 = 8
ans = 0 + (8 % 8) * 10
= 0
c = 77/10 = 7, c = 8/8 = 1
n1 = 77/10 = 7, n2 = 0/10 = 0, pow = 100

n1 = 777, n2 = 1, b = 8

1000

7!=0, 0==0, c>0
rem1 = 7 % 10 = 7
rem2 = 0
temp = 7+0+1 = 8
ans = 0 + (8 % 8) * 100
ans = 0
c = 8/8 = 1
n1 = 7/8 = 0, n2 = 0, pow = 1000
0!=0, 0==0, c = 1 > 0
rem1 = 0, rem2 = 0
temp = 0+0+1 = 1
ans = 0 + (1 % 8) * 1000
= 1000
c = 1/8 = 0, pow = 1000
n1 = 0, n2 = 0, c = 0 > 0
X X X

Program:

```
int c=0;
int ans=0;
int pow=1;
while(n1!=0 || n2!=0 || c>0) {
    int rem1 = n1%10;
    int rem2 = n2%10;
    int temp = rem1+rem2+c;
    ans += (temp%b)*pow;
    c = temp/b;
    pow = pow*10;
    n1 = n1/10;
    n2 = n2/10;
}
return ans;
}
```

6. Any Base subtraction

Input 8 [8 n1 n2]

100

Output 77

	-1	-1		
	1	0	0	c=0
			1	0+0 < 1
	0	7	7	-1+10

Constraints; $2 \le b \le 10$

$0 \le n1 \le 10^9$

$0 \le n2 \le 10^9$

Steps;

1. Get the last Digit of n1 and 2, set c=0
2. Now add c to rem2 (~~rem2~~ last digit of n2) ~~set~~
3. If rem2 >= rem1, set c=0 and update answer with rem2-rem1 in the multiples of 10
4. Else, modify c=-1, update ans with rem2+b-rem1 in the multiples of 10.
5. Modify n1 → n/10, n2 → 10 and pow
6. Continue till n1>0 || n2>0 || c!=0

Dry Run;

b=8,

pow=1

ans=0

n2 1 0 0 = n2 b=8

n1 1 = n1

n1>0 || n2>0 || c!=0

rem1 = n1%10 = 1

rem2 = 100%10 = 0

rem2 = 0+0 = (0+c) = 0

0 >= 1 ×

So, c=-1

ans = 0 + (0+8-1) × 10 = 7

n1 → 1/10 = 0

n2 → 100/10 = 10.

pow = 10

~~i=0~~

n1>0, n2>0, c!=0

rem1 = 0

rem2 = 0

reme = 0 - 1 = -1

-1 >= 0 ✗

c = -1,

ans = 7 + (-1 + 8 - 0) × 10

= 7 + 70 = 77

pow = 100, n1 = 0

n2 = 10 / 10 = 1

n1 > 0, n2 > 0, c != 0

rem1 = 0, reme = 1 % 10 = 1

rem2 = 1 - 1 = 0

0 >= 0 ✓

c = 0, ans = 77 + (0 - 0) × 1000

= 77

pow = 1000, n1 = 0, n2 = 0

n1 > 0, n2 > 0, c = 0

Program:

```
public static int getDifference(int b, int n1, int n2){
    int c=0;
    int pow=1;
    int ans=0;
    while(n1>0 || n2>0 || c!=0){
        int rem1= n1%10;
        int rem2= n2%10;
        if(rem2>=rem1){
            c=0;
            ans+=(rem2-rem1)*pow;
        }
        else{
            c=-1;
            ans+=(rem2+b-rem1)*pow;
        }
        n1=n1/10; pow=pow*10;
        n2=n2/10;
    }return ans;
```

7. Any Base Multiplication

Input: 1220 [b, n1, n2]

Output: 43320

Constraints: $2 <= b <= 10$

$0 <= n1 <= 100000$

$0 <= n2 <= 100000$

Steps:

