SAMPLE QUESTIONS REPOSITORY

For Infosys Sample Questions

Job Roles: Specialist Programmer L3, L2, L1 & Digital Specialist Engineer

noteswithlove ❤️

14. There is a shop that sells action figures near Monocarp's house. A new set of action figures will be released shortly; this set contains n figures, the i-th figure costs i coins and is available for purchase from day i to day n.

    For each of the n days, Monocarp knows whether he can visit the shop.

Every time Monocarp visits the shop, he can buy any number of action figures which are sold in the shop (of course, he cannot buy an action figure that is not yet available for purchase). If Monocarp buys at least two figures during the same day, he gets a discount equal to the cost of the most expensive figure he buys (in other words, he gets the most expensive of the figures he buys for free).

Monocarp wants to buy exactly one 1-st figure, one 2-nd figure, ..., one n-th figure from the set. He cannot buy the same figure twice. What is the minimum amount of money he has to spend?

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 one integer n ($1 \le n \le 4 \cdot 10^5$) — the number of figures in the set (and the number of days);

- The second line contains a string s ($|s| = n$, each s_i is either 0 or 1). If Monocarp can visit the shop on the i-th day, then s_i is 1; otherwise, s_i is 0.

Additional constraints on the input:

- In each test case, s is 1, so Monocarp is always able to buy all figures during the n-th day;

- The sum of n over all test cases does not exceed $4 \cdot 10^5$.

Output

For each test case, print one integer — the minimum amount of money Monocarp has to spend.

Example

Input

```
4
1
1
6
101101
7
1110001
5
11111
```

Output

```
1
8
18
6
```

Note

In the first test case, Monocarp buys the 1-st figure on the 1-st day and spends 1 coin.

In the second test case, Monocarp can buy the 1-st and the 3-rd figure on the 3-rd day, the 2-nd and the 4-th figure on the 4-th day, and the 5-th and the 6-th figure on the 6-th day. Then, he will spend $1 + 2 + 5 = 8$ coins.

In the third test case, Monocarp can buy the 2-nd and the 3-rd figure on the 3-rd day, and all other figures on the 7-th day. Then, he will spend $1 + 2 + 4 + 5 + 6 = 18$ coins.

Code:

```
#include<bits/stdc++.h>

using namespace std;



const int N = 400043;

char buf[N];



bool can(const string& s, int k)

{

    int n = s.size();

    vector<int> used(n);

    for(int i = n - 1; i >= 0; i--)

        if(k > 0 && s[i] == '1')

        {

            used[i] = 1;

            k--;

        }

    int cur = 0;

    for(int i = 0; i < n; i++)

        if(used[i])

        {

            cur--;
        if(cur < 0) return false;

    }

    else cur++;

    return true;

}


void solve()

{

    int n; scanf("%d", &n);

    scanf("%s", buf); if(n

    == 1)

    {

        puts("1");

        return;

    }

    string s = buf;

    int count_1 = 0;

    for(auto x : s) if (x == '1') count_1++;

    int l = 1;

    int r = count_1 + 1;

    while(r - l > 1)
    {

        int mid = (l + r) / 2;

        if(can(s, mid))

        l = mid;

        else

            r = mid;

    }

    long long ans = 0;

    for(int i = n - 1; i >= 0; i--)

           if(s[i] == '1' && l > 0)

            l--;

        else

               ans += (i + 1);

    printf("%lld\n", ans);

}


int main()

{

    int t;

    scanf("%d", &t);

    for(int i = 0; i < t; i++)

           solve();

}
```

15. Your University has a large auditorium and today you are on duty there. There will be n lectures today — all from different lecturers, and your current task is to choose in which order ord they will happen.

    Each lecturer will use one marker to write something on a board during their lecture. Unfortunately, markers become worse the more you use them and lecturers may decline using markers which became too bad in their opinion.

