Mock Questions Set -2

1. Vasya has recently learned to type and log on to the Internet. He immediately entered a chat room and decided to say hello to everybody. Vasya typed the word s. It is considered that Vasya managed to say hello if several letters can be deleted from the typed word so that it resulted in the word "`hello`". For example, if Vasya types the word "`ahhellllloou`", it will be considered that he said hello, and if he types "`hlelo`", it will be considered that Vasya got misunderstood and he didn't manage to say hello. Determine whether Vasya managed to say hello by the given word s.

Input

The first and only line contains the word s, which Vasya typed. This word consisits of small Latin letters, its length is no less that 1 and no more than 100 letters.

Output

If Vasya managed to say hello, print "`YES`", otherwise print "`NO`".

Examples Input

```
ahhellllloou
```

Output

```
YES
```

Input `hlelo`

Output

```
NO
```

<u>Code:</u>

```
#include <iostream>

using namespace std;



int main() {

        // your code goes here string

        s;
        cin>>s;

        string s1="hello";

        int j=0;

        int count=0;

        for(int i=0;i<s.size();i++){

            if(s[i]==s1[j]){ count++;

                j++;

                    }

        }

        if (count==5){

            cout<<"YES"<<endl;

        }

        else{

            cout<<"NO"<<endl;

        }

        return 0;

}
```

Mock Questions Set -2

2. Suneet and Slavic play a card game. The rules of the game are as follows

- Each card has an integer value between **1 and 10**.
- Each player receives **2 cards**, which are face-down (so a player doesn't know their cards).
- The game is **turn-based** and consists exactly of **two turns**.
- In a round, both players pick a random unflipped card and flip it.
- The player who flipped a card with a **strictly greater** number wins the round.
- In case of **equality**, no one wins the round.
- A player wins the game if he wins the **most number of rounds** (i.e., strictly more than the other player). In case of equality, no one wins the game.

Since Suneet and Slavic aren't best friends, you need to calculate the **number of ways the game could happen that Suneet would end up as the winner**.

Input

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

Each of the next t lines contains 4 integers $a1$, $a2$, $b1$, $b2$ (**$1 \le a1, a2, b1, b2 \le 10$**) where:

- $a1$ and $a2$ are the two cards Suneet has,
- $b1$ and $b2$ are the two cards Slavic has.

Output

For each test case, output a single integer — the number of games Suneet would win considering **all possible ways** the game could happen.

Mock Questions Set -2

Example Input

```
5

3  8  2  6

 1  1  1  1

 10  10  2  2

 1  1  10  10

 3  8  7  2
```

Example Output

```
2

0

4

0

2
```

Code:

```
#include <bits/stdc++.h>
using namespace std;
int main() {
        int t;

        cin >> t;
```

Mock Questions Set -2

```
while (t--) {

    int a, b, c, d;

    cin >> a >> b >> c >> d;

    int wins = 0;



    // Scenario 1: Suneet (a, b) vs Slavic (c, d)

    int suneetWins = 0;

    int slavicWins = 0;

    if (a > c) suneetWins++; if (b > d)

    suneetWins++;    if    (a    <    c)

    slavicWins++;

    if (b < d) slavicWins++;

    if (suneetWins > slavicWins) wins++;



    // Scenario 2: Suneet (a, b) vs Slavic (d, c)

    suneetWins = 0;

    slavicWins = 0;

    if (a > d) suneetWins++; if (b > c)

    suneetWins++;    if    (a    <    d)

    slavicWins++;
```

```
if (b < c) slavicWins++;

if (suneetWins > slavicWins) wins++;



// Scenario 3: Suneet (b, a) vs Slavic (c, d)

suneetWins = 0;

slavicWins = 0;

if  (b  >  c)  suneetWins++;  if  (a  >  d)

suneetWins++; if (b < c) slavicWins++;

if (a < d) slavicWins++;

if (suneetWins > slavicWins) wins++;



// Scenario 4: Suneet (b, a) vs Slavic (d, c)

suneetWins = 0;

slavicWins = 0;

if  (b  >  d)  suneetWins++;  if  (a  >  c)

suneetWins++; if (b < d) slavicWins++;

if (a < c) slavicWins++;

if (suneetWins > slavicWins) wins++;
```

```
        cout << wins << endl;

    }

    return 0;

}
```

3. You have a stripe of checkered paper of length **n**. Each cell is either **white ('W')** or **black ('B')**.

   Your task is to find the **minimum number of cells that must be recolored from white to black**

   in
order to have a segment of **k consecutive black cells** on the stripe.

If a segment of **k consecutive black cells already exists**, you should print **0**.

Input

- The first line contains an integer **t** (**$1 \le t \le 10^4$**) — the number of test cases.
- The descriptions of **t** test cases follow.
- Each test case contains:
    - A line with two integers **n** and **k** (**$1 \le k \le n \le 2 \cdot 10^5$**).
    - A line of length **n**, consisting of letters 'W' (white) and 'B' (black), representing the stripe.

It is guaranteed that the **sum of all n** over all test cases does not exceed **$2 \cdot 10^5$**.

Output

For each test case, print a single integer — the **minimum number of white cells ('W')** that need to be recolored to black in order to have **a segment of k consecutive 'B' characters**.

Mock Questions Set -2

Example Input

Copy Edit

```
4
5 3
BBWBW
5 5
BBWBW
5 1
BBWBW
1 1
W
```

Example Output

CopyEdit

```
1
2
0
1
```

Explanation

- **Test case 1**: "BBWBW" — You can flip the 'W' at position 3 to 'B' → "BBBBW" → segment of length 3 of 'B' → only 1 repaint needed.
- **Test case 2**: "BBWBW" — Flip positions 3 and 5 → "BBBBB" → now a full segment of 5 **'B'** → 2 repaints.
- **Test case 3**: Already has at least one **'B'**, and k=1 → no repaints needed.
- **Test case 4**: "W" → must repaint it to **'B'** → 1 repaint.

Code:

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

using namespace std;



int main() {

        // your code goes here int t;

        cin>>t;

        while(t--){ int n,k;

            cin>>n>>k; string

            s; cin>>s;

            int ans=INT_MAX;

            int count=0;
        bool cheak=false;

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

            if(s[i]=='W'){

                count++;

    }

}

if(count==0){

    cout<<0<<endl;

} else{ ans=min(ans,count); int

j=0;

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

            if(s[j]=='W'){

                count--;

    }

            if(s[i]=='W'){

                count++;

    }

            if(count==0){
                cheak==true;

                break;

            }

            j++;

            ans=min(ans,count);

        } ans=min(ans,count);

        if(cheak==true){

            cout<<0<<endl;

        }

        else{

            cout<<ans<<endl;

        }

        }

    }



}
```

4. You are given list of n integers. You can perform the following operations

- Choose an element x from the list.
- **Erase x** from the list.
- **Subtract x** from **all remaining elements** in the list.

This operation **reduces the list size by exactly 1**.

You are also given a **target value k**. Your task is to determine whether it is possible to perform **exactly $n-1$ operations** so that **only one number remains in the list and it is equal to k**.

Input

- The first line contains an integer t (**$1 \le t \le 10^4$**) — the number of test cases.
- The description of t test cases follows.
- For each test case:
  - The first line contains two integers n and k (**$2 \le n \le 2 \cdot 10^5$, $1 \le k \le 10^9$**) — the number of elements in the list and the target value.
  - The second line contains n integers $a_1, a_2, ..., a$ (**$-10^9 \le a_i \le 10^9$**) — the initial list.

It is guaranteed that the **sum of all n over all test cases does not exceed $2 \cdot 10^5$**.

Output

For each test case, print "YES" if you can perform the operations to make the final remaining number equal to k. Otherwise, print "NO".

You may print the result in **any letter case**.

Example Input

```
4

4 5

4 2 2 7

5 4

1 9 1 3 4

2 17

17 0

2 17

18 18
```

Example Output

```
YES NO YES

NO
```

Explanation

Test Case 1:

Initial: [4, 2, 2, 7], Target: 5
One possible sequence:

- Remove 2 → [4, 2, 7] becomes [2, 0, 5]
- Remove 2 → [0, 5] becomes [−2, 3]
- Remove −2 → [3] becomes [5]

So, output is YES.

Test Case 2:

[1, 9, 1, 3, 4] → No sequence of operations leads to 4 → NO.

Test Case 3:

[17, 0] → remove 0 → [17] YES

Test Case 4:

[18, 18] → whatever you remove, the remaining becomes 0 → not 17 → NO.

Code:

```
#include<iostream>

#include<vector>

#include<algorithm> using

namespace std; int main()

{
```

```
int t;

cin >> t;

while(t--) {

        int n, a;

  cin >> n >> a;

  vector<int> v(n);

  for(int& x : v) cin >> x;

   bool ans = false;

   if(n == 1) ans = (v[0] == a);

   else {

          sort(v.begin(), v.end());

          int i = 0;

          int j = 1;

          while(j < n and i < n) {

                 if(v[i] + abs(a) == v[j]) {

                        ans = true;

                        break;

                 }

                 else if(v[i] + abs(a) < v[j]) ++i;

                 else ++j;

          }

   }

   cout << (ans? "YES" : "NO") << '\n';
 }
 }
```

5. Alice and Bob are playing a game. They have n cards numbered from 1 to n. At the beginning of the game, some of these cards are given to Alice, and the rest are given to Bob.

Card with number i beats card with number j if and only if i > j, with one exception: card 1 beats card n.

The game continues as long as each player has at least one card. During each turn, the following occurs:

- Alice chooses one of her cards and places it face up on the table;
- Bob, seeing Alice's card, chooses one of his cards and places it face up on the table;
- if Alice's card beats Bob's card, both cards are taken by Alice. Otherwise, both cards are taken by Bob.

A player can use a card that they have taken during one of the previous turns.

The player who has no cards at the beginning of a turn loses. Determine who will win if both players play optimally.

Input

The first line contains a single integer t ($1 \le t \le 5000$) — the number of test cases.

Each test case consists of two lines:

- The first line contains a single integer n ($2 \le n \le 50$) — the number of cards;
- The second line contains n characters, each either A or B. If the i-th character is A, then card number i is initially given to Alice; otherwise, it is given to Bob.

Additional constraint on the input: in each test case, at least one card is initially given to Alice, and at least one card is initially given to Bob.

Output

For each test case, output Alice if Alice wins with optimal play, or Bob if Bob wins. It can be shown that if both players play optimally, the game will definitely end in a finite number of turns with one of the players winning.

Example

Input

```
8

2

AB

2

BA

4

ABAB

4

BABA

3

BAA

5

AAAAB

5
BAAAB

6

BBBAAA
```

Output

```
Alice Bob

Bob Bob

Alice

Alice Bob

Alice
```

Note

In the first test case, Alice has only one card, and Bob has only one card. Since Alice's card beats Bob's card, she wins after the first turn.

In the second test case, Alice has only one card, and Bob has only one card. Since Bob's card beats Alice's card, he wins after the first turn.

In the third test case, there are two possible game scenarios:

- if Alice plays the card 1 on the first turn, Bob can respond with the card 2 and take both cards. Then, Alice has to play the card 3 on the second turn, and Bob will respond by playing the card 4. Then, he wins;
- if Alice plays the card 3 on the first turn, Bob can respond with the card 4 and take both cards. Then, Alice has to play the card 1, and Bob can respond either with the card 2 or the card 3. Then, he wins.

In the fourth test case, there are two possible game scenarios:

- if Alice plays the card 2 on the first turn, Bob can respond with the card 3 and take both cards. Then, Alice has to play the card 4 on the second turn, and Bob will respond by playing the card 1. Then, he wins;

- if Alice plays the card 4 on the first turn, Bob can respond with the card 1 and take both cards. Then, Alice has to play the card 2, and Bob can respond either with the card 3 or the card 4. Then, he wins.

Code:

```
def beats(n, x, y):
    if x == 0:
        return y == n - 1
    if x == n - 1:
        return y != 0
    return x > y

for _ in range(int(input())):
    n = int(input())
    owner = input() good
    = False
    for i in range(n):
        if owner[i] != 'A':
            continue good_move
        = True for j in
        range(n):
            if owner[j] == 'B' and beats(n, j, i):
                good_move = False
        if good_move:

            good = True if

    good:

        print('Alice')

    else:

        print('Bob')
```

6. On another boring day, Egor got bored and decided to do something. But since he has no friends, he came up with a game to play.

Egor has a deck of n cards, the i-th card from the top has a number $a[i]$ written on it. Egor wants to play a certain number of rounds until the cards run out. In each round, he takes a non-zero number of cards from the top of the deck and finishes the round. If the sum of the numbers on the cards collected during the round is between l and r, inclusive, the round is won; otherwise, it is lost.

Egor knows by heart the order of the cards. Help Egor determine the maximum number of rounds he can win in such a game. Note that Egor is not required to win rounds consecutively.

Input

Each test consists of several test cases.
The first line contains an integer t ($1 \le t \le 10^4$) — the number of test cases. This is followed by a description of the test cases.

The first line of each test case contains three integers n, l, and r ($1 \le n \le 10^5$, $1 \le l \le r \le 10^9$).

The second line of each test case contains n integers $a_1, a_2, \dots, a$ ($1 \le a_i \le 10^9$) — the numbers on the cards from top to bottom.

It is guaranteed that the sum of n for all test cases does not exceed 2×10^5.

Output

For each test case, output a single number — the maximum number of rounds Egor can win.

Example

Input

```
8

5 3 10

2 1 11 3 7

10 1 5

17 8 12 11 7 11 21 13 10 8

3 4 5

3 4 2

8 12 25

10 7 5 13 8 9 12 7

2 3 3

5 2

9 7 9

2 10 5 1 3 7 6 2 3

1 8 10

9

5 5 6

1 4 2 6 4
```

Output

```
3
0
1
4
0
3
1
2
```

Note

In the first test case, Egor can win **3 rounds**:

- In the first round, take the top 2 cards with values 2 and 1 and win, as their sum is 3. After this, the deck will look like this: [11, 3, 7].
- In the second round, take the top card and **lose**, as its value 11 is greater than $r = 10$. After this, the deck will look like this: [3, 7].
- In the third round, take the top card with value 3 and win. After this, the deck will look like this: [7].
- After this, in the fourth round, Egor only has to take the last card in the deck with value 7 and win again.

In the second test case, Egor cannot win any rounds, no matter how hard he tries.

In the third test case, you can take one card in each round, then the first and third rounds will be losing, and the second round will be winning.

In the fourth test case, you can take two cards in each round and always win.

Code:

```
t = int(input())

for T in range(t):

    n, l, r = map(int, input().split())

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

    ans = 0

    cur = 0

    L, R = 0, 0

    while L < n:

        while R < n and cur < l:

            cur += a[R]

            R += 1

        if l <= cur and cur <= r:

            ans += 1

            L = R

            cur = 0

        else:

            cur -= a[L]

            L += 1

    print(ans)
```

7. Polycarp has just finished writing down the lecture on elvish languages. The language this week was "VwV" (pronounced as "uwu"). The writing system of this language consists of only two lowercase Latin letters: 'v' and 'w'

   Unfortunately, Polycarp has written all the lecture in cursive and without any spaces, so the notes look like a neverending sequence of squiggles. To be exact, Polycarp can't tell 'w' apart from 'vv' in his notes as both of them consist of the same two squiggles.

Luckily, his brother Monocarp has better writing habits, so Polycarp managed to take his notes and now wants to make his own notes more readable. To do that he can follow the notes of Monocarp and underline some letters in his own notes in such a way that there is no more ambiguity. If he underlines a 'v', then it can't be mistaken for a part of some 'w', and if the underlines a 'w', then it can't be mistaken for two adjacent letters 'v'.

What is the minimum number of letters Polycarp should underline to make his notes unambiguous?

Input

The first line contains a single integer t ($1 \le t \le 100$) — the number of testcases.

Each of the next t lines contains a non-empty string in VwV language, which consists only of lowercase Latin letters 'v' and 'w'. The length of the string does not exceed 100.

Output

For each testcase print a single integer: the minimum number of letters Polycarp should underline so that there is no ambiguity in his notes.

Example

Input

5 vv v

w vwv

vwvvwv

Output

```
1

0

1

1

3
```

Note

In the first testcase it's enough to underline any of the two letters 'v'.

In the second testcase the letter 'v' is not ambiguous by itself already, so you don't have to underline anything.

In the third testcase you have to underline 'w', so that you don't mix it up with two letters 'v'. In the

fourth testcase you can underline 'w' to avoid all the ambiguity.

In the fifth testcase you can underline both letters 'w' and any of the two letters 'v' between them.

Code:

```
fun main() {

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

        val s = readLine()!!

        val answer = s.count { it == 'w' } + s.split("w").sumOf { it.length / 2
}

        println(answer)

    }

}
```

8. You are given two integers lll and rrr, where l<rl < rl<r. We will add 1 to lll until the result is equal to rrr. Thus, there will be exactly r−lr - lr−l additions performed. For each such addition, let's look at the number of digits that will be changed after it.

For example:

- If l=909l = 909l=909, then adding one will result in 910 and 2 digits will be changed.

- If you add one to l=9l = 9l=9, the result will be 10 and 2 digits will also be changed.

- If you add one to l=489999l = 489999l=489999, the result will be 490000 and 5 digits will be changed.

Changed digits always form a suffix of the result written in the decimal system.

Output the total number of changed digits, if you want to get rrr from lll, adding 1 each time.

Input:

The first line contains an integer ttt (1≤t≤ 10^4). Then ttt test cases follow.

Each test case is characterized by two integers lll and rrr (1≤l<r≤ 10^9).

Output:

For each test case, calculate the total number of changed digits if you want to get rrr from lll, adding one each time.

Example Input:

```
4

1  9

9  10

10  20
1  1000000000
```

Example Output:

```
8

2

11

1111111110
```

Code:

```cpp
#include <iostream>


using namespace std;


void solve () {

  int L, R;

  cin >> L >> R;


  int ans = 0;

  while (L != 0 || R != 0) {

    ans += R - L;

        L /= 10;

        R /= 10;

  }

  cout << ans << '\n';

}
```

```
int main () {

  ios::sync_with_stdio(false);

  cin.tie(0);


  int testc;

  cin >> testc;


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

       solve();

  }

}
```

9. You are given a binary string s consisting of n zeros and ones.

Your task is to divide the given string into the minimum number of subsequences in such a way that each character of the string belongs to exactly one subsequence and each subsequence looks like "010101..." or "101010..." (i.e. the subsequence should not contain two adjacent zeros or ones).

Recall that a subsequence is a sequence that can be derived from the given sequence by deleting zero or more elements without changing the order of the remaining elements. For example, subsequences of "1011101" are "0", "1", "11111", "0111", "101", "1001", but not "000", "101010" and "11100".

You have to answer t independent test cases.

Input

The first line of the input contains one integer t ($1 \le t \le 2 \cdot 10^4$) — the number of test cases. Then t test cases follow.
The first line of the test case contains one integer n ($1 \le n \le 2 \cdot 10^5$) — the length of s.
The second line of the test case contains n characters '0' and '1' — the string s.

It is guaranteed that the sum of n does not exceed $2 \cdot 10^5$ ($\sum n \le 2 \cdot 10^5$).

Output

For each test case, print the answer: in the first line print one integer k ($1 \le k \le n$) — the minimum number of subsequences you can divide the string s to.
 In the second line print n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le k$), where a_i is the number of subsequence the i-th character of s belongs to.

If there are several answers, you can print any.

Example Input:

```
4

4

0011

6

111111

5

10101

8

01010000
```

Example Output:

```
2

1 2 2 1

6

1 2 3 4 5 6

1
```

```
1 1 1 1 1

4

1 1 1 1 1 2 3 4
```

Code:

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


using namespace std;


int main() {

#ifdef _DEBUG

      freopen("input.txt", "r", stdin);

//    freopen("output.txt", "w", stdout);

#endif



      int t;

      cin >> t;

      while (t--) { int n;

            string s;

            cin >> n >> s; vector<int>

            ans(n); vector<int> pos0,

            pos1;

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

```
        int newpos = pos0.size() + pos1.size();
        if (s[i] == '0') {
            if (pos1.empty()) {
                pos0.push_back(newpos);
            } else {
                newpos = pos1.back();
                pos1.pop_back();
                pos0.push_back(newpos);
            }
        } else {
            if (pos0.empty()) {
                pos1.push_back(newpos);
            } else {
                newpos = pos0.back();
                pos0.pop_back();
                pos1.push_back(newpos);
            }
        }
        ans[i] = newpos;
    }
    cout << pos0.size() + pos1.size() << endl; for (auto
    it : ans) cout << it + 1 << " "; cout << endl;
```

```
        }

        return 0;
}
```

10. You are given a permutation p of length n (a permutation of length n is an array of length n in which each integer from 1 to n occurs exactly once).

You can perform the following operation any number of times (possibly zero):

- choose two different elements x and y and erase them from the permutation;
- insert the minimum of x and y into the permutation in such a way that it becomes the first element;
- insert the maximum of x and y into the permutation in such a way that it becomes the last element.

For example, if $p = [1, 5, 4, 2, 3]$ and we want to apply the operation to the elements 3 and 5, then after the first step of the operation, the permutation becomes $p = [1, 4, 2]$; and after we insert the elements, it becomes $p = [3, 1, 4, 2, 5]$.

Your task is to calculate the minimum number of operations described above to sort the permutation p in ascending order (i.e. transform p so that $p1 < p2 < \cdots < pn$).

Input

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

The first line of the test case contains a single integer n ($1 \le n \le 2 \cdot 10^5$) — the number of elements in the permutation.

The second line of the test case contains n distinct integers from 1 to n — the given permutation p.

The sum of n over all test cases doesn't exceed $2 \cdot 10^5$.

Output

For each test case, output a single integer — the minimum number of operations described above to sort the array p in ascending order.

Example Input:

CopyEdit

```
4

5

1 5 4 2 3

3

1 2 3

4

2 1 4 3

6

5 2 4 1 6 3
```

Example Output:

```
2

0

1

3
```

Note

In the first example, you can proceed as follows:

- in the permutation $p = [1,5,4,2,3]$, let's choose the elements 4 and 2, then, after applying the operation, the permutation becomes $p = [2,1,5,3,4]$;
- in the permutation $p = [2,1,5,3,4]$, let's choose the elements 1 and 5, then, after applying operation, the permutation becomes $p = [1,2,3,4,5]$.

Code:

```
#include <bits/stdc++.h>
using namespace std;
int main() {
  int t;
  cin >> t;
  while (t--) {
    int n;
    cin >> n;
    vector<int> pos(n + 1);
    for (int i = 0; i < n; ++i) {
      int x;
      cin >> x;
      pos[x] = i;
    }
    int l = (n + 1) / 2, r = (n + 2) / 2;
    while (l > 0 && (l == r || (pos[l] < pos[l + 1] && pos[r - 1] < pos[r]))) {
            --l;

            ++r;

        }

        cout << (n - r + l + 1) / 2 << '\n';

    }

}
```

11. Isaac begins his training. There are n running tracks available, and the i-th track (1 ≤ i ≤ n) consists of $a[i]$ equal-length sections.

Given an integer u ($1 \le u \le 10^9$), finishing each section can increase Isaac's ability by a certain value, described as follows:

- Finishing the 1-st section increases Isaac's performance by u.
- Finishing the 2-nd section increases Isaac's performance by $u-1$.
- Finishing the 3-rd section increases Isaac's performance by $u-2$.
- …
- Finishing the k-th section (k ≥ 1) increases Isaac's performance by $u + 1 - k$. (The value $u + 1 - k$ can be negative, which means finishing an extra section decreases Isaac's performance.)

You are also given an integer l. You must choose an integer r such that $l \le r \le n$ and Isaac will finish each section of each track $l, l+1, \dots, r$ (that is, a total of ∑ from i = l to r of $a[i]$ sections).

Answer the following question: what is the optimal r you can choose so that the increase in Isaac's performance is **maximum possible**?

If there are multiple r that maximize the increase in Isaac's performance, output the **smallest** r.

To increase the difficulty, you need to answer the question for q different values of l and u.

Input

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

The descriptions of the test cases follow.

The first line contains a single integer n ($1 \le n \le 10^5$).

The second line contains n integers $a_1, a_2, \ldots, a$ ($1 \le a_i \le 10^4$).

The third line contains a single integer q ($1 \le q \le 10^5$).

The next q lines each contain two integers l and u ($1 \le l \le n$, $1 \le u \le 10^9$) — the descriptions for each query.

The sum of n over all test cases does not exceed $2 \cdot 10^5$. The sum of q over all test cases does not exceed $2 \cdot 10^5$.

Output

For each test case, output q integers: the i-th integer contains the optimal r for the i-th query. If there are multiple solutions, output the smallest one.

Example Input

```
5

6

3 1 4 1 5 9

3

1 8

2 7

5 9
```

```
1
10
1
1 1
9
5 10 9 6 8 3 10 7 3
5
8 56
1 12
9 3
1 27
5 45
5
7 9 2 5 2
10
1 37
2 9
3 33
4 32
4 15
2 2
```

```
4  2

2  19

3  7

2  7

10

9  1  6  7  6  3  10  7  3  10

5

10  43

3  23

9  3

6  8

5  14
```

Example Output

```
3  4  5

1

9  2  9  4  9

5  2  5  5  5  2  4  5  4  2

10  6  9  7  7
```

Note

For the 1-st query in the first test case:

By choosing r=3, Isaac finishes a1+a2+a3 = 3+1+4 = 8 sections in total, hence his increase in performance is
$u+(u-1)+\ldots+(u-7) = 8+7+6+5+4+3+2+1 = 36$.

By choosing r=4, Isaac finishes a1+a2+a3+a4 = 3+1+4+1 = 9 sections in total, hence his increase in performance is
$u+(u-1)+\ldots+(u-8) = 8+7+6+5+4+3+2+1+0 = 36$.

Both choices yield the optimal increase in performance, however we want to choose the smallest r. So we choose r = 3.

For the 2-nd query in the first test case, by choosing r = 4, Isaac finishes a2+a3+a4 = 1+4+1 = 6 sections in total, hence his increase in performance is
$u+(u-1)+\ldots+(u-5) = 7+6+5+4+3+2 = 27$. This is the optimal increase in performance.

For the 3-rd query in the first test case:

By choosing r = 5, Isaac finishes a5 = 5 sections in total, hence his increase in performance is
$u+(u-1)+\ldots+(u-4) = 9+8+7+6+5 = 35$.

By choosing r = 6, Isaac finishes a5+a6 = 5+9 = 14 sections in total, hence his increase in performance is
$u+(u-1)+\ldots+(u-13) = 9+8+7+6+5+4+3+2+1+0+(-1)+(-2)+(-3)+(-4) = 35$.

Both choices yield the optimal increase in performance, however we want to choose the smallest r. So we choose r = 5.

Hence the output for the first test case is $3\ 4\ 5$.

Code:

```
#include "bits/stdc++.h"

using namespace std;

#define int long long
#define double long double
```

```
void solve(int tc) {

  int n;

  cin >> n;

  int a[n + 1];

  for(int i = 1; i <= n; i++) cin >> a[i];

  int ps[n + 1];

  ps[0] = 0;

  for(int i = 1; i <= n; i++) ps[i] = ps[i - 1] + a[i];

  int q;

  cin >> q;

  while(q--) {

    int l, u;

    cin >> l >> u;

    int lb = l, rb = n;

    while(lb < rb) {

      int mid = (lb + rb + 1) >> 1; if(ps[mid] -

      ps[l - 1] <= u) lb = mid; else rb = mid - 1;

    }

    int maxu = -1e18, optid;

    for(int i = max(l, lb - 2); i <= min(n, lb + 2); i++) {

      int t = ps[i] - ps[l - 1];

      int ut = (u + (u - t + 1)) * t / 2;
```

```
        if(ut > maxu) { maxu
          = ut; optid = i;
        }
      }
      cout << optid << " ";
    }
}

signed main() {
  int t = 1; cin >> t;
  for(int i = 1; i <= t; i++){
    solve(i);
    cout << "\n";
  }
}
```

12. Bessie the cow has just intercepted a text that Farmer John sent to Burger Queen! However, Bessie is sure that there is a secret message hidden inside.

The text is a string s of lowercase Latin letters. She considers a string t as hidden in string s if t exists as a subsequence of s whose indices form an arithmetic progression. For example, the string aab is hidden in string $aaabb$ because it occurs at indices 1, 3, and 5, which form an arithmetic progression with a common difference of 2.

Bessie thinks that any hidden string that occurs the most times is the secret message. Two occurrences of a subsequence of s are distinct if the sets of indices are different. Help her find the number of occurrences of the secret message!

Input

The first line contains a string s of lowercase Latin letters ($1 \le |s| \le 10^5$) — the text that Bessie intercepted.

Output

Output a single integer — the number of occurrences of the secret message.

Examples **Input**

```
aaabb
```

Output

```
6
```

Input `usaco`

Output

```
1
```

Input `lol`

Output

```
2
```

Note

In the first example, these are all the hidden strings and their index sets:

- `a` occurs at (1), (2), (3)

```
#include <iostream>

using namespace std;



typedef long long ll;

ll arr1[26],arr2[26][26];
```

```
int main(){ string
  S; cin>>S;
  for (int i=0;i<S.length();i++){
    int c=S[i]-'a';
    for (int j=0;j<26;j++)
      arr2[j][c]+=arr1[j];
    arr1[c]++;
  }
  ll ans=0;
  for (int i=0;i<26;i++)
    ans=max(ans,arr1[i]);
  for (int i=0;i<26;i++)
    for (int j=0;j<26;j++)
      ans=max(ans,arr2[i][j]);
  cout<<ans<<endl;
}
```

13. There is a classroom with two rows of computers. There are n computers in each row and each computer has its own grade. Computers in the first row have grades $a_1, a_2, \ldots, a$ and in the second row — $b_1, b_2, \ldots, b$.

Initially, all pairs of neighboring computers in each row are connected by wire (pairs $(i, i+1)$ for all $1 \leq i < n$), so two rows form two independent computer networks.

Your task is to combine them into one common network by connecting one or more pairs of computers from different rows. Connecting the i-th computer from the first row and the j-th computer from the second row costs $|a_i - b_j|$.

You can connect one computer to several other computers, but you need to provide at least a basic fault tolerance: you need to connect computers in such a way that the network stays connected, despite one of its computers failing. In other words, if one computer is broken (no matter which one), the network won't split into two or more parts.

What is the minimum total cost to make a fault-tolerant network?

Input

The first line contains a single integer t ($1 \le t \le 10^4$) — the number of test cases. Next, t cases follow.

The first line of each test case contains the single integer n ($3 \le n \le 2 \cdot 10^5$) — the number of computers in each row.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the grades of computers in the first row.

The third line contains n integers $b_1, b_2, \ldots, b_n$ ($1 \le b_i \le 10^9$) — the grades of computers in the second row.

It’s guaranteed that the total sum of n doesn’t exceed $2 \cdot 10^5$.

Output

For each test case, print a single integer — the minimum total cost to make a fault-tolerant network.

Example

Input

```
2

3

1 10 1
```

$20 \quad 4 \quad 25$

4

$1 \quad 1 \quad 1 \quad 1$

$1000000000 \quad 1000000000 \quad 1000000000 \quad 1000000000$

Output

31

1999999998

Note

In the first test case, it's optimal to connect four pairs of computers:

- computer 1 from the first row with computer 2 from the second row: cost $|1-4| = 3$
- computer 3 from the first row with computer 2 from the second row: cost $|1-4| = 3$
- computer 2 from the first row with computer 1 from the second row: cost $|10-20| = 10$
- computer 2 from the first row with computer 3 from the second row: cost $|10-25| = 15$

In total, $3 + 3 + 10 + 15 = 31$.

In the second test case, it's optimal to connect 1 from the first row with 1 from the second row, and 4 from the first row with 4 from the second row.

Code:

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


using namespace std;
```

```
#define fore(i, l, r) for(int i = int(l); i < int(r); i++)

typedef long long li; const int

INF = int(1e9); int n;
vector<int> a, b;

inline bool read() {
	if(!(cin >> n))
		return false;
	a.resize(n);
	fore (i, 0, n)
		cin >> a[i];
	b.resize(n);
	fore (i, 0, n)
		cin >> b[i];
	return true;
}

int bestCandidate(const vector<int> &vals, int cur) {
	int bst = INF + 10, pos = -1;
```

```
    fore (i, 0, n) {
        if (bst > abs(cur - vals[i])) { bst =
            abs(cur - vals[i]); pos = i;
        }
    }
    return pos;
}

inline void solve() {
    li bst = 10ll * INF;

    vector<int> cds1 = {0, bestCandidate(b, a[0]), n - 1};
    vector<int> cds2 = {0, bestCandidate(b, a[n - 1]), n - 1};

    for (int var1 : cds1) {
        for (int var2 : cds2) {
            li ans = (li)abs(a[0] - b[var1]) + abs(a[n - 1] - b[var2]);

            if (var1 > 0 && var2 > 0)
                ans += abs(b[0] - a[bestCandidate(a, b[0])]);
            if (var1 < n - 1 && var2 < n - 1)
                ans += abs(b[n - 1] - a[bestCandidate(a, b[n - 1])]);
            bst = min(bst, ans);
        }
    }
    cout << bst << endl;
}

int main() {
#ifdef _DEBUG
    freopen("input.txt", "r", stdin);
#endif
    ios_base::sync_with_stdio(false);
    cin.tie(0), cout.tie(0);

    int t; cin >> t;
    while (t--) {
        read();
        solve();
    }
    return 0;
}
```

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 $s = \text{baba}$ 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≤n,m,k≤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≤sxi≤n, 1≤syi≤m) — the starting position of the i-th chip.