1. Get last Digit from n2 and multiply n1 with ~~that last~~ digit with base 5

2. 1220 , take carry as 0 Initially

```
    1 2 2 0
  x       1
  ---------
  1 2 2 0      ---- (1)
```

last digit from n2

$1 \times 0 = 0 + C = 0 + 0 \xrightarrow{\cdot 5} 15 \Rightarrow r = 0, C = 1$

$1 \times 2 = 2 + 0 = 2 \xrightarrow{\cdot 5} 15 \Rightarrow r = 2, C = 0$

$1 \times 2 = 2 + 0 = 2 \xrightarrow{\cdot 5} 15 \Rightarrow r = 2, C = 0$

$1 \times 1 = 1 + 0 = 1 \xrightarrow{\cdot 5} 15 \Rightarrow r = 1, C = 0$

$n1 = 0 \times, C > 0^{x}$

After each op do n1/10

3. Similary perform for other digit

```
    1 1 0
    1 2 2 0
  x       3      ---- (2)
  ---------
  4 2 1 0
  ---------
```

$3 \times 0 = 0 + C = 0 + 0 \xrightarrow{\cdot 5} 15 \Rightarrow r = 0, C = 0$

$3 \times 2 = 6 + 0 = 6 \xrightarrow{\cdot 5} 15 \Rightarrow r = 1, C = 1$

$3 \times 2 = 6 + 1 = 7 \xrightarrow{\cdot 5} 15 \Rightarrow r = 2, C = 1$

$3 \times 1 = 3 + 1 = 4 \xrightarrow{\cdot 5} 15 \Rightarrow r = 4, C = 0$

$n1 = 0 \times, C > 0$

After each op, do n1/10

4. Express each product in the multiples of 10 to get total value

5. We know,

```
    a b c
  x d e   =>
```

```
      1 1 1
    x   1 1
  ---------
      1 1 1      ---- x10^0
    1 1 1 (x)    ---- Replace this x with multiples of 10
  ---------
    1 2 2 1
  ---------
```

[x10^1]

⑥ Now perform ~~after~~ addition (Base) after multiplying with the 10 power

⑦ Refer to problem 5 for base addition.

⑧ Continue the process till we multiply with all digits of n2 with n1 and perform addition

```
① x 10^0      1 2 2 0
② x 10^1    + 4 2 1 0 0     base = 5
            -------------
            4 3 3 2 0       0+0 = 0
```

Program:

```
public static int getProduct (int b, int n1, int n2) {
    int ans = 0;
    int pow = 1;
    while (n2 > 0) {                    // continue till n2 digits are covered
        int temp = n2 % 10;
        int value = getSingleProduct (b, n1, temp);
        ans = getProductSum (b, ans, value * pow);
        pow = pow * 10;
        n2 = n2 / 10;
    }
    return ans;
}
                                        // Multiply n1 with last digit of n2
public static int getSingleProduct (int b, int n, int d) {
    int pow = 1;
    int ans = 0;
    int c = 0;
    while (n1 > 0 || c > 0) {
```

```
        int rem = n1 % 10;
        int temp = (rem * d) + c;
        ans += (temp % b) * pow;
        c = temp / b;
        n1 = n1 / 10;
        pow = pow * 10;
    }
    return ans;
}
// Each product sum
public static int getProductSum(int b, int n1, int n2) {
    int pow = 1;
    int ans = 0;
    int c = 0;
    while (n1 > 0 || n2 > 0 || c > 0) {
        int rem1 = n1 % 10;
        int rem2 = n2 % 10;
        int temp = rem1 + rem2 + c;
        ans += (temp % b) * pow;
        c = temp / b;
        pow = pow * 10;
        n1 = n1 / 10;
        n2 = n2 / 10;
    }
    return ans;
}
```

Array:

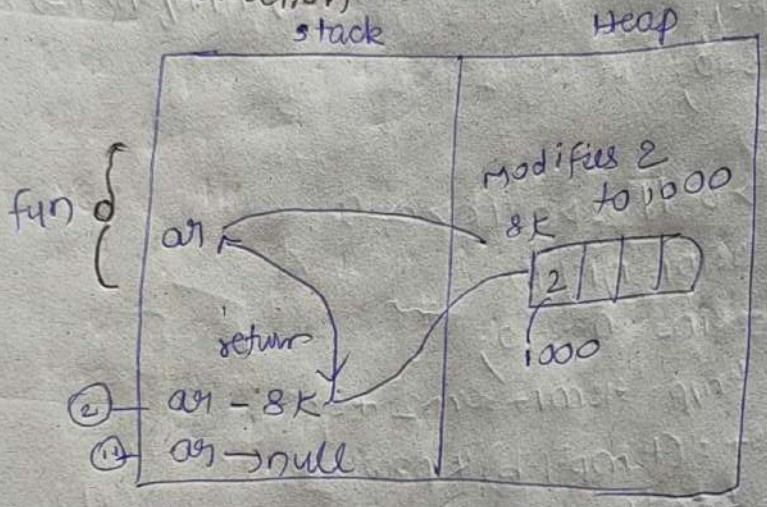
Contigous Space Allocation

Indexing

```
int [] ar;   ①
ar = new int[5]
```

↓

HeapMemory Collection




```
public static fun(int[] ar)
{
    ar[0] = 1000;
}
```

After function call, members in stack will wipe out, the values stored will also be get modified due to the changes in the function.

8. Span Of An Array:

Input:
6
15
30
40
4
1
9

[n
element 1
element 2
...
element n]

Output:
36 [40-4=36, max-min=36]

Constraints: $1 <= n <= 10^4$
$0 <= n1, n2 \ldots$ n elements $<= 10^9$

Steps:
1. Take input n and fill it with elements
2. Find the max element by max = arr[0] initially, and then go through i:0 to n-1 and check if arr[i] > max so then modify max = arr[i].
3. Find the min element by min = arr[0] initially and then go through i:0 to n-1 and check if arr[i] < min ; so then modify min = arr[i].
4. Print max-min

Program:

```
int n = sc.nextInt();
int a[] = new int[n];
for(int i = 0; i<n; i++)
    a[i] = sc.nextInt();
int max = a[0];
int min = a[0];
for(int i=0; i<n; i++) {
    if(a[i] > max)
        max = a[i];
    if(a[i] < min)
        min = a[i];
}
print(max-min);
```

9: Find Element in An Array

Input: 6 Elements: 15 30 40 4 1 9 Find: 40

Output: 2

Constraints: $1 <= n <= 10^7$

$-10^9 <= n1, n2 .. \text{elements} <= 10^9$

$-10^9 <= d <= 10^9$

Steps:

1. Take input: size and elements and target element.
2. Traverse through the array $i \rightarrow 0$ to $n-1$ and check if $arr[i] == d$, print index
3. If no element after traversing, print -1

Dry Run: $n = 3$

12 9 1 , $d = 9$

$i = 0$, $a[0] != d$,

$a[1] == d$ ✓

print 1

Program:

```
int n = sc.nextInt();
int a[] = new int[n];
for(int i=0; i<n; i++){
    a[i] = sc.nextInt();
}
int d = sc.nextInt();
for(int i=0; i<n; i++){
    if(a[i] == d)
        print(i)
        return;
}
print(-1);
```

10. Sum of two Arrays

Input:
5 — n1
3
1
0
7
5
6 — n2
1
1
1
1
1

Output:
1
4
2
1
8
6

Constraints:
1 <= n1, n2 <= 100
0 <= a1[i], a2[i] < 10

Steps:

1) Take the input n1 and elements, then n2, elements, c=0
2) The resulting array size after sum will be ~~greatk~~ maxSize(of n1, n2) + 1.
3) Get the last element of two arrays and add them store the remainder in new Array and quotient in the c
4) To avoid Array Index Out of Bound, add elements from the two arrays only ~~if~~ if i >= 0 and j >= 0. Then at last decrement i, j, k
5) ~~Use appropriate var~~ Continue the process till size of New Array Reaches 0.
6) Check for the Leading zeros are present in the first element of array, if present ignore the first element print the remaining elements.

Dry Run:

C=0 (carry)

3>2

a = 7 2 3 k = 3 + 1 sum = 0

b = 1 1 k = 4

i = 2 ≥ 0, j = 1 ≥ 0, k = 3 ≥ 0

sum = c = 0

sum = 0 + 3 = 3 [i ≥ 0]

sum = 3 + 1 = 4 [j ≥ 0]

sum = 4 % 10 = 4

add[3] = 4

c = sum / 10 = 0

i--, j--, k--

i = 1, j = 0, k = 2 ≥ 0 ✓

sum = 0 [c = 0]

sum = 0 + a[i] = 0 + 2 = 2 [i ≥ 0]

sum = 2 + a[j] = 2 + 1 = 3 [j ≥ 0]