Formally, the i-th lecturer has their acceptance value a_i which means they will not use the marker that was used at least in a_i lectures already and will ask for a replacement. More specifically:

- Before the first lecture you place a new marker in the auditorium;
- Before the ord -th lecturer (in the order you've chosen) starts, they check the quality of the marker and if it was used in at least $a\_\{ord\ \}$ lectures before, they will ask you for a new marker;
- If you were asked for a new marker, then you throw away the old one, place a new one in the auditorium, and the lecturer gives a lecture.

You know: the better the marker — the easier for an audience to understand what a lecturer has written, so you want to **maximize the number of used markers**. Unfortunately, the higher-ups watch closely how many markers were spent, so you can't just replace markers before each lecture. So, you have to replace markers only when you are asked by a lecturer. The marker is considered used if at least one lecturer used it for their lecture.

You can choose the order ord in which lecturers will give lectures. Find such order that leads to the **maximum possible number of the used markers**.

Input

The first line contains one integer t ($1 \le t \le 100$) — the number of independent tests.

The first line of each test case contains one integer n ($1 \le n \le 500$) — the number of lectures and lecturers.

The second line of each test case contains n integers $a_1, a_2, \ldots, a$ ($1 \le a_i \le n$) — acceptance values of each lecturer.

Output

For each test case, print n integers — the order ord of lecturers which maximizes the number of used markers. The lecturers are numbered from 1 to n in the order of the input. If there are multiple answers, print any of them

Example

Input

```
4

4

1 2 1 2

2

2 1

3

1 1 1

4

2 3 1 3
```

Output

```
4 1 3 2

1 2

3 1 2

4 3 2 1
```

Note

In the first test case, one of the optimal orders is the following:

- The 4-th lecturer comes first. The marker is new, so they don't ask for a replacement;
- The 1-st lecturer comes next. The marker is used once and since $a_1 = 1$, the lecturer asks for a replacement;
- The 3-rd lecturer comes next. The second marker is used once and since $a_3 = 1$, the lecturer asks for a replacement;
- The 2-nd lecturer comes last. The third marker is used once but $a_2 = 2$, so the lecturer uses this marker.

In total, 3 markers are used.

In the second test case, 2 markers are used. In

the third test case, 3 markers are used.

In the fourth test case, 3 markers are used.

Code:

```
fun main() {

    repeat(readLine()!!.toInt()) {

        val n = readLine()!!.toInt()

        val a = readLine()!!.split(" ").map { it.toInt() }
```

```
            .withIndex().sortedBy { it.value }
        var i = 0
        var j = n - 1
        val answer = ArrayList<Int>(n)
        var bc = 0
        while (i <= j) {
            if (bc >= a[i].value) {


                answer += a[i++].index bc =
                1
            } else {
                answer += a[j--].index bc++
            }
        }
        println(answer.joinToString(" ") { (it + 1).toString() })
    }
}
```

16. You are given an array a1,a2,…,…,an, consisting of nnn positive integers.

Initially you are standing at index 1 and have a score equal to a1a_1a1. You can perform two kinds of moves:

- **move right** — go from your current index xxx to x+1 and add ax+1a_{x+1}ax+1 to your score. This move can only be performed if x<nx < nx<n.

- **move left** — go from your current index xxx to x−1x-1x−1 and add ax−1 to your score. This move can only be performed if x>1. Also, you **can't perform two or more moves to the left in a row**.

You want to perform exactly kkk moves. Also, there should be **no more than zzz** moves to the left among them.

What is the maximum score you can achieve?

Input

The first line contains a single integer ttt
 (1≤t≤ 10^4) — the number of testcases.

The first line of each testcase contains three integers n,kn, kn,k and zzz
(2≤n≤10^5,1≤k≤n−1,0≤z≤min(5,k)) — the number of elements in the array, the total number of moves you should perform and the maximum number of moves to the left you can perform.

The second line of each testcase contains nnn integers a1,a2,……,an
 (1≤ai≤ 10^4)— the given array.

It is guaranteed that the **sum of nnn over all testcases does not exceed 3⋅1053 \cdot 10^53⋅105**.