The next k lines contains two integers each fxi, fyi (1≤fxi≤n, 1≤fyi≤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;
}
```

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 $45-6\cdot 2+1-6\cdot 1+39-6\cdot 2+11-6\cdot 1=60$

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:

```
#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 ≤ i ≤ 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 ≤ l ≤ r ≤ 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;
```

```cpp
 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;

}
```

25. You are given two binary strings a and b of length n. In each move, the string a is modified in the following way:

An index $\mathbf{i}$ ($1 \le \mathbf{i} \le n$) is chosen uniformly at random. The character $\mathbf{a[i]}$ will be flipped. That is, if $\mathbf{a[i]}$ is 0, it becomes 1, and if $\mathbf{a[i]}$ is 1, it becomes 0.

What is the expected number of moves required to make both strings equal for the first

time? A binary string is a string, in which the character is either 0 or 1.

Input

The first line contains a single integer $\mathbf{t}$ ($1 \le \mathbf{t} \le 10^5$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer n ($1 \le n \le 10^6$) — the length of the

strings. The second line of each test case contains the binary string a of length n.

The third line of each test case contains the binary string b of length n.

It is guaranteed that the sum of n over all test cases does not exceed 10^6.

Output

For each test case, output a single line containing the expected number of moves modulo

998244353. Formally, let $M = 998244353$. It can be shown that the answer can be expressed as

an irreducible
fraction p/q, where p and q are integers and $q \not\equiv 0 \pmod{M}$. Output the integer equal to $p * q^{-1} \bmod M$. In other words, output such an integer x that $0 \le x < M$ and $x * q \equiv p \pmod{M}$.

Example

Input

```
4
1
2
00
00
4
1000
1110
5
01001
10111
1
0
```

Output

```
1

0

665496254

665496277
```

Code:

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

using namespace std;
```

```
#define mod 998244353

#define N 1000005


template<int MOD>

struct ModInt {

  unsigned x;

  ModInt() : x(0) { }

  ModInt(signed sig) : x(sig) {  }

  ModInt(signed long long sig) : x(sig%MOD) { }

  int get() const { return (int)x; }

  ModInt pow(ll p) { ModInt res = 1, a = *this; while (p) { if (p & 1) res *=
a; a *= a; p >>= 1; } return res; }


  ModInt &operator+=(ModInt that) { if ((x += that.x) >= MOD) x -= MOD; return
*this; }

  ModInt &operator-=(ModInt that) { if ((x += MOD - that.x) >= MOD) x -= MOD;
return *this; }

  ModInt &operator*=(ModInt that) { x = (unsigned long long)x * that.x % MOD;
return *this; }

  ModInt &operator/=(ModInt that) { return (*this) *= that.pow(MOD - 2); }


  ModInt operator+(ModInt that) const { return ModInt(*this) += that; } ModInt

  operator-(ModInt that) const { return ModInt(*this) -= that; } ModInt

  operator*(ModInt that) const { return ModInt(*this) *= that; }
  ModInt operator/(ModInt that) const { return ModInt(*this) /= that; }
  bool operator<(ModInt that) const { return x < that.x; }
  friend ostream& operator<<(ostream &os, ModInt a) { os << a.x; return os; }
};
typedef ModInt<998244353> mint;

mint a[N], b[N], c[N], d[N];


int main(){
    ios::sync_with_stdio(false);
    cin.tie(0);

    int t;
    cin >> t;
    while(t--){
        int n;
        cin >> n;

        string s1, s2;
        cin >> s1 >> s2;
     int diff = 0;

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

         diff += s1[i]!=s2[i];

     }



     c[n] = 1, d[n] = 1, a[1] = 1, b[1] = ((mint) n-1)/n;



     for(int i = 2; i <= n; i++){

         a[i] = ((mint) n + a[i-1]*i)/((mint) n - b[i-1]*i);

         b[i] = ((mint) n-i)/((mint) n - b[i-1]*i);

     }



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

         c[i] = ((mint) n + c[i+1]*(n-i))/((mint) n - d[i+1]*(n-i));

         d[i] = (mint) i / ((mint) n - d[i+1]*(n-i));

     }



     mint ans = (c[diff]+d[diff]*a[diff-1])/((mint) 1 - d[diff]*b[diff-1]);



     cout << ans << endl;

 }
```

```
        return 0;

    }
```

26. You are climbing a staircase. It takes n steps to reach the top.

    Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Example 1:

Input: n = 2

Output: 2

Explanation: There are two ways to climb to the top.

1. 1 step + 1 step

2. 2 steps

Example 2:

Input: n = 3

Output: 3

Explanation: There are three ways to climb to the top.

1. 1 step + 1 step + 1 step

2. 1 step + 2 steps

3. 2 steps + 1 step

27. You are given an integer array coin representing coins of different denominations and an integer amount representing a total amount of money.

Return *the fewest number of coins that you need to make up that amount*. If that amount of money cannot be made up by any combination of the coins, return -1.

You may assume that you have an infinite number of each kind of coin.

Example 1:

Input: coins = [1,2,5], amount = 11

Output: 3

Explanation: 11 = 5 + 5 + 1

Example 2:

Input: coins = [2], amount = 3

Output: -1

Example 3:

Input: coins = [1], amount = 0

Output: 0

28**.** Given a string s, find *the longest palindromic subsequence's length in* s.

A subsequence is a sequence that can be derived from another sequence by deleting some or no elements without changing the order of the remaining elements.

Example 1:

Input: s = "bbbab"

Output: 4

Explanation: One possible longest palindromic subsequence is "bbbb".

Example 2:

Input: s = "cbbd"

Output: 2

Explanation: One possible longest palindromic subsequence is "bb".

29. Given two strings word1 and word2, return *the minimum number of operations required to convert word1 to word2*.

You have the following three operations permitted on a word:

- **Insert a character**
- **Delete a character**
- **Replace a character**

Example 1:

Input: word1 = "horse", word2 = "ros"

Output: 3

Explanation:

horse -> rorse (replace 'h' with 'r')

rorse -> rose (remove 'r')

rose -> ros (remove 'e')

Example 2:

Input: word1 = "intention", word2 = "execution"

Output: 5

Explanation:

intention -> inention (remove 't')

inention -> enention (replace 'i' with 'e')

enention -> exention (replace 'n' with 'x')

exention -> exection (replace 'n' with 'c')

exection -> execution (insert 'u')

Constraints:

- 0 <= word1.length, word2.length <= 500
- word1 and word2 consist of lowercase English letters.

30. Alice has n candies, where the i^{th} candy is of type candyType[i]. Alice noticed that she started to gain weight, so she visited a doctor.

The doctor advised Alice to only eat n / 2 of the candies she has (n is always even). Alice likes her candies very much, and she wants to eat the maximum number of different types of candies while still following the doctor's advice.

Given the integer array candyType of length n, return *the maximum number of different types of candies she can eat if she only eats* n / 2 *of them*.

Example 1:

Input: candyType = [1,1,2,2,3,3]

Output: 3

Explanation: Alice can only eat 6 / 2 = 3 candies. Since there are only 3 types, she can eat one of each type.

Example 2:

Input: candyType = [1,1,2,3]

Output: 2

Explanation: Alice can only eat 4 / 2 = 2 candies. Whether she eats types [1,2], [1,3], or [2,3], she still can only eat 2 different types.

Example 3:

Input: candyType = [6,6,6,6]

Output: 1

Explanation: Alice can only eat 4 / 2 = 2 candies. Even though she can eat 2 candies, she only has 1 type.

Constraints:

- n == candyType.length
- 2 <= n <= 10^4
- n is even.
- -10^5 <= candyType[i] <= 10^5

31. You are given a string s. Simulate events at each second i:

- If s[i] == 'E', a person enters the waiting room and takes one of the chairs in it.

- If s[i] == 'L', a person leaves the waiting room, freeing up a chair.

Return the minimum number of chairs needed so that a chair is available for every person who enters the waiting room given that it is initially empty.

Example 1:

Input: s = "EEEEEEE"

Output: 7

Explanation:

After each second, a person enters the waiting room and no person leaves it. Therefore, a minimum of 7 chairs is needed.

Example 2:

Input: s = "ELELEEL"

Output: 2

Explanation:

Let's consider that there are 2 chairs in the waiting room. The table below shows the state of the waiting room at each second.

Second	Event	People in the Waiting Room	Available Chairs
0	Enter	1	1
1	Leave	0	2
2	Enter	1	1
3	Leave	0	2
4	Enter	1	1
5	Enter	2	0
6	Leave	1	1

Example 3:

Input: s = "ELEELEELLL"

Output: 3

Explanation:

Let's consider that there are 3 chairs in the waiting room. The table below shows the state of the waiting room at each second.

Second	Event	People in the Waiting Room	Available Chairs
0	Enter	1	2
1	Leave	0	3
2	Enter	1	2
3	Enter	2	1
4	Leave	1	2
5	Enter	2	1
6	Enter	3	0
7	Leave	2	1
8	Leave	1	2
9	Leave	0	3

Constraints:

- 1 <= s.length <= 50
- s consists only of the letters 'E' and 'L'.
- s represents a valid sequence of entries and exits.

32. Given an array nums of size n, return *the majority element*.

The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

Example 1:

Input: nums = [3,2,3]

Output: 3

Example 2:

Input: nums = [2,2,1,1,1,2,2]

Output: 2

Constraints:

- n == nums.length
- 1 <= n <= 5 * 10^4
- -10^9 <= nums[i] <= 10^9

33. Given two strings s and t of lengths m and n respectively, return *the minimum window substring of* s *such that every character in* t *(including duplicates) is included in the window*. If there is no such substring, return *the empty string* "".

The testcases will be generated such that the answer is unique.

Example 1:

Input: s = "ADOBECODEBANC", t = "ABC"

Output: "BANC"

Explanation: The minimum window substring "BANC" includes 'A', 'B', and 'C' from string t.

Example 2:

Input: s = "a", t = "a"

Output: "a"

Explanation: The entire string s is the minimum window.

Example 3:

Input: s = "a", t = "aa"

Output: ""

Explanation: Both 'a's from t must be included in the window.

Since the largest window of s only has one 'a', return empty string.

Constraints:

- m == s.length
- n == t.length
- 1 <= m, n <= 10^5
- s and t consist of uppercase and lowercase English letters.