sum = 3 % 10 = 3

add[2] = 3

c = sum / 10 = 3 / 10 = 0

i--, j--, k--

i = 0, j = -1, k = 1 ≥ 0 ✓

sum = 0

sum = 0 + a[i] = 0 + 7 = 7 [i ≥ 0]

j ≥ 0 ✗

sum = 7 % 10 = 7

add[1] = 7

c = 7 / 10 = 0

i--, j--, k--

i = -1, j = -1, k = 0 ≥ 0 ✓

sum = 0

i ≥ 0 ✗, j ≥ 0 ✗

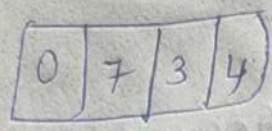
sum = 0 % 10 = 0

add[0] = 0

c = 0 / 10 = 0

i--, j--, k--

i = -2, j = -2, k = -1 ≥ 0 ✗

add[] 

skip 0 and print 7 3 4

Program:

```
int n1=sc.nextInt();
int a[]=new int[n1];
for(int i=0;i<n1;i++)
        a[i]=sc.nextInt();
int n2=sc.nextInt();
int b[]=new int[n2];
for(int i=0; i<n2;i++)
        b[i]=sc.nextInt();
int maxSize=0;
if(n1>=n2)
        maxSize=n1+1;
else
        maxSize=n2+1;
int ans[]=new int[maxSize];
int c=0;
for(int i=n1-1,j=n2-1,k=maxSize-1;k>=0;i--,j--,k--){
        int sum=c;
        if(i>=0)
                sum+=a[i];
        if(j>=0)
                sum+=b[j];
        int add=sum%10;
        ans[k]=add;
        c=sum/10;
}
for(int i=0;i<n;i++){                    //skipping leading 0's
        if(ans[i]==0 && i==0)
                continue;
        else
                println(ans[i]);
}
```

19. Difference of Two Arrays;

Input: 3 — n_1
2
6
7
4 — n_2
1
0
0
0

Output: 1
3
3

Constraints:
$1 <= n1, n2 <= 100$
$0 <= a1(i), a2(i) <= 9$
~~a1 > a2~~
$n1 < n2$ **

Steps:
1. Take the inputs
2. The max size of Resultant Array will be the size of n2.
3. Get the last element of first array only → temp if i >= 0 else give 0 and give initially carry as 0.
4. Add the last element of second array and carry, if the sum is greter than temp, then subtract the sum and temp, assign c = 0
5. Else, Add 10 to the sum and subtract the temp, modify to c = -1
6. The resultant value is stored in the k position of new Array.
7. Continue the whole process till maxSize becomes 0.
8. Neglect the 0's and print the remaining elements.

Dry Run:

b = 1000 (Example)

a = 674

---

i=2, j=3, k=3≥0 c=0

temp = a[2] = 4 2≥0

res: a[j]+c: a[3]+c = 0+0=0

res ≥ temp X c=-1

res: 0+10-4=6

c[3]=6 ✓

i--, j--, k--

i=1, j=2, k=2≥0 ✓

temp: a[1]=7 (1≥0)

res: a[j]+c: 0 ~~+1~~ = ~~0~~ -1

res ≥ temp X c=-1

res: 0+10-7=3 = 3-1=2 [b[j]+10+c-b(i)]

c[2]=~~3~~

i--, j--, k--

i=0, j=1, k=1≥0

temp=a[0]=6 (0≥0)

res=0+-1=-1

res ≥ temp X c=-1

res: 0+10-6= 4-1 [0+10-1-6=3]

c[1]=~~3~~

i=0, j=0, k=0≥0

temp=0, (-1≥0) X

res: 1-1=0

res ≥ temp ✓ c=0

c[0]=0

i--, j--, k--

-2, -3, -4 X

0	3	2	6

↓ Neglect leading zeros

Program:

```
int n1=sc.nextInt();
int a[]= new int[n1];
for(int i=0; i<n1; i++)
    a[i]=sc.nextInt();
int n2=sc.nextInt(); int b[]=new int[n2];
for(int i=0; i<n2; i++)
    b[i]=sc.nextInt();
int maxSize=n2+1;
int ans[]= new int[maxSize];
int c=0; int res=0;
for(int i=n1-1, j=n2-1, k=maxSize-1; k>=0; i--, j--, k--){
    int temp= i>=0? a[i]:0;
    if(b[j]+c>=temp){
        res=b[j]+c-temp;
        c=0;
    }
    else{
        res=b[j]+c+10-temp;
        c=-1;
    }
    ans[k]=res;
}
int count=0;
for(int i=0; i<maxSize; i++){          // counting how many leading zeros are present
    if(ans[i]==0)
        count++;
    else
        break;
}
for(int i=count; i<maxSize; i++)       // printing elements
    print(ans[i]);
```