Output

Print ttt integers — for each testcase output the maximum score you can achieve if you make exactly kkk moves in total, no more than zzz of them are to the left and there are no two or more moves to the left in a row.

Example

Input

```
4

5 4 0

1 5 4 3 2

5 4 1

1 5 4 3 2

5 4 4

10 20 30 40 50

10 7 3

4 6 8 2 9 9 7 4 10 9
```

Output

```
15

19

150

56
```

Note

In the first testcase you are not allowed to move left at all. So you make four moves to the right and obtain the score a1+a2+a3+a4+a5

In the second example you can move one time to the left. So we can follow these moves: right, right, left, right. The score will be a1+a2+a3+a2+a3

In the third example you can move four times to the left but it's not optimal anyway, you can just move four times to the right and obtain the score a1+a2+a3+a4+a5.

Code:

```
for _ in range(int(input())):

    n, k, z = map(int, input().split())

    a = [int(x) for x in input().split()]

    ans = 0

    s = 0

    mx = 0

    for i in range(k + 1):

        if i < n - 1:

            mx = max(mx, a[i] + a[i + 1])

        s += a[i]

        if i % 2 == k % 2:

            tmp = (k - i) // 2

            if tmp <= z:
                    ans = max(ans, s + mx * tmp)

    print(ans)
```

17. Phoenix has a string s consisting of lowercase Latin letters. He wants to distribute all the letters of his string into k non-empty strings $a_1, a_2, \ldots, a$ such that every letter of s goes to exactly one of the strings a_i. The strings a_i do not need to be substrings of s. Phoenix can distribute letters of s and rearrange the letters within each string a_i however he wants.

For example, if $s = baba$ and $k = 2$, Phoenix may distribute the letters of his string in many ways, such as:

- ba and ba
- a and abb
- ab and ab
- aa and bb

But these ways are invalid:

- baa and ba
- b and ba
- baba and empty string (a_i should be non-empty)

Phoenix wants to distribute the letters of his string s into k strings $a_1, a_2, \ldots, a$ to **minimize the lexicographically maximum string among them**, i.e., minimize $max(a_1, a_2, \ldots, a)$. Help him find the optimal distribution and print the minimal possible value of $max(a_1, a_2, \ldots, a)$.

Definition

String x is lexicographically less than string y if either:

- x is a prefix of y and $x \neq y$, or
- there exists an index i $(1 \leq i \leq \min(|x|, |y|))$ such that $x_i < y_i$ and for every j $(1 \leq j < i)$, $x = y$.

Here, $|x|$ denotes the length of string x.

Input

The input consists of multiple test cases.
The first line contains an integer t $(1 \leq t \leq 1000)$ — the number of test cases. Each test case consists of two lines:

- The first line of each test case consists of two integers n and k $(1 \leq k \leq n \leq 10^5)$ — the length of string s and the number of non-empty strings into which Phoenix wants to distribute letters of s, respectively.
- The second line of each test case contains a string s of length n consisting only of lowercase Latin letters.

It is guaranteed that the **sum of n over all test cases is ≤ 10^5**.

Output

Print t answers — one per test case. The i-th answer should be the minimal possible value of $\max(a_1, a_2, \ldots, a)$ in the i-th test case.

Example

Input

```
6

4  2 baba
```

5 2 baacb

5 3 baacb

5 3 aaaaa

6 4 aaxxzz

7 1 phoenix

Output

ab abbc b

aa x ehinopx

Note

- In the first test case, one optimal solution is to distribute baba into ab and ab
- In the second test case, one optimal solution is to distribute baacb into abbc and a.
- In the third test case, one optimal solution is to distribute baacb into ac, ab, and b.
- In the fourth test case, one optimal solution is to distribute aaaaa into aa, aa, and a.
- In the fifth test case, one optimal solution is to distribute aaxxzz into az, az, x, and x.
- In the sixth test case, one optimal solution is to distribute phoenix into ehinopx.