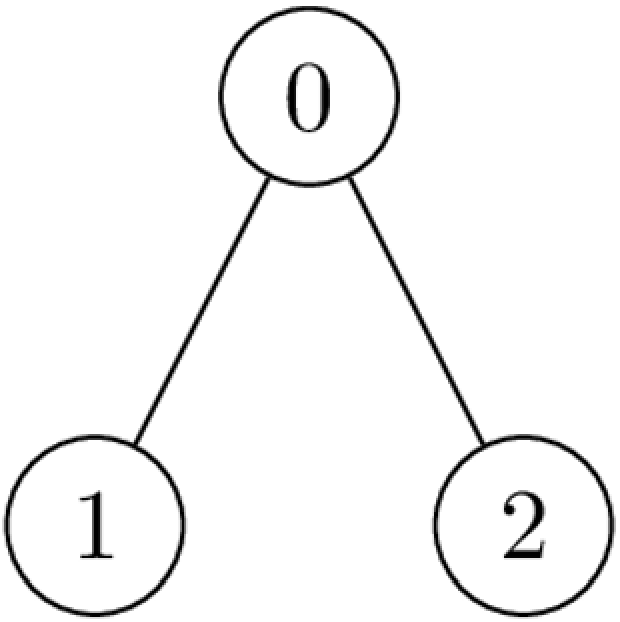
34. There exists an undirected tree with n nodes numbered 0 to n - 1. You are given a 0-indexed 2D integer array edges of length n - 1, where edges[i] = [u_i, v_i] indicates that there is an edge between nodes u_i and v_i in the tree. You are also given a positive integer k, and a 0-indexed array of non-negative integers nums of length n, where nums[i] represents the value of the node numbered i.

Alice wants the sum of values of tree nodes to be maximum, for which Alice can perform the following operation any number of times (including zero) on the tree:

- Choose any edge [u, v] connecting the nodes u and v, and update their values as follows:
  - nums[u] = nums[u] XOR k
  - nums[v] = nums[v] XOR k

Return *the maximum possible sum of the values Alice can achieve by performing the operation any number of times*.

Example 1:



Input: nums = [1,2,1], k = 3, edges = [[0,1],[0,2]]

Output: 6

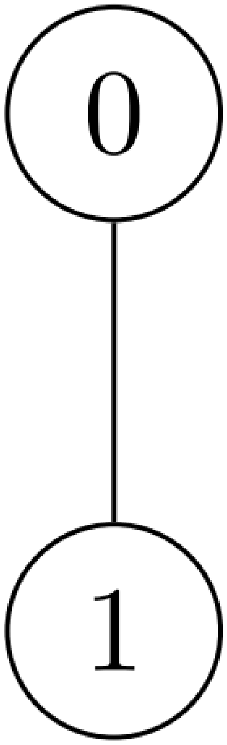
Explanation: Alice can achieve the maximum sum of 6 using a single operation:

- Choose the edge [0,2]. nums[0] and nums[2] become: 1 XOR 3 = 2, and the array nums becomes: [1,2,1] -> [2,2,2].

The total sum of values is 2 + 2 + 2 = 6.

It can be shown that 6 is the maximum achievable sum of values.

Example 2:



Input: nums = [2,3], k = 7, edges = [[0,1]]

Output: 9

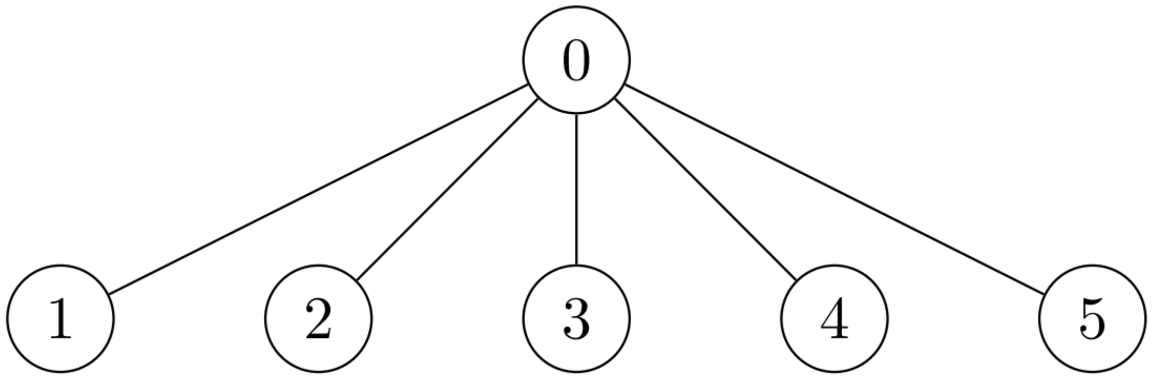
Explanation: Alice can achieve the maximum sum of 9 using a single operation:

- Choose the edge [0,1]. nums[0] becomes: 2 XOR 7 = 5 and nums[1] become: 3 XOR 7 = 4, and the array nums becomes: [2,3] -> [5,4].

The total sum of values is 5 + 4 = 9.

It can be shown that 9 is the maximum achievable sum of values.

Example 3:



Input: nums = [7,7,7,7,7,7], k = 3, edges = [[0,1],[0,2],[0,3],[0,4],[0,5]]

Output: 42

Explanation: The maximum achievable sum is 42 which can be achieved by Alice performing no operations.

Constraints:

- 2 <= n == nums.length <= 2 * 10^4
- 1 <= k <= 10^9
- 0 <= nums[i] <= 10^9
- edges.length == n - 1
- edges[i].length == 2
- 0 <= edges[i][0], edges[i][1] <= n - 1
- The input is generated such that edges represent a valid tree.

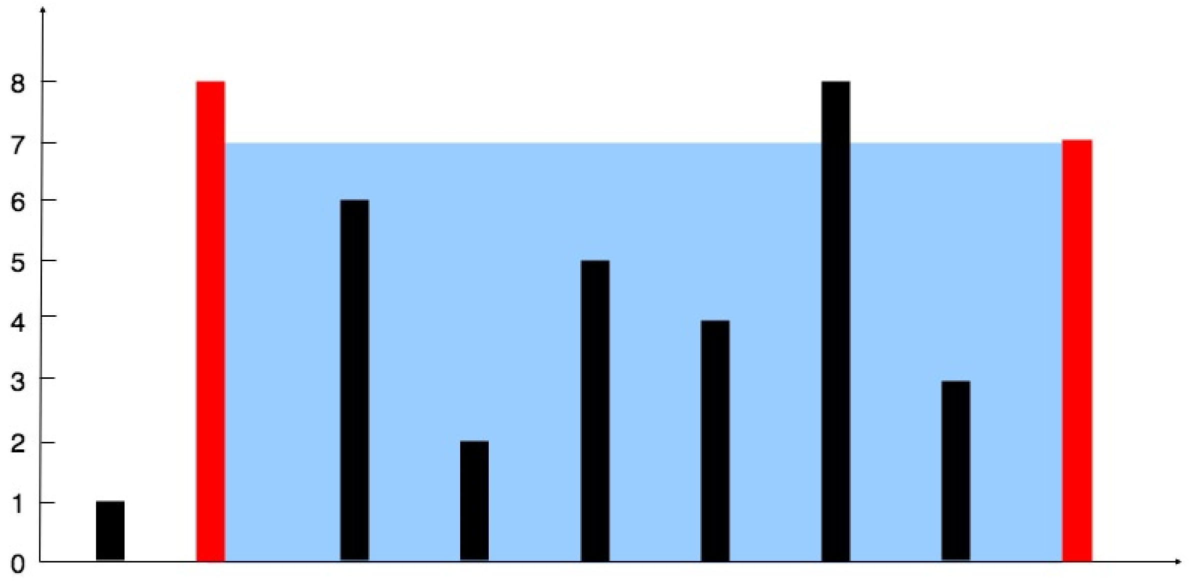
35. You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the i^{th} line are (i, 0) and (i, height[i]).

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return *the maximum amount of water a container can store*.

Notice that you may not slant the container.

Example 1:



Input: height = [1,8,6,2,5,4,8,3,7]

Output: 49

Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Example 2:

Input: height = [1,1]

Output: 1

Constraints:

- n == height.length
- 2 <= n <= 10^5
- 0 <= height[i] <= 10^4

36. You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and it will automatically contact the police if two adjacent houses were broken into on the same night.

Given an integer array nums representing the amount of money of each house, return *the maximum amount of money you can rob tonight without alerting the police*.

Example 1:

Input: nums = [1,2,3,1]

Output: 4

Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).

Total amount you can rob = 1 + 3 = 4.

Example 2:

Input: nums = [2,7,9,3,1]

Output: 12

Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).

Total amount you can rob = 2 + 9 + 1 = 12.

Constraints:

- 1 <= nums.length <= 100
- 0 <= nums[i] <= 400

37. Given an integer array nums, find a subarray that has the largest product, and return *the product*.

The test cases are generated so that the answer will fit in a 32-bit integer.

Example 1:

Input: nums = [2,3,-2,4]

Output: 6

Explanation: [2,3] has the largest product 6.

Example 2:

Input: nums = [-2,0,-1]

Output: 0

Explanation: The result cannot be 2, because [-2,-1] is not a subarray.

Constraints:

- 1 <= nums.length <= 2 * 10^4
- -10 <= nums[i] <= 10
- The product of any subarray of nums is **guaranteed** to fit in a **32-bit** integer.

38**.** You are given an array prices where prices[i] is the price of a given stock on the ith day.

You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock.

Return *the maximum profit you can achieve from this transaction*. If you cannot achieve any profit, return 0.

Example 1:

Input: prices = [7,1,5,3,6,4]

Output: 5

Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.

Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.

Example 2:

Input: prices = [7,6,4,3,1]

Output: 0

Explanation: In this case, no transactions are done and the max profit = 0.

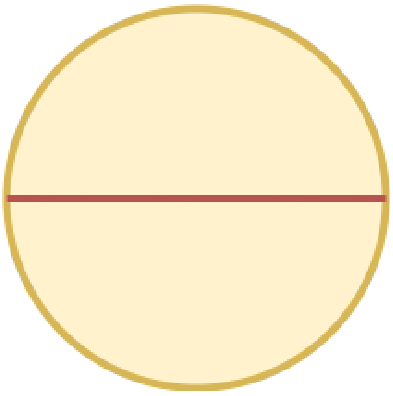
Constraints:

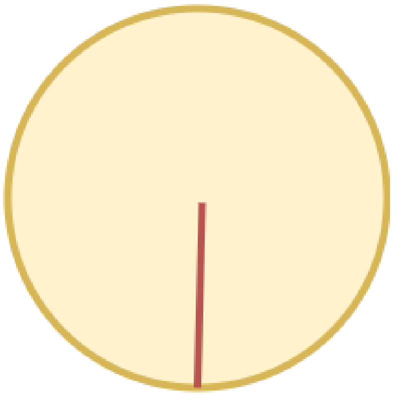
- $1 <= prices.length <= 10^5$
- $0 <= prices[i] <= 10^4$

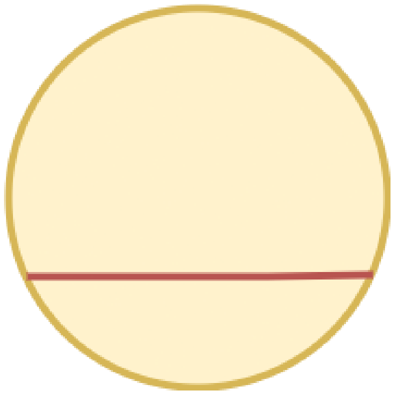
39. A valid cut in a circle can be:

- A cut that is represented by a straight line that touches two points on the edge of the circle and passes through its center, or
- A cut that is represented by a straight line that touches one point on the edge of the circle and its center.