12. Reverse an Array

Input:	Output	Constraints
5 → n 1 2 3 4 5	5 4 3 2 1	$1 <= n <= 10^4$ $-10^9 <= a[i] <= 10^9$

Steps:

1. Take the starting position and end position of array.
2. Check if start < end
3. If yes, store the starting element in temp, and store the ending element in start and store the temp in ending element (swapping)
4. Increment start++, end--; continue till start < end

Dry Run:

a; 1 2 3 , start = 0
end = n-1 = 2

0 < 2 ✓
temp = ~~start~~ a[0] = 1
a[0] = a[2] , [a[0] = 3]
a[2] = 1
start = 1, end = 1
1 < 1 ✗ print a; 3 2 1 ✓

Program:-

```
Public static void reverse (int[] a) {
    int start = 0;
    int end = a.length - 1;
    while (start < end) {
        int temp = a[start];
        a[start] = a[end];
        a[end] = temp;
        start++;
        end--;
    }
}
```

13. Rotate An Array

Input:	Output:
5 ← n	3
1	4
2	5
3	1
4	2
5	
3 ← K	

Constraints

$0 <= n <= 10^4$

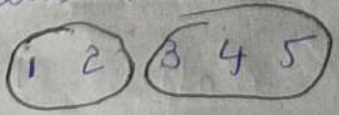
$-10^9 <= a[i], k <= 10^9$

i) K=3 == K=8

ii) K=-3 is equal to K=2

Steps:

1. ~~First~~, As the cycle continues, we will modify the k to k.length.
2. We can build for Rotating left cases on the basis of Rotating Right cases
3. So, if k<0, then k = k + a.length
   [k=-3 is equal to k=2]
4. We will partition the arrays



k=3

5. We will ~~reve~~ reverse the two partitions

(2 1) (5 4 3)

6. Now, we will reverse the entire Array

3 4 5 1 2

7. For Reversing the partitioned array / total Array Refer Problem-(12)

Program;

```
public static void rotate(int[] a, int k) {
    k = k % a.length;
    if (k < 0) {
        k = k + a.length;
    }
    reverseArray(a, 0, a.length-1-k);   // 1st partition
    reverseArray(a, a.length-k, a.length-1);   // 2nd partition
    reverseArray(a, 0, a.length-1);   // Entire Array
}
public static void reverseArray(int a[], int left, int right) {
    while (left < right) {
        int temp = a[left];
        a[left] = a[right];
        a[right] = temp;
        left++;
        right--;
    }
}
```

14. Bar chart

Input: 5 \n
3
1
0
7
5

Output:

Constraints

1 <= n <= 30
0 <= n1, n2 ... n elements

Steps:

1. As we observe the numbers of rows are max-element of the array
2. From i = max to 0, traverse j: 0 to n-1 and print * only if a[j] >= i else print(" ");
3. Hit New Line After Each Row

Program:

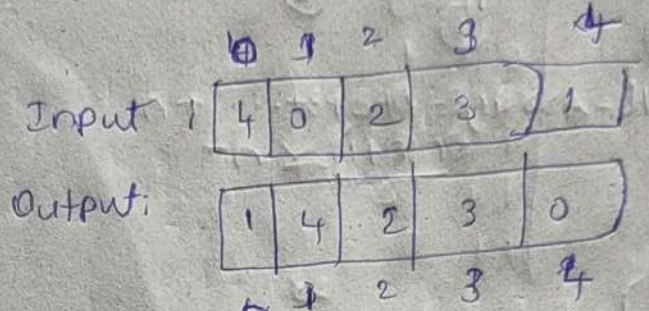
```
int n=sc.nextInt();
int a[]=new int[n];
for(int i=0;i<n;i++)
    a[i]=sc.nextInt();
int max=a[0];
for(int i=0;i<n;i++){
    if(a[i]>max)
        max=a[i];
}
for(int i=max;i>0;i--){
    for(int j=0;j<n;j++){
        if(a[j]>=i)
            print("*");
        else
            print(" ");
    }
    println();
}
```

15. Inverse of an Array;

Input:	output	Constraints:
5	1	
4	4	
0	2	
2	3	
3	0	
1		

Steps;

1.