Code:

```
#include <bits/stdc++.h>

using namespace std;
```

```
void solve(){ int
  n,k; cin>>n>>k;
  string s;
  cin>>s;
  sort(s.begin(),s.end());
  //if smallest k letters are not all the same, answer is kth smallest letter
  if (s[0]!=s[k-1]){
    cout<<s[k-1]<<endl;
    return;
  }
  cout<<s[0];
  //if remaining letters aren't the same, we append remaining letters to answer
  if (s[k]!=s[n-1]){
    for (int i=k;i<n;i++)
      cout<<s[i];
  }
  else{
    //remaining letters are the same, so we distribute evenly
    for (int i=0;i<(n-k+k-1)/k;i++)
      cout<<s[n-1];
  }
  cout<<endl;
}
```

```
int main(){

  int t; cin>>t;

  while (t--)

    solve();

}
```

18. Petya has a rectangular Board of size n×m.
Initially, k chips are placed on the board, i-th chip is located in the cell at the intersection of sxi-th row and syi-th column.

In one action, Petya can move all the chips to the left, right, down or up by 1 cell. If the

chip was in the (x,y) cell, then after the operation:

- left, its coordinates will be (x,y−1);
- right, its coordinates will be (x,y+1);
- down, its coordinates will be (x+1,y);
- up, its coordinates will be (x−1,y).

If the chip is located by the wall of the board, and the action chosen by Petya moves it towards the wall, then the chip remains in its current position.

Note that several chips can be located in the same cell.

For each chip, Petya chose the position which it should visit. Note that it's not necessary for a chip to end up in this position.

Since Petya does not have a lot of free time, he is ready to do no more than 2nm actions.

You have to find out what actions Petya should do so that each chip visits the position that Petya selected for it at least once. Or determine that it is not possible to do this in 2nm actions.

Input

The first line contains three integers n, m, k ($1 \le n,m,k \le 200$) — the number of rows and columns of the board and the number of chips, respectively.

The next k lines contains two integers each sxi, syi ($1 \le sx_i \le n$, $1 \le sy_i \le m$) — the starting position of the i-th chip.

The next k lines contains two integers each fxi, fyi ($1 \le fx_i \le n$, $1 \le fy_i \le m$) — the position that the i-th chip should visit at least once.

Output

In the first line print the number of operations so that each chip visits the position that Petya selected for it at least once.

In the second line output the sequence of operations. To indicate operations left, right, down, and up, use the characters L, R, D, U respectively.

If the required sequence does not exist, print -1 in the single line.

Examples

Input

```
3 3 2

1 2

2 1

3 3

3 2
```

Output

```
3

DRD
```

Input

```
5 4 3

3 4

3 1

3 3

5 3

1 3

1 4
```

Output

```
9

DDLUUUURR
```

```
Code:

n, m, _ = map(int, input().split())


print(2 * (n - 1) + (n + 1) * (m - 1)) print("U"

* (n - 1) + "L" * (m - 1), end="") for i in

range(n):

    if i != 0:

        print("D", end="")

    if i % 2 == 0:

        print("R" * (m - 1), end="")

    else:

        print("L" * (m - 1), end="")
```

19. Let's call the string beautiful if it does not contain a substring of length at least 2, which is a palindrome. Recall that a palindrome is a string that reads the same way from the first character to the last and from the last character to the first. For example, the strings a, bab, acca, bcabcbacb are palindromes, but the strings ab, abbbaa, cccb are not.

Let's define cost of a string as the minimum number of operations so that the string becomes beautiful, if in one operation it is allowed to change any character of the string to one of the first 3 letters of the Latin alphabet (in lowercase).

You are given a string s of length n, each character of the string is one of the first 3 letters of the Latin alphabet (in lowercase).

You have to answer m queries — calculate the cost of the substring of the string s from li-th to ri-th position, inclusive.

Input

The first line contains two integers n and m ($1 \le n, m \le 2 \cdot 10^5$) — the length of the string s and the number of queries.

The second line contains the string s, it consists of n characters, each character one of the first 3 Latin letters. The following m lines contain two integers li and ri ($1 \le l_i \le r_i \le n$) — parameters of the i-th query.