Some valid and invalid cuts are shown in the figures below.

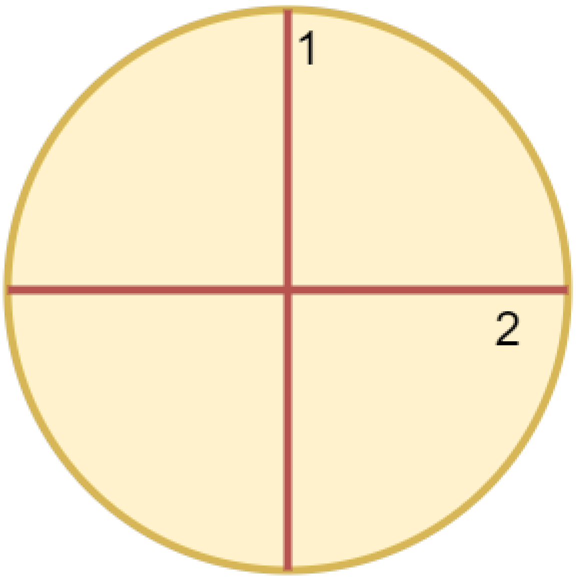

Valid cut


Valid cut


Invalid cut

Given the integer n, return *the **minimum** number of cuts needed to divide a circle into* n *equal slices*.

Example 1:



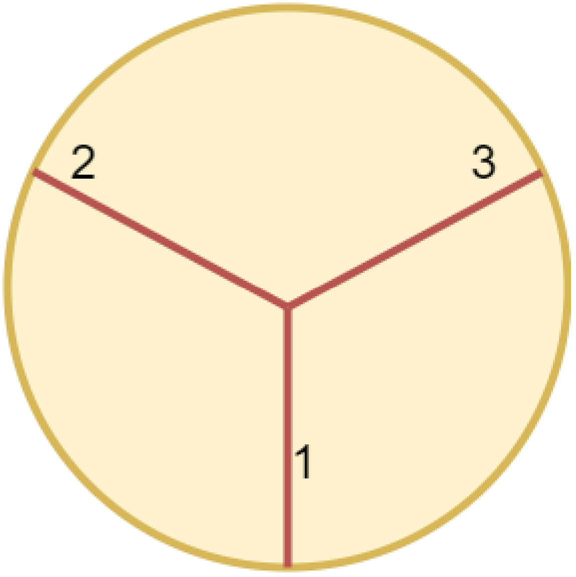
Input: n = 4

Output: 2

Explanation:

The above figure shows how cutting the circle twice through the middle divides it into 4 equal slices.

Example 2:



Input: n = 3

Output: 3

Explanation:

At least 3 cuts are needed to divide the circle into 3 equal slices.

It can be shown that less than 3 cuts cannot result in 3 slices of equal size and shape.

Also note that the first cut will not divide the circle into distinct parts.

Constraints:

- 1 <= n <= 100

40. Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

Example 1:

Input: s = "()"

Output: true

Example 2:

Input: s = "()[]{}"

Output: true

Example 3:

Input: s = "(]"

Output: false

Example 4:

Input: s = "([])"

Output: true

Constraints:

- 1 <= s.length <= 10^4
- s consists of parentheses only '()[]{}'.

41. You have intercepted a secret message encoded as a string of numbers. The message is decoded via the following mapping:

"1" -> 'A'
"2" -> 'B'
...
"25" -> 'Y'
"26" -> 'Z'

However, while decoding the message, you realize that there are many different ways you can decode the message because some codes are contained in other codes ("2" and "5" vs "25").

For example, "11106" can be decoded into:

- "AAJF" with the grouping (1, 1, 10, 6)
- "KJF" with the grouping (11, 10, 6)
- The grouping (1, 11, 06) is invalid because "06" is not a valid code (only "6" is valid).

Note: there may be strings that are impossible to decode.

Given a string s containing only digits, return the number of ways to decode it. If the entire string cannot be decoded in any valid way, return 0.

The test cases are generated so that the answer fits in a 32-bit integer**.**

Example 1:

Input: s = "12"

Output: 2

Explanation:

"12" could be decoded as "AB" (1 2) or "L" (12).

Example 2:

Input: s = "226"

Output: 3

Explanation:

"226" could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).

Example 3:

Input: s = "06"

Output: 0

Explanation:

"06" cannot be mapped to "F" because of the leading zero ("6" is different from "06"). In this case, the string is not a valid encoding, so return 0.

Constraints:

- 1 <= s.length <= 100
- s contains only digits and may contain leading zero(s).

42. You are given an integer array nums. You are initially positioned at the array's first index, and each element in the array represents your maximum jump length at that position.

Return true *if you can reach the last index, or* false *otherwise*.

Example 1:

Input: nums = [2,3,1,1,4]

Output: true

Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.

Example 2:

Input: nums = [3,2,1,0,4]

Output: false

Explanation: You will always arrive at index 3 no matter what. Its maximum jump length is 0, which makes it impossible to reach the last index.

Constraints:

- 1 <= nums.length <= 10^4
- 0 <= nums[i] <= 10^5

43. Given two non-negative integers low and high. Return the *count of odd numbers between* low *and* high *(inclusive)*.

Example 1:

Input: low = 3, high = 7

Output: 3

Explanation: The odd numbers between 3 and 7 are [3,5,7].

Example 2:

Input: low = 8, high = 10

Output: 1

Explanation: The odd numbers between 8 and 10 are [9].

Constraints:

- 0 <= low <= high <= 10^9

44. We have two arrays arr1 and arr2 which are initially empty. You need to add positive integers to them such that they satisfy all the following conditions:

- arr1 contains uniqueCnt1 distinct positive integers, each of which is not divisible by divisor1.
- arr2 contains uniqueCnt2 distinct positive integers, each of which is not divisible by divisor2.
- No integer is present in both arr1 and arr2.

Given divisor1, divisor2, uniqueCnt1, and uniqueCnt2, return *the minimum possible maximum integer that can be present in either array*.

Example 1:

Input: divisor1 = 2, divisor2 = 7, uniqueCnt1 = 1, uniqueCnt2 = 3

Output: 4

Explanation:

We can distribute the first 4 natural numbers into arr1 and arr2.

arr1 = [1] and arr2 = [2,3,4].

We can see that both arrays satisfy all the conditions.

Since the maximum value is 4, we return it.

Example 2:

Input: divisor1 = 3, divisor2 = 5, uniqueCnt1 = 2, uniqueCnt2 = 1

Output: 3

Explanation:

Here arr1 = [1,2], and arr2 = [3] satisfy all conditions.

Since the maximum value is 3, we return it.

Example 3:

Input: divisor1 = 2, divisor2 = 4, uniqueCnt1 = 8, uniqueCnt2 = 2

Output: 15

Explanation:

Here, the final possible arrays can be arr1 = [1,3,5,7,9,11,13,15], and arr2 = [2,6].

It can be shown that it is not possible to obtain a lower maximum satisfying all conditions.

Constraints:

- 2 <= divisor1, divisor2 <= 10^5
- 1 <= uniqueCnt1, uniqueCnt2 < 10^9

- 2 <= uniqueCnt1 + uniqueCnt2 <= 10^9

45. Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string "".

Example 1:

Input: strs = ["flower","flow","flight"]

Output: "fl"

Example 2:

Input: strs = ["dog","racecar","car"]

Output: ""

Explanation: There is no common prefix among the input strings.

Constraints:

- 1 <= strs.length <= 200
- 0 <= strs[i].length <= 200
- strs[i] consists of only lowercase English letters if it is non-empty.

46. You have N meeting requests. Each meeting has a start and end time. Find the maximum number of non-overlapping meetings you can schedule.

Parameters:

First line: N

Next 2×N lines: First N lines = start times, Next N lines = end times

Case#: 1

Input:

3
1
3
0
3
4
6

Output:

2

Case#: 2

Input:

4
0
1
3
5
6
2
4
7

Output:

3

47. You have N projects. Each project requires a certain number of days to complete and gives a profit if finished before or on its deadline. You can only work on one project at a time. Maximize total profit.

Parameters:
First line: N

Next N lines: days_required (number of days needed to complete)

Next N lines: deadline

Next N lines: profit

Case#: 1

Input:
3
2
1
2
2
2
3
100
50
200

Output:
250

Case#: 2

Input:

5
10
20
10
30
40

Output:

4

48. You are given the ages of N people. Find the average age, rounded down to the nearest integer.

Parameters:

First line: N

Next N lines: each person's age

Case#: 1

Input:

4
21

25
19
30

Output:

23

Case#: 2

Input:

3
40
50
60

Output:

50

49. You have N intervals with start and end times and a value for each. Select non-overlapping intervals to maximize the sum of values.

Parameters:
First line: N

Next N lines: start_time

Next N lines: end_time

Next N lines: value

Case#: 1

Input:
3
1
3
0
3
4
6
50
20
100

Output:
150

50. You have N lectures with start and end times. You can attend only one lecture at a time. Find the maximum number of lectures you can attend.

Parameters:
First line: N

Next N lines: start_time

Next N lines: end_time

Case#: 1

Input:
5
1
3
0
5
8
2
4
6
7
9

Output:
3

51. You are given N integers. Find the next greater element for each integer in the list. If none, output -1.

Parameters:

First line: N

Next N lines: values

Case#: 1

Input:

4
4
5
2
10

Output:

5
10
10
-1

Case#: 2

Input:

5
1
3
2
4
2

Output:

3
4
4
-1
-1

52. You are given N numbers. Find the second largest number.

Parameters:

First line: N

Next N lines: numbers

Case#: 1

Input:

5
5
10
20
15
25

Output:

20

Case#: 2

Input:

4
4
3
2
1

Output:

3

53**.** Given N integers, find the maximum sum of a non-adjacent subsequence.

Parameters:

First line: N

Next N lines: array elements

Case#: 1

Input:

5
3
2
5
10
7

Output:

15

Case#: 2

Input:

6
2
1
4
9
3
2

Output:

15

54. You are given N people in a line. People are served in the order they appear, but every Kth person is moved to the end of the queue instead of being served. Find the order in which people are finally served.

Parameters:

First line: N

Next line: K

Next N lines: person ID (numbered 1 to N or any values)

Case#: 1

Input:

5
2
1
2
3
4
5

Output:

1
3
5

2
4

Case#: 2

Input:

4
3
10
20
30
40

Output:

10
20
40
30

55. You have N tasks. Each task has a deadline and a reward. Do at most one task per day (before or on deadline). Maximize the total reward.

Parameters:

First line: N

Next N lines: deadline

Next N lines: reward

Case#: 1

Input:

4
2
1
2
1
50
20
10
40

Output:

90

Case#: 2

Input:

3
1
1
2
100
200
50

Output:250

56. **Burst Balloons**

You are given n balloons, indexed from 0 to n - 1. Each balloon is painted with a number on it represented by an array nums. You are asked to burst all the balloons.