2. We observe element in 1st array is 4, so we place the face of element in the new array at 4th position, element 0 → index 1

New Array Element 1 → 0th position

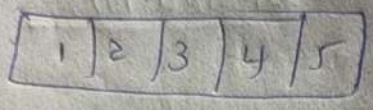
3. We should use a new Array to store the values and to return it.

Program; public static int[] inverse(int[] a) {

```
int b[] = new int(a.length);
for(int i=0; i<a.length; i++) {
    int k = a[i];
    b[k] = i;
}
return b;
}
```

16.

Sub Array: continous part of an array

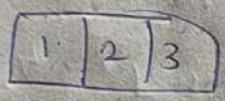


1-size	2-size	3-size	4-size	5-size
1	1,2	1,2,3	1,2,3,4	1,2,3,4,5
2	2,3	2,3,4	2,3,4,5	
3	3,4	3,4,5		
4	4,5			
5				
(5)	(4)	(3)	(2)	

Total No of sub-Arrays = $1+2+3+\cdots+n$

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

Subset:

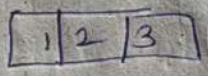


-, 1, 2, 3, 12, 13, 23, 123

for Every element consideration, there are two choices either select or not.

Total No of subset = 2^n

Subsequence



-
1
2
3
1, 2 → sequency strictly follows
1, 3
2, 3 — 2,3 - subsequence
1, 2, 3 — 3,2 not sub sequency
3,2 it can be subset

Total No of subsequences = 2^n

16. Sub Array Problem

Input: 3 — n

10
20
30
} elements

Output:

10 [0-0)
20 30 [0-1)
10 20 30 [0-2)
20 [1-0)
20 30 [1-2)
30 [2-0)

Constraints: $1 <= n <= 10$

$0 <= n_1, n_2 \ldots$ n elements $<= 10^3$

Steps:

1. Fix ~~the~~ one outer loop i goes through the array.
2. Now take another loop j: i to n-1
3. Then print the elements with tab spaces
   k = i to k <= j ~~else print~~.
4. Enter to new line after each row

Program:

```
int a[], length = n
for(int i=0; i<n; i++){
    for(int j=i; j<n; j++){
        for(int k=i; k<=j; k++)
            print(a[k] + "\t");
        println();
    }
}
```

17. Subsets of Array

Input: 3 → n
10
20 } Elements
30

Output:

- - -	0 0 0
- - 30	0 0 1
- 20 -	0 1 0
- 20 30	0 1 1
10 - -	1 0 0
10 - 30	1 0 1
10 20 -	1 1 0
10 20 30	1 1 0

Steps:

1. We know total No of subsets formed = 2^n, so 2^n rows by looking and we can also observe that the pattern is in the form of binary numbers
2. For Every i to 2^n-1 we will get the Binary numbers (Using Decimal to Binary Conversion)
3. After Extracting Binary Number, we should get each digit, we will divide the number with 10^{n-1} if r==0, print - else print the corresponding array number. [by ele] [initially ele=0]
4. Modify the binary number by 10 and pow by 10
5. eleft and continue process till power >0

Program:

```
int n = sc.nextInt();
int a[] = new int[n];
for(int i = 0; i < n; i++)
    a[i] = sc.nextInt();
int count = (int) Math.pow(2, n);
for(int i = 0; i < count; i++) {
    int bin = decimalToBinary(i);
    int div = (int) Math.pow(10, n-1);
    int ele = 0;
    while (div > 0) {
        int q = bin/div;
        if q == 0
            print("-f");
        else print(a[ele] + "(t");
        bin = bin % div;
        div = div/10;
        ele++;
    }
    println();
}
```

Decimal to Binary Conversion

```
public static int decimalToBinary(int n) {
    int ans = 0;
    int pow = 1;
    while (n != 0) {
        int r = n%2;
        n = n/2;
        ans += r * pow;
        pow = pow*10;
    }
    return ans;
}
```