Grind 4

06 May 2025 17:44

A. Vanya and Cubes

time limit per test: 1 second

memory limit per test: 256 megabytes

Vanya got n cubes. He decided to build a pyramid from them. Vanya wants to build the pyramid as follows: the top level of the pyramid must consist of 1 cube, the second level must consist of $1+2=3$ cubes, the third level must have $1+2+3=6$ cubes, and so on. Thus, the i-th level of the pyramid must have $1+2+...+(i-1)+i$ cubes.

Vanya wants to know what is the maximum height of the pyramid that he can make using the given cubes.

Input

The first line contains integer n $(1 \le n \le 10^4)$ — the number of cubes given to Vanya.

Output

Print the maximum possible height of the pyramid in the single line.

Examples

input
```
1
```
output
```
1
```

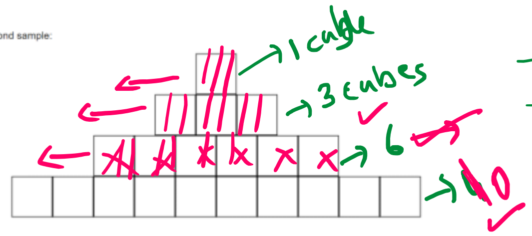
input
```
25
```
output
```
4
```

Note

Illustration to the second sample:

Note

Illustration to the second sample:



Math → Basic ✓ + logic → 800 ↓ 1000

n cubes → how many level can be build

→ order↑ } Pattern

1, 3, 6, 10 ✓

1   1+2   1+2+3   1+2+3+4

→ sum of n element

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

$\frac{3 \times (3+1)^{2}}{\cancel{2}}$ → 6 cubes

how many, require 3 levels as a whole

Σ (required blocks of level) = total blocks

$\boxed{n = 25}$? → level

1 → 1
\+
2 → 3 → $\frac{(n)(n+1)}{2}$
\+
3 → 6 → " "
\+
4 → 10 →
\+
5 → 15

~~1~~ + 3 1 ✓

4 < 25 ✓

4+6 < 25 ✓

10+10 < 25 ✓

20+15 ≮ 25 ✗

} 4 levels

```java
int testcases = 1;
while (testcases-- > 0) {

    int n = sc.nextInt();
    int total = 0;
    int sum = 0;
    int count = 0;
    for (int i = 1; i <= 10000; i++) {
        sum += i;
        total += sum;
        if (total > n) {
            break;
        }
        count++;
    }
    System.out.println(count);

}
```

totally

sum →

C. Phoenix and Balance

time limit per test: 2 seconds

memory limit per test: 256 megabytes

} no of coins is even }

Math + Greedy

better

Phoenix has n coins with weights $2^1, 2^2, \dots, 2^n$. He knows that n is even.

He wants to split the coins into two piles such that each pile has exactly $\frac{n}{2}$ coins and the difference of weights between the two piles is **minimized**. Formally, let a denote the sum of weights in the first pile, and b denote the sum of weights in the second pile. Help Phoenix minimize $|a - b|$, the absolute value of $a - b$.

Input

The input consists of multiple test cases. The first line contains an integer t ($1 \le t \le 100$) — the number of test cases.

The first line of each test case contains an integer n ($2 \le n \le 30$; n is even) — the number of coins that Phoenix has.

Output

For each test case, output one integer — the minimum possible difference of weights between the two piles.

Example

input

```
2
2
4
```

output

```
2
6
```

Note

In the first test case, Phoenix has two coins with weights 2 and 4. No matter how he divides the coins, the difference will be $4 - 2 = 2$.

In the second test case, Phoenix has four coins of weight 2, 4, 8, and 16. It is optimal for Phoenix to place coins with weights 2 and 16 in one pile, and coins with weights 4 and 8 in another pile. The difference is $(2 + 16) - (4 + 8) = 6$.

n is even ?

→ total no of coins will be even

n = 4

$2^1, 2^2$ | $2^3, 2^4$ ⇒ equally

1 odd

n/2 →

2

2

differ

$|(2^1, 2^2) - (2^3, 2^4)|$ = minimum ★

testcase 1 → 2 = n

$2^1, 2^2$ → split it equally in 2 piles

$\left| 2^1 - 2^2 \right| \Rightarrow |4-2| = 2 =$ minimize?