Output

For each query, print a single integer — the cost of the substring of the string s from li-th to ri-th position, inclusive.

Example

Input

```
5  4 baacb

1  3

1  5

4  5

2  3
```

Output

```
1

2

0

1
```

Note

Consider the queries of the example test.

- in the first query, the substring is baa, which can be changed to bac in one operation;
- in the second query, the substring is baacb, which can be changed to cbacb in two operations;
- in the third query, the substring is **cb**, which can be left unchanged;
- in the fourth query, the substring is **aa**, which can be changed to **ba** in one operation

Code:

```
#include <bits/stdc++.h>


using namespace std;


int main() {

  ios_base::sync_with_stdio(false);

  cin.tie(NULL);

  int n, m;

  cin >> n >> m;

  string s;

  cin >> s;

  vector<vector<int>> pr(6, vector<int>(n + 1));

  string t = "abc";

  int cur = 0;

  do {
    for (int i = 0; i < n; ++i)
          pr[cur][i + 1] = pr[cur][i] + (s[i] != t[i % 3]);
        ++cur;
  } while (next_permutation(t.begin(), t.end()));
  while (m--) {

    int l, r;
    cin >> l >> r;
    int ans = n;
    for (int i = 0; i < 6; ++i)
      ans = min(ans, pr[i][r] - pr[i][l - 1]);
    cout << ans << "\n";
  }
}
```

20. You are given a huge integer aaa consisting of nnn digits (nnn is between 1 and 3 · 1053 \cdot 10^53 · 105, inclusive). It may contain leading zeros.

You can swap two digits on adjacent (neighboring) positions if the swapping digits are of different parity (that is, they have different remainders when divided by 2).

For example, if a=032867235 you can get the following integers in a single operation:

- 302867235 if you swap the first and the second digits;
- 023867235if you swap the second and the third digits;
- 032876235 if you swap the fifth and the sixth digits;
- 032862735 if you swap the sixth and the seventh digits;
- 032867325 if you swap the seventh and the eighth digits.

Note, that you can't swap digits on positions 2 and 4 because the positions are not adjacent. Also, you can't swap digits on positions 3 and 4 because the digits have the same parity.

You can perform any number (possibly, zero) of such operations. Find

the minimum integer you can obtain.

Note that the resulting integer also may contain leading zeros.

Input

The first line contains one integer ttt (1≤t≤ 10^4) — the number of test cases in the input.

The only line of each test case contains the integer aaa, its length nnn is between 1 and 3⋅1053 \cdot 10^53⋅105, inclusive.

It is guaranteed that the sum of all values nnn does not exceed 3⋅1053 \cdot 10^53⋅105.

Output

For each test case print a line — the minimum integer you can obtain.

Example

Input

```
3

0709

1337

246432
```

Output

```
0079

1337

234642
```

Note

In the first test case, you can perform the following sequence of operations (the pair of swapped digits is highlighted):

0709→swap ’7’ and ’0’0079

In the second test case, the initial integer is optimal.

In the third test case you can perform the following sequence of operations:

246432->246342->243642->234642

Code:

```
#include <bits/stdc++.h>


using namespace std;


int t;

string a;


int main() {

      cin >> t;

      for(int tc = 0; tc < t; ++tc){

            cin >> a; string s[2];

            for(auto x : a)

                      s[int(x - '0') & 1] += x;



            reverse(s[0].begin(), s[0].end());

            reverse(s[1].begin(), s[1].end());



            string res = "";

            while(!(s[0].empty() && s[1].empty())){
```

```
            if(s[0].empty()){
                res += s[1].back();
                s[1].pop_back(); continue;
            }
            if(s[1].empty()){
                res += s[0].back();
                s[0].pop_back(); continue;
            }

            if(s[0].back() < s[1].back()){ res +=
                s[0].back(); s[0].pop_back();
            }
            else{
                res += s[1].back();
                s[1].pop_back();
            }
        }

        cout << res << endl;
    }

  return 0;
}
```

3

21. A palindrome is a string ttt which reads the same backward as forward (formally, t[i]=t[∣t∣+1−i] for all i∈[1,∣t∣]). Here ∣t∣ denotes the length of a string ttt. For example, the strings 010, 1001 and 0 are palindromes.