If you burst the ith balloon, you will get nums[i - 1] * nums[i] * nums[i + 1] coins. If i - 1 or i + 1 goes out of bounds of the array, then treat it as if there is a balloon with a 1 painted on it.

Return the maximum coins you can collect by bursting the balloons wisely.

Example 1:

Input: nums = [3,1,5,8]

Output: 167

Explanation:

nums = [3,1,5,8] --> [3,5,8] --> [3,8] --> [8] --> []

coins = 3*1*5   +   3*5*8   + 1*3*8 + 1*8*1 = 167

Example 2:

Input: nums = [1,5]

Output: 10

Constraints:

n == nums.length

1 <= n <= 300

0 <= nums[i] <= 100

57. **Maximum Candies you can get from Boxes**

You have n boxes labeled from 0 to n - 1. You are given four arrays: status, candies, keys, and containedBoxes where:

status[i] is 1 if the ith box is open and 0 if the ith box is closed,

candies[i] is the number of candies in the ith box,

keys[i] is a list of the labels of the boxes you can open after opening the ith box.

containedBoxes[i] is a list of the boxes you found inside the ith box.

You are given an integer array initialBoxes that contains the labels of the boxes you initially have. You can take all the candies in any open box and you can use the keys in it to open new boxes and you also can use the boxes you find in it.

Return the maximum number of candies you can get following the rules above.

Example 1:

Input: status = [1,0,1,0], candies = [7,5,4,100], keys = [[],[],[1],[]], containedBoxes = [[1,2],[3],[],[]], initialBoxes = [0]

Output: 16

Example 2:

Input: status = [1,0,0,0,0,0], candies = [1,1,1,1,1,1], keys = [[1,2,3,4,5],[],[],[],[],[]], containedBoxes = [[1,2,3,4,5],[],[],[],[],[]], initialBoxes = [0]

Output: 6

Constraints:

n == status.length == candies.length == keys.length == containedBoxes.length

1 <= n <= 1000

status[i] is either 0 or 1.

1 <= candies[i] <= 1000

0 <= keys[i].length <= n

0 <= keys[i][j] < n

All values of keys[i] are unique.

0 <= containedBoxes[i].length <= n

0 <= containedBoxes[i][j] < n

All values of containedBoxes[i] are unique.

Each box is contained in one box at most.

0 <= initialBoxes.length <= n

0 <= initialBoxes[i] < n

58. **FROG JUMP**

A frog is crossing a river. The river is divided into some number of units, and at each unit, there may or may not exist a stone. The frog can jump on a stone, but it must not jump into the water.Given a list of stones positions (in units) in sorted ascending order, determine if the frog can cross the river by landing on the last stone. Initially, the frog is on the first stone and assumes the first jump must be 1 unit.

If the frog's last jump was k units, its next jump must be either k - 1, k, or k + 1 units. The frog can only jump in the forward direction.

Example 1:

Input: stones = [0,1,3,5,6,8,12,17]

Output: true

Example 2:

Input: stones = [0,1,2,3,4,8,9,11]

Output: false

Constraints:

2 <= stones.length <= 2000

0 <= stones[i] <= 231 - 1

stones[0] == 0

stones is sorted in a strictly increasing order.

59. **MIN COST CLIMBING STAIRS**

You are given an integer array cost where cost[i] is the cost of ith step on a staircase. Once you pay the cost, you can either climb one or two steps.

You can either start from the step with index 0, or the step with index 1.

Return the minimum cost to reach the top of the floor.

Example 1:

Input: cost = [10,15,20]

Output: 15

Example 2:

Input: cost = [1,100,1,1,1,100,1,1,100,1]

Output: 6

Constraints:

2 <= cost.length <= 1000

0 <= cost[i] <= 999

60. **CHAMPAGNE TOWER**

We stack glasses in a pyramid, where the first row has 1 glass, the second row has 2 glasses, and so on until the 100th row. Each glass holds one cup of champagne.

Then, some champagne is poured into the first glass at the top. When the topmost glass is full, any excess liquid poured will fall equally to the glass immediately to the left and right of it. When those glasses become full, any excess champagne will fall equally to the left and right of those glasses, and so on. (A glass at the bottom row has its excess champagne fall on the floor.)

For example, after one cup of champagne is poured, the top most glass is full. After two cups of champagne are poured, the two glasses on the second row are half full. After three cups of champagne are poured, those two cups become full - there are 3 full glasses total now. After four cups of champagne are poured, the third row has the middle glass half full, and the two outside glasses are a quarter full, as pictured below.

Now after pouring some non-negative integer cups of champagne, return how full the jth glass in the ith row is (both i and j are 0-indexed.)

Example 1:

Input: poured = 1, query_row = 1, query_glass = 1

Output: 0.00000

Explanation: We poured 1 cup of champange to the top glass of the tower (which is indexed as (0, 0)). There will be no excess liquid so all the glasses under the top glass will remain empty.

Example 2:

Input: poured = 2, query_row = 1, query_glass = 1

Output: 0.50000

Explanation: We poured 2 cups of champange to the top glass of the tower (which is indexed as (0, 0)). There is one cup of excess liquid. The glass indexed as (1, 0) and the glass indexed as (1, 1) will share the excess liquid equally, and each will get half cup of champange.

Example 3:

Input: poured = 100000009, query_row = 33, query_glass = 17

Output: 1.00000

Constraints:

0 <= poured <= 109

0 <= query_glass <= query_row < 100

61. **Minimum Number of Refueling Stops**

A car travels from a starting position to a destination which is target miles east of the starting position.

There are gas stations along the way. The gas stations are represented as an array stations where stations[i] = [positioni, fueli] indicates that the ith gas station is positioni miles east of the starting position and has fueli liters of gas.

The car starts with an infinite tank of gas, which initially has startFuel liters of fuel in it. It uses one liter of gas per one mile that it drives. When the car reaches a gas station, it may stop and refuel, transferring all the gas from the station into the car.

Return the minimum number of refueling stops the car must make in order to reach its destination. If it cannot reach the destination, return -1.

Note that if the car reaches a gas station with 0 fuel left, the car can still refuel there. If the car reaches the destination with 0 fuel left, it is still considered to have arrived.

Example 1:

Input: target = 1, startFuel = 1, stations = []

Output: 0

Explanation: We can reach the target without refueling.

Example 2:

Input: target = 100, startFuel = 1, stations = [[10,100]]

Output: -1

Explanation: We can not reach the target (or even the first gas station).

Example 3:

Input: target = 100, startFuel = 10, stations = [[10,60],[20,30],[30,30],[60,40]]

Output: 2

Explanation: We start with 10 liters of fuel.

We drive to position 10, expending 10 liters of fuel. We refuel from 0 liters to 60 liters of gas.

Then, we drive from position 10 to position 60 (expending 50 liters of fuel),

and refuel from 10 liters to 50 liters of gas. We then drive to and reach the target.

We made 2 refueling stops along the way, so we return 2.

Constraints:

1 <= target, startFuel <= 109

0 <= stations.length <= 500

1 <= positioni < positioni+1 < target

1 <= fueli < 109

62. **Couples Holding Hands**

There are n couples sitting in 2n seats arranged in a row and want to hold hands.

The people and seats are represented by an integer array row where row[i] is the ID of the person sitting in the ith seat. The couples are numbered in order, the first couple being (0, 1), the second couple being (2, 3), and so on with the last couple being (2n - 2, 2n - 1).

Return the minimum number of swaps so that every couple is sitting side by side. A swap consists of choosing any two people, then they stand up and switch seats.

Example 1:

Input: row = [0,2,1,3]

Output: 1

Explanation: We only need to swap the second (row[1]) and third (row[2]) person.

Example 2:

Input: row = [3,2,0,1]

Output: 0

Explanation: All couples are already seated side by side.

Constraints:

2n == row.length

2 <= n <= 30

n is even.

0 <= row[i] < 2n

All the elements of row are unique.

63. **Reduce Array Size to Half**

You are given an integer array arr. You can choose a set of integers and remove all the occurrences of these integers in the array.

Return the minimum size of the set so that at least half of the integers of the array are removed.

Example 1:

Input: arr = [3,3,3,3,5,5,5,2,2,7]

Output: 2

Explanation: Choosing {3,7} will make the new array [5,5,5,2,2] which has size 5 (i.e equal to half of the size of the old array).

Possible sets of size 2 are {3,5},{3,2},{5,2}.

Choosing set {2,7} is not possible as it will make the new array [3,3,3,3,5,5,5] which has a size greater than half of the size of the old array.

Example 2:

Input: arr = [7,7,7,7,7,7]

Output: 1

Explanation: The only possible set you can choose is {7}. This will make the new array empty.

Constraints:

2 <= arr.length <= 105

arr.length is even.

1 <= arr[i] <= 105

64. **CINEMA SEAT ALLOCATION**

A cinema has n rows of seats, numbered from 1 to n and there are ten seats in each row, labelled from 1 to 10 as shown in the figure above.

Given the array reservedSeats containing the numbers of seats already reserved, for example, reservedSeats[i] = [3,8] means the seat located in row 3 and labelled with 8 is already reserved.

Return the maximum number of four-person groups you can assign on the cinema seats. A four-person group occupies four adjacent seats in one single row. Seats across an aisle (such as [3,3] and [3,4]) are not considered to be adjacent, but there is an exceptional case on which an aisle split a four-person group, in that case, the aisle split a four-person group in the middle, which means to have two people on each side.

Example 1:

Input: n = 3, reservedSeats = [[1,2],[1,3],[1,8],[2,6],[3,1],[3,10]]

Output: 4

Explanation: The figure above shows the optimal allocation for four groups, where seats mark with blue are already reserved and contiguous seats mark with orange are for one group.

Example 2:

Input: n = 2, reservedSeats = [[2,1],[1,8],[2,6]]

Output: 2

Example 3:

Input: n = 4, reservedSeats = [[4,3],[1,4],[4,6],[1,7]]

Output: 4

Constraints:

1 <= n <= 10^9

1 <= reservedSeats.length <= min(10*n, 10^4)

reservedSeats[i].length == 2

1 <= reservedSeats[i][0] <= n

1 <= reservedSeats[i][1] <= 10

All reservedSeats[i] are distinct.

65. **MINIMUM NUMBER OF PEOPLE TO TEACH**

On a social network consisting of m users and some friendships between users, two users can communicate with each other if they know a common language.

You are given an integer n, an array languages, and an array friendships where:

There are n languages numbered 1 through n,

languages[i] is the set of languages the ith user knows, and

friendships[i] = [ui, vi] denotes a friendship between the users ui and vi.

You can choose one language and teach it to some users so that all friends can communicate with each other. Return the minimum number of users you need to teach.

Note that friendships are not transitive, meaning if x is a friend of y and y is a friend of z, this doesn't guarantee that x is a friend of z.

Example 1:

Input: n = 2, languages = [[1],[2],[1,2]], friendships = [[1,2],[1,3],[2,3]]

Output: 1

Explanation: You can either teach user 1 the second language or user 2 the first language.

Example 2:

Input: n = 3, languages = [[2],[1,3],[1,2],[3]], friendships = [[1,4],[1,2],[3,4],[2,3]]

Output: 2

Explanation: Teach the third language to users 1 and 3, yielding two users to teach.