No we cant

test case - 2 , $n = 4$

$2^1 \quad 2^2 \quad 2^3 \quad 2^4$ - All the coins

$\left| 2^1 \; 2^2 - 2^3 \; 2^4 \right|$ = minimized

$\left| 2+4 - 8+16 \right| \Rightarrow$

$|6-24| \Rightarrow 18$ value minimize

Yes

$2^1 \quad 2^2 \quad 2^3 \quad 2^4 \qquad \left| n/2 - n/2 \right|$ = minimum

$\left| 2^4+2^2 - 2^1+2^3 \right| \Rightarrow |20 - 10| = 10$

L S Rest others

L S Rest

$$\left| 2^4 + 2^1 - 2^2 + 2^3 \right| = \left| 18 - 12 \right| = 6$$

4 + 8

$2^1 \; 2^2 \; 2^3 \quad 2^4 \; 2^5 \; 2^6$

Largest difference

smallest

+

[n = 6)

(n = 0)

Smallest/

= 3

largest + smallest − n.

```java
while (testcases-- > 0) {

    int coins = sc.nextInt();

    int req = coins/2;
    long sum1 = 0;
    long sum2 = 0;
    long cur = 2;

    for(int i = 1 ; i <= coins ; i++){

        if(i < req || i == coins){
            sum1 = sum1 + cur;
        }else{
            sum2 = sum2 + cur;
        }

        cur = cur * 2;

    }
    System.out.println(sum1 - sum2);

}
```

maximize

minimum

x sum


```java
while (testcases-- > 0) {

    int coins = sc.nextInt();

    int req = coins/2;
    long sum1 = 0;
    long sum2 = 0;
    long cur = 2;

    for(int i = 1 ; i <= coins ; i++){

        if(i < req || i == coins){
            sum1 = sum1 + cur;
        }else{
            sum2 = sum2 + cur;
        }

        cur = cur * 2;

    }
    System.out.println(sum1 - sum2);

}
```


✓ → sn

✓ → req ✓

B. Queries about less or equal elements

time limit per test: 2 seconds

memory limit per test: 256 megabytes

You are given two arrays of integers a and b. For each element of the second array b_j you should find the number of elements in array a that are less than or equal to the value b_j.

Input

The first line contains two integers n, m ($1 \le n, m \le 2 \cdot 10^5$) — the sizes of arrays a and b.

The second line contains n integers — the elements of array a ($-10^9 \le a_i \le 10^9$).

The third line contains m integers — the elements of array b ($-10^9 \le b_j \le 10^9$).

Output

Print m integers, separated by spaces: the j-th of which is equal to the number of such elements in array a that are less than or equal to the value b_j.

Examples

input

```
5 4
1 3 5 7 9
6 4 2 8
```

→ queries

output

```
3 2 1 4
```

} Ans

input

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

output

```
4 2 4 2 5
```

2 Arrays

a= [1, 3, 5, 7, 9] O(n)

question

b= [6, 4, 2, 8] m

Time

3 2 1 4 → ti

problem with this approach is time

Most →

n – 2×10^5

m – 2×10^5

$m \times n \Rightarrow 2\times10^5 \times 2\times10^5$

$\Rightarrow 4\times10^{10}$ / 2 seconds

↓

2×10^9

$4\times10^{10} >>>> 2\times10^9$

* Searching → Binary Search ?

5 4

1 3 5 7 9

6 4 2 8 → [1 2 4 5 6 7 8] sorted

log n

6 4 2 8

log n << n → mlogn >> mxn Binary

log n → time complexity

$< b_j$

B.S

5 4

1 3 5 7 9

6 4 2 8

upperbound → learn this

lower bound → learn this

1 3 5

6 4 2 8

upperbound
lower bound → learn this

upperbound → Exactly

↳ how many are less than or equal to the current element

* C++ Upperbound (start ind, end ind, arr)

* Java → upperbound — now

* python → left bisect / Right bisect - lw

overall → $\boxed{\text{Sort the array}}$ + $\boxed{\text{Search}}$

nlogn + mlogn — put <<< mxn

Binary search

```java
int testcases = 1;
while (testcases-- > 0) {

    int n = sc.nextInt();
    int m = sc.nextInt();

    int[] arr = inputArray(sc, n);
    int[] queries = inputArray(sc, m);
    Arrays.sort(arr);

    for (int i = 0; i < queries.length; i++) {
        int ans = upperBound(arr, queries[i]);
        out.print(ans + " ");
    }
    out.println("");

}
out.close();
```

E. AvtoBus

time limit per test: 1 second
memory limit per test: 256 megabytes

Spring has come, and the management of the AvtoBus bus fleet has given the order to replace winter tires with summer tires on all buses.