You have n binary strings s1,s2,…,sn(each si consists of zeroes and or ones). You can swap any pair of characters any number of times (possibly, zero). Characters can be either from the same string or from different strings — there are no restrictions.

Formally, in one move you:

choose four integer numbers x,a,y,b such that 1≤x,y≤n and 1≤a≤∣sx∣ and 1≤b≤∣sy∣ (where xxx and yyy are string indices and a and b are positions in strings sx and sy respectively), swap (exchange) the characters sx[a] and sy[b].

What is the maximum number of strings you can make palindromic simultaneously?

Input

The first line contains single integer Q(1≤Q≤50) — the number of test cases.

The first line on each test case contains single integer nnn (1≤n≤50) — the number of binary strings you have.

Next nnn lines contains binary strings s1,s2,…,sn— one per line. It's guaranteed that 1≤∣si∣≤50 and all strings consist of zeroes and/or ones.

Output

Print QQQ integers — one per test case. The iii-th integer should be the maximum number of palindromic strings you can achieve simultaneously performing zero or more swaps on strings from the iii-th test case.

Example

4

1

0

3

```
1110

100110

010101

2

11111

000001

2

001

11100111
```

Output

```
1

2

2

2
```

Note

In the first test case, s1 is palindrome, so the answer is 1.

In the second test case you can't make all three strings palindromic at the same time, but you can make any pair of strings palindromic. For example, let's make
 s1=0110, s2=111111 and s3=010000

In the third test case we can make both strings palindromic. For example, s1=11011 and s2=100001.

In the last test case s2 is palindrome and you can make s1 palindrome, for example, by swapping s1[2]
and s1[3].

Code:

```
fun main() {

    val q = readLine()!!.toInt()

    for (ct in 1..q) {

        val n = readLine()!!.toInt()

        var (odd, evenGood, evenBad) = listOf(0, 0, 0)

        for (i in 1..n) {

            val s = readLine()!!

            when {

                s.length % 2 == 1 -> odd++

                s.count { it == '0' } % 2 == 0 -> evenGood++

                else -> evenBad++

            }

        }

        println(n - if (odd == 0 && evenBad % 2 == 1) 1 else 0)

    }

}
```

22. Recently, Kolya found out that a new movie theatre is going to be opened in his city soon, which will show a new movie every day for nnn days. So, on the day with the number 1≤i≤n1 \leq i \leq n1≤i≤n, the movie theatre will show the premiere of the iii-th movie. Also, Kolya found out the schedule of the movies and assigned the entertainment value to each movie, denoted by aia_iai.

However, the longer Kolya stays without visiting a movie theatre, the larger the decrease in entertainment value of the next movie. That decrease is equivalent to d⋅cntd \cdot \text{cnt}d⋅cnt, where ddd is a predetermined value and cnt\text{cnt}cnt is the number of days since the last visit to the movie theatre. It is also known that Kolya managed to visit another movie theatre a day before the new one opened:

The day with the number 0. So if we visit the movie theatre the first time on the day with the number iii, then cnt\text{cnt}cnt — the number of days since the last visit to the movie theatre will be equal to iii.

For example, if d=2 and a=[3,2,5,4,6], then by visiting movies with indices 1 and 3, cnt value for the day 1 will be equal to 1−0=1 and cnt value for the day 3 will be 3−1=2, so the total entertainment value of the movies will be

a1−d⋅1+a3−d⋅2=3−2⋅1+5−2⋅2=2

Unfortunately, Kolya only has time to visit at most mmm movies. Help him create a plan to visit the cinema in such a way that the total entertainment value of all the movies he visits is maximized.

Input

Each test consists of multiple test cases. The first line contains a single integer ttt (1≤t≤10^4) — the number of test cases. The description of the test cases follows.