You own a small bus service business and you have just received an order to replace n tires. You know that the bus fleet owns two types of buses: with two axles (these buses have 4 wheels) and with three axles (these buses have 6 wheels).

You don't know how many buses of which type the AvtoBus bus fleet owns, so you wonder how many buses the fleet might have. You have to determine the minimum and the maximum number of buses that can be in the fleet if you know that the total number of wheels for all buses is n.

Input

The first line contains an integer t $(1 \le t \le 1000)$ — the number of test cases. The following lines contain description of test cases.

The only line of each test case contains one integer n $(1 \le n \le 10^{18})$ — the total number of wheels for all buses.

Output

For each test case print the answer in a single line using the following format.

Print two integers x and y $(1 \le x \le y)$ — the minimum and the maximum possible number of buses that can be in the bus fleet.

If there is no suitable number of buses for the given n, print the number -1 as the answer.

Example

input

```
4
4
7
24
998244353998244352
```

output

```
1 1
-1
4 6
166374058999707392 249561088499561088
```

7 → tie

n = 7

2 types of bus

6 tires   4 tires

n = number of tires

maximum

Maximum of bus   minimum

8     $\boxed{4 \text{ or } 6 \text{ tires}}$

8/4 ⇒ 2 buses

8/6 ⇒ 8−6 ⇒ 2 tires × any bus ×

max   8 → 2 max
8 → 2 min

8 → 2 bus of 4 tires

12/4 ⇒ 3

12/6 = 2

12 → 3 buses → max
12 → 2 buses → min

problem statement

24

max → 24/4 ⇒ 6 → max

min → 24/6 ⇒ 4 − min

6 and 4 combined ×

26 tires → Is it possible
→ Is it not

Note

In the first test case the total number of wheels is 4. It means that there is the only one bus with two axles in the bus fleet.

In the second test case it's easy to show that there is no suitable number of buses with 7 wheels in total.

In the third test case the total number of wheels is 24. The following options are possible:

- Four buses with three axles.
- Three buses with two axles and two buses with three axles. → combinations
- Six buses with two axles.

So the minimum number of buses is 4 and the maximum number of buses is 6.

24/6

4 and 6

* 7 tires → buses ✗

4 + 6 = 10 → ✗

→ odd is wrong ✗

* 3 tires/2/1 ✗ → −1

edge cases

26 → fit the axels

4, 6 → combined can add upto any positive number ≥ 4

26 % 4 == 2
26 % 6 == 2
not divisible

26 − 4 − 4 − 4 − 4 − ~~4~~ ⇒ 2 → 6 →

26 % 4 ≠ 2 remainder

$26 \% 4 \neq 2$ remainder

$\frac{26}{4} \Rightarrow \cancel{6}\ 5$ $2+4 \Rightarrow 6 \rightarrow$ remainder

$26 \rightarrow$ maximum possible divides ?

$26 - 4\times 5 \Rightarrow 6 \rightarrow 1+5 \Rightarrow 6$

$\frac{26}{4} \Rightarrow$ Ans ✓ $\rightarrow 6$

$26 - 20 + 6 \rightarrow \frac{26}{4}$ $\frac{26}{4}$ is true $\rightarrow$ maximum

1

$\uparrow 6$

$26-6 \Rightarrow 20-6 \Rightarrow 14-6 \Rightarrow 8/4$

4,6 are magical together

* Max number of 6

1) $x \% 6 == 2 \rightarrow \frac{x-8}{6} + 2$ ✓   sach ✓

2) $x \% 6 == 4 \rightarrow \frac{x-4}{6} + 1$ ✓

$26 \% 6 == 2 \Rightarrow 8 \rightarrow 4\times 2$

$26 \rightarrow 6\times 3 \Rightarrow \cancel{18}\ 24$

3

$26 - 24$ $26 - 18$

$22 \% 6 = 4$

$\frac{22}{6} \Rightarrow 3+1 \Rightarrow 4$

$22 \% 6 \Rightarrow 4$

$$\begin{array}{r} 26 \\ -24 \\ \hline 2 \end{array}$$ tires ✗

$$\begin{array}{r} 26 \\ -18 \\ \hline 8 \end{array} \rightarrow 2$$ ✓

6

22%6 ⇒ 4

```java
int testcases = sc.nextInt();
while (testcases-- > 0) {

    long n = sc.nextLong();

    if (n < 4 || (n & 1) == 1) {
        out.println(-1);
    } else {
        long min = n / 6;
        if (n % 6 != 0) {
            min++;
        }
        long max = n / 4;
        out.println(min + " " + max);
    }
```

how this works.

D. Maximum Sum

time limit per test: 2 seconds

memory limit per test: 256 megabytes

You are given an array $a_1, a_2, \ldots, a_n$, where all elements are different.

You have to perform **exactly** k operations with it. During each operation, you do **exactly one** of the following two actions (you choose which to do yourself):

- find **two minimum elements** in the array, and delete them;
- find **the maximum element** in the array, and delete it.

You have to calculate the maximum possible sum of elements in the resulting array.

Input

The first line contains one integer t ($1 \le t \le 10^4$) — the number of test cases.

Each test case consists of two lines:

- the first line contains two integers n and k ($3 \le n \le 2 \cdot 10^5$; $1 \le k \le 99999$; $2k < n$) — the number of elements and operations, respectively.
- the second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$; all a_i are different) — the elements of the array.

Additional constraint on the input: the sum of n does not exceed $2 \cdot 10^5$.

Output

For each test case, print one integer — the maximum possible sum of elements in the resulting array.

For each test case, print one integer — the maximum possible sum of elements in the resulting array.

Example

input

```
6
5 1
2 5 1 10 6
5 2
2 5 1 10 6
3 1
1 2 3
6 1
15 22 12 10 13 11
6 2
15 22 12 10 13 11
5 1
999999996 999999999 999999997 999999998 999999995
```

output

```
21
11
3
62
46
3999999986
```

ans

Note

In the first testcase, applying the first operation produces the following outcome:

- two minimums are 1 and 2; removing them leaves the array as $[5, 10, 6]$, with sum 21;
- a maximum is 10; removing it leaves the array as $[2, 5, 1, 6]$, with sum 14.

21 is the best answer.

In the second testcase, it's optimal to first erase two minimums, then a maximum.

Q? ✗

array ✓

k ops ✓

[1, 2, 3, 4]

[3, 4]   [1, 2]

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

[3, 4] ⇒ 7 ✓

[1, 2, 3] ⇒ 6

k=1

[1, 2, 3, 4]

3   4 ✓

Greedy strategy.

* This will fail

6   k=2

max
min

6 $k=2$

max / min

15 22 12 10 13 11

1 → removed

6 2

$\underbrace{\cancel{10} + \cancel{11}}_{21}$ 12 13 15 $\underbrace{22}_{22}$

<

k=1 → one move

[$\underbrace{\cancel{12, + 13}}_{25}$, 15, $\cancel{22}$]

25 ✓ > 22

[12, 13, 15] ⇒ 40

[$\underbrace{10 + 11}_{21}$ 12 13 15 $\cancel{22}$] $\cancel{22}$

21 <

[10, 11, 12, 13, $\cancel{15}$]

[10, 11, 12, 13] ⇒ 46

* Greedy fails

dp → (MLE) / TLE ✓ X All combination

dp → MLE/TLE ✗ All combination

k = 2

[1, 2, 3]

* here is all combination → dp all combos

k = 1

op1	op2
✓	✗
✗	✓

k = 2

op1	op2
✓	✓
✓	✗
✗	✓

- k1

no

0

$\boxed{\text{k+1 combinations}}$ → possibilities → 2^K ✗

[

k - 1

2

k+1

[10, 11, 12, 13, 15, 22]

[10, 11, 12, 13, 15, 22]

[10, 11, 12, 13, 15, 22]

[12, 13, 15, 22]

[12, 13, 15]

- array

[10, 11, 12, 13, 15]

[12, 13, 15]

dp ✗

crux of the problem → k+1

↓

* we need to take the max of all the possible combinations (i.e k+1)

* We can calculate the remaining sum with O(1) time complexity using prefix sums

```java
int testcases = sc.nextInt();
while (testcases-- > 0) {

    int n = sc.nextInt();
    int k = sc.nextInt();
    int[] arr = new int[n];
    long[] prefix = new long[n + 1];

    for (int i = 0; i < n; i++) {
        arr[i] = sc.nextInt();

    }

    Arrays.sort(arr);
    for (int i = 0; i < n; i++) {
        if (i == 0) {
            prefix[i + 1] = arr[i];
        } else {
            prefix[i + 1] = arr[i] + prefix[i];
        }
    }

      printArray(prefix, out
      );


    long max = 0;

    for (int i = 0; i <= k; i++) {
        max = Math.max(max, prefix[n - i] - prefix[2 * (k - i)]);
    }

    System.out.println(max);
```