The first line of each test case contains three integers n,m,dn, m, dn,m,d (1≤n≤2⋅10^5, 1≤m≤n , 1≤d≤10^9).

The second line of each test case contains nnn integers a1,a2,…,an(-10^9 ≤ai≤10^9) — the entertainment values of the movies.

It is guaranteed that the sum of nnn over all test cases does not exceed 2⋅1052 \cdot 10^52⋅105.

Output

For each test case, output a single integer — the maximum total entertainment value that Kolya can get.

Example

Input

```
6

5 2 2

3 2 5 4 6
```

```
 4  3  2
 1  1  1  1
  6  6  6
 –82  45  1  –77  39  11
  5  2  2
  3  2  5  4  8
  2  1  1
 –1  2
  6  3  2
 –8  8  –2  –1  9  0
```

Output

```
2
0
60
3
0
7
```

Note:

- The first test case is explained in the problem statement.
- In the second test case, it is optimal **not** to visit any movies.
- In the third test case, it is optimal to visit movies with numbers 2, 3, 5, 6, so the total entertainment value of the visited movies will be $45-6\cdot2+1-6\cdot1+39-6\cdot2+11-6\cdot1=60$

Code:

```
#include <bits/stdc++.h>


using namespace std;


#define int long long


int32_t main() {

    int t;

    cin >> t;

    for (int _ = 0; _ < t; ++_) {

        int n, m, d;

        cin >> n >> m >> d;

        vector<int> a(n);

        for (int i = 0; i < n; ++i) {

            cin >> a[i];

        }
        int ans = 0; set<pair<int,
        int>> s; int sum = 0;
        for (int i = 0; i < n; ++i) {
            int cur = sum + a[i] - d * (i + 1);
            ans = max(ans, cur);
if (a[i] > 0) {

                s.insert({a[i], i});
                sum += a[i];
                if (s.size() >= m) {
                    sum -= (s.begin()->first);
                    s.erase(s.begin());
                }
            }
        }
        cout << ans << endl;
    }
    return 0;
}
```

23. Tema decided to improve his ice cream making skills. He has already learned how to make ice cream in a cone using exactly **two balls**.

Before his ice cream obsession, Tema was interested in mathematics. Therefore, he is curious about the

minimum number of balls he needs to have in order to make exactly **n different types** of ice cream.

There are plenty possible ice cream flavours: 1, 2, 3, …
Tema can make **two-ball ice cream** with any flavours (possibly the same).

Two ice creams are considered different if their **sets of ball flavours are different**. For example,

- {1,2} = {2,1}
- but {1,1} ≠ {1,2}

For example, having the following ice cream balls: {1,1,2}, Tema can make only **two types** of ice cream:

{1,1} and {1,2}.

Note:

- Tema **does not need to make all cones at the same time**.
- He can reuse the balls for different cones as long as he has enough of each kind.
- To make a cone {x, x}, he must have at least **two balls** of flavour x.

Input

Each test consists of multiple test cases. The first line contains a single integer t (1≤t≤10^4) — the number of test cases.

Then follow the test cases, each described by a single integer n

(1≤n≤10^18) — the number of different ice cream types Tema wants to make.

Output

For each test case, output a single integer — the **minimum number of balls** Tema needs to buy.

Example

Input

```
5

1

3

6

179

1000000000000000000
```

Output

```
2

3
```

```
4

27

2648956421
```

Code:

```cpp
#include <bits/stdc++.h>

using namespace std;

#define int long long

int32_t main() {
    int q;
    cin >> q;
        while (q--) {
        int n;
        cin >> n;
        int l = 0, r = min<int>(2e9, 2 * n);
        while (r - l > 1) {
            int m = (l + r) >> 1;
            // m = x + y, answer = x + 2 * y
            if (m * (m - 1) / 2 + m < n) {
                l = m;
            } else {
                r = m;
            }
        }
        int y = n - r * (r - 1) / 2;
        if ((r + 1) * r / 2 <= n) {
            cout << min(r + y, r + 1 + n - (r + 1) * r / 2) << "\n";
        } else {
            cout << r + y << "\n";
        }
    }
    return 0;
}
```

24. You have two strings a and b, each of length n. There are at most **10 different characters** in string a. You also have a set Q which is **initially empty**.

You are allowed to apply the following operation **any number of times** on string a:

- Choose an index **i** ($1 \le i \le n$) and a lowercase English letter c.
- Add the character a[i] (before changing) to the set Q.
- Replace a[i] with the character c.

Example:

Let a = "abecca".

- First operation: Choose `i = 3`, `c = 'x'`. Then `a[3] = 'e'` is added to `Q`. So `Q = {e}`, and `a = "abxcca"`.

- Second operation: Choose `i = 6`, `c = 's'`. Then `a[6] = 'a'` is added to `Q`. So `Q = {e, a}`, and `a = "abxccs"`.

You can perform any number of such operations on `a`, but **at the end**, the set `Q` must contain at most **`k` different characters**.

Your goal is to **maximize the number of pairs `(l, r)`** ($1 \le l \le r \le n$) such that `a[l..r] == b[l..r]`
(i.e., the substrings of `a` and `b` are equal from index `l` to `r`, both inclusive).

Input Format

- First line: Integer `t` — number of test cases ($1 \le t \le 10^4$)
- Then for each test case:
  - First line: Two integers `n` and `k` ($1 \le n \le 10^5$, $0 \le k \le 10$)
  - Second line: String `a` of length `n` (max 10 different characters)
  - Third line: String `b` of length `n`

Total sum of all `n` across all test cases does not exceed 10^5.

Output Format

For each test case, print one integer — the maximum number of valid `(l, r)` pairs.

Example Input

```
  6

 3 1 abc abd

3 0 abc abd

3 1 xbb xcd

4  1 abcd

axcb

3  10 abc abd

10   3 lkwhbahuqa

qoiujoncjb
```

Example Output

```
6

3

6

6

6

11
```

Explanation

Case 1:

- Change `a[3]='c'` → `d`, Q = {c} (size ≤ k).
- Now `a == b`, and all `(l, r)` pairs are equal → total pairs = `n*(n+1)/2 = 3*4/2 = 6`.

Case 2:

- No changes allowed (`k=0`), so only substrings that already match are counted.

Code:

```
#include <bits/stdc++.h>

using namespace std;


#define ll long long

#define pb push_back
```

```
#define EL '\n'

#define fastio
std::ios_base::sync_with_stdio(false);cin.tie(NULL);cout.tie(NULL);


string a, b; string

char_list; bool

mark[26];

ll ans, k;


ll count_matching_pair()

{

    ll tot_pair = 0, match_count = 0;


    for(ll i = 0; i < a.size(); i++) {

        if(a[i] == b[i] || mark[ a[i]-'a' ])

            match_count++;

        else {

            tot_pair += match_count*(match_count+1)/2;

            match_count = 0;

        }

    }

    tot_pair += match_count*(match_count+1)/2;
    return tot_pair;
}

void solve(ll pos, ll cnt)
{
    if(cnt > k) return;

    if(pos == char_list.size()) {
        if(cnt == k) ans = max(ans, count_matching_pair());
        return;
    }

    solve(pos+1, cnt);

    mark[ char_list[pos]-'a' ] = 1;
    solve(pos+1, cnt+1);
    mark[ char_list[pos]-'a' ] = 0;
}

int main()
{
    fastio;
```

```
  ll t;

  cin >> t;


  while(t--) {

      ll n; cin >> n >> k;

      cin >> a >> b;

      unordered_set <char> unq;

      for(auto &ch : a) unq.insert(ch);

      char_list.clear();

      for(auto &x : unq) char_list.pb(x);

      k = min(k, (ll)unq.size()); memset(mark,

      0, sizeof mark); ans = 0;

      solve(0, 0);



      cout << ans << EL;

  }

  return 0;

}
```