Constraints:

2 <= n <= 500

languages.length == m

1 <= m <= 500

1 <= languages[i].length <= n

1 <= languages[i][j] <= n

1 <= ui < vi <= languages.length

1 <= friendships.length <= 500

All tuples (ui, vi) are unique

languages[i] contains only unique values

SHORT QUESTIONS

66. Energy Depletion V2
Given initial energy E and exercises (each usable at most 2 times), find the minimum number needed to make energy ≤ 0. Return -1 if not possible.

67.Hero vs Villains - Variant
Given M heroes with same health H, and list of N villains, find the minimum number of villains to remove from front so heroes win.

68.Mountain Array Transformation
Transform array into a symmetric mountain ([1, 2, 3, 2, 1]), with minimal changes.

69.Balanced Rearrangement Cuts
Cut a string into max identical contiguous pieces using rearranged original string.

70.Smart Swap for Lexicographic Order
Swap at most one pair of elements ≤ K indices apart to get the lexicographically smallest array.

71.Terrain Leveling
Make terrain heights strictly decreasing by digging on days (D) that reduce heights by 2^(D-1).

72.Max Dish Orders
Eat maximum number of dishes from unique types, where next order must be twice the size of the previous.

73.Minimum Jumps to End
Jump from index i up to A[i] steps. Find minimum jumps to reach the end.

74.Longest Good Substring
Find the longest substring with no repeating characters.

75. Two Sum with Distance Constraint
Find if two elements sum to K and their indices differ by at most D.

76. Interval Merge
Given a list of intervals, merge overlapping ones.

77.Rotate & Maximize
Rotate array to maximize the sum of i*A[i].

78.Equi-Split Array
Determine if array can be split into two parts with equal sum.

79.Sum of K Smallest Elements
Find the sum of the K smallest numbers in the array.

80.Binary Ones Flip
Flip at most K zeroes to ones. Return max length of contiguous 1s.

81.Frequency-based String Sort
Sort string characters by frequency (descending).

82. Majority Element
Return the element appearing > n/2 times. Else, return -1.

83.Find Peak in Mountain Array
Return index of peak in a mountain array (increasing then decreasing).

84.Trapping Rain Water
Compute trapped water between bars represented by an array.

85.Custom Sort String
Sort a string using a custom order string.

MCQs (20)

86.What is the time complexity of heap sort?
a) O(n) b) O(n log n) c) O(log n) d) $O(n^2)$

87. Which data structure is best for LRU cache?
a) Stack b) Queue c) HashMap + DLL d) Binary Heap

88.Which sorting algorithm is stable and in-place?
a) QuickSort b) MergeSort c) HeapSort d) SelectionSort

89.Best case time for linear search?
a) O(1) b) O(n) c) O(log n) d) O(n log n)

90.Which uses divide and conquer?
a) BFS b) DFS c) Merge Sort d) Hashing

91.Worst-case of QuickSort?
a) O(n log n) b) $O(n^2)$ c) O(n) d) $O(n^3)$

92.In a min-heap, smallest element is at:
a) Leaf b) Root c) Middle d) Last

93.Sorted array from BST by:
a) Preorder b) Inorder c) Postorder d) Level order

94.Dijkstra's algorithm type:
a) BFS b) DFS c) Greedy d) Backtracking

95.Edges in complete graph with n vertices?
a) n b) n(n-1)/2 c) n^2 d) n+1

96.Not a linear data structure?
a) Array b) Tree c) Stack d) Queue

97.Which follows FIFO?
a) Stack b) Queue c) Heap d) Tree

98.Max children in binary tree node?
a) 1 b) 2 c) 3 d) 4

99.Best for nearly sorted data?
a) Bubble b) Insertion c) Selection d) Heap

100.Stack push function in C++ STL?
a) add() b) insert() c) push() d) put()

101.Which finds connected components?
a) Dijkstra b) DFS c) Prim d) Kruskal

102.Merge sort space complexity?
a) O(1) b) O(log n) c) O(n) d) O(n log n)

103.Worst case for binary search?
a) O(1) b) O(log n) c) O(n) d) $O(n^2)$

104.Not required for hashing?
a) Hash function b) Key c) BST d) Collision handling

105.Output of: print(2**3**2) in Python?
a) 512 b) 64 c) 256 d) 8

Solutions & Explanations

Coding Questions

66. Use greedy: sort exercises by drain, pick highest repeatedly (max 2 times each) until energy ≤ 0.
67. Simulate battle; remove villains from front until all can be defeated by heroes.
68. Construct ideal mountain, count changes to match.
69. Find largest len such that string can be divided into equal parts using original char count.
70. Find smallest A[j] within K distance from A[i] and swap for min lex order.
71. Apply minimum digging steps using bit manipulation (power of twos).
72. Start from a unique dish, double next orders, maximize length.
73. Use greedy/BFS or DP to jump using A[i] as max reach.
74. Use sliding window and hash set.
75. Use hash map to check if target sum exists with index difference ≤ D.
76. Sort intervals by start, merge if overlap.
77. Use prefix sum formula S(i) = i*A[i]; rotate and update in O(n).
78. Check prefix sums for split point where left == right.
79. Sort and sum first K elements.
80. Sliding window with max K flips.
81. Use frequency count + sorting.
82. Boyer-Moore Voting Algorithm or hash map.
83. Binary search for peak in mountain.
84. Use two pointers to compute trapped water.
85. Map char frequency and sort based on order string.

MCQ Answers

86. **b** 87. **c** 88. **b** 89. **a** 90. **c**

91. **b** 92. **b** 93. **b** 94. **c** 95. **b**

96. b 97. **b** 98. **b** 99. **b** 100. **c**

101.b 102. **c** 103. **c** 104. **c** 105. **a**

106. Given an array arr[] of n weights. Find the least weight capacity of a boat to ship all weights within d days.

The ith weight has a weight of arr[i]. Each day, we load the boat with weights given by arr[i]. We may not load more weight than the maximum weight capacity of the ship.

Note: You must load weights on the boat in the given order.

Input 1:
n = 3
arr[] = {1, 2, 1}
d = 2
Output:
3

Input 2:
n = 3
arr[] = {9, 8, 10}
d = 3
Output:
10

Your task is to complete the function leastWeightCapacity() which takes 2 integers n, and d, and an array arr of size n as input and returns the least weight capacity of the boat required.

107. Given N candies and K people. In the first turn, the first person gets 1 candy, the second gets 2 candies, and so on till K people. In the next turn, the first person gets K+1 candies, the second person gets K+2 candies and so on. If the number of candies is less than the required number of candies at every turn, then the person receives the remaining number of candies. Find the total number of candies every person has at the end.

Input 1:
N = 7, K = 4
Output:
1 2 3 1

Input 2:
N = 10, K = 3
Output:
5 2 3

Your task is to complete the function distributeCandies() which takes 2 integers N and K as input and returns a list, where the ith integer denotes the number of candies the ith person gets.

108. Given an array arr[] containing only 0s, 1s, and 2s. Sort the array in ascending order.

You need to solve this problem without utilizing the built-in sort of function.

Input: arr[] = [0, 1, 2, 0, 1, 2]
Output: [0, 0, 1, 1, 2, 2]

Input: arr[] = [0, 1, 1, 0, 1, 2, 1, 2, 0, 0, 0, 1]
Output: [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 2, 2]
Follow up: Could you come up with a one-pass algorithm using only constant extra space

109. Given an array arr[] containing N integers. In one step, any element of the array can either be increased or decreased by one. Find minimum steps required such that the product of the array elements becomes 1.

Input:
N = 3
arr[] = {-2, 4, 0}
Output:
5

Input:
N = 3
arr[] = {-1, 1, -1}
Output:
0

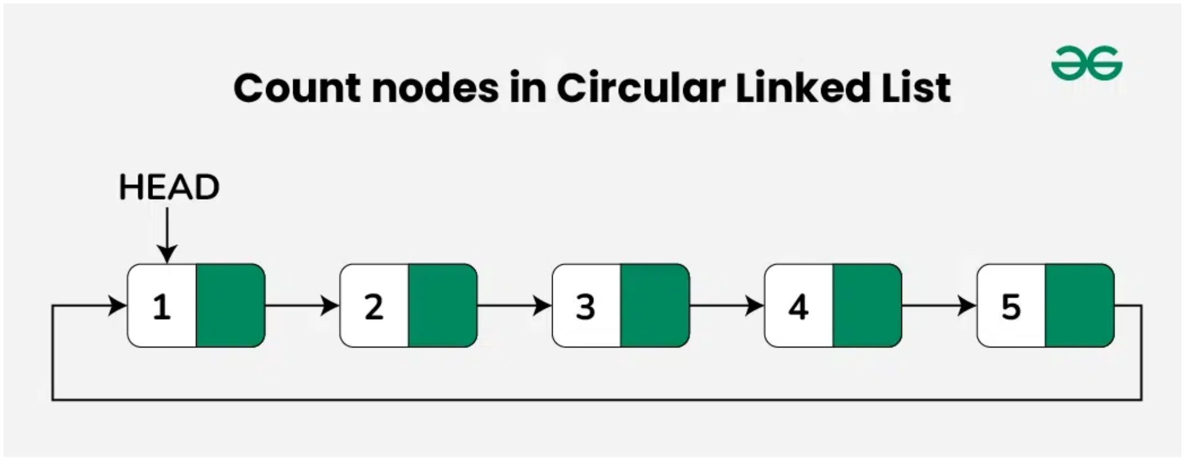
Your task is to complete the function makeProductOne() which takes an integer N and an array arr of size N as input and returns the minimum steps required.

110. Given three coordinate points A, B and C, find the missing point D such that ABCD can be a parallelogram. If there are multiple such points, find the lexicographically smallest coordinate.

Input:
A = (3, 2)
B = (3, 4)
c = (2, 2)
Output:
2.000000 0.000000

Your task is to complete the function findPoint() which takes three list of integers A[], B[] and C[] and return D[] list of two numbers with a precision of 6 decimal places where first element of D[ ] denote x coordinate and second element of D[ ] denote y coordinate.

111. Given a circular linked list. The task is to find the length of the linked list, where length is defined as the number of nodes in the linked list.



112. Given the head of a linked list that may contain a loop. A loop means that the last node of the linked list is connected back to a node in the same list. The task is to remove the loop from the linked list (if it exists).

Custom Input format:

A head of a singly linked list and a pos (1-based index) which denotes the position of the node to which the last node points to. If pos = 0, it means the last node points to null, indicating there is no loop.

The generated output will be true if there is no loop in list and other nodes in the list remain unchanged, otherwise, false.

Input: head = 1 -> 3 -> 4, pos = 2
Output: true

Input: head = 1 -> 8 -> 3 -> 4, pos = 0
Output: true

Input: head = 1 -> 2 -> 3 -> 4, pos = 1
Output: true

113. Given an array arr. Find the number of non-empty subsets whose product of elements is less than or equal to a given integer k.

Input: arr[] = [2, 4, 5, 3], k = 12
Output: 8

Input: arr[] = [9, 8, 3], k = 2

Output: 0

Input: arr[] = [9, 2, 2], k = 2
Output: 2

114. Given a string S which only contains lowercase alphabets. Find the length of the longest substring of S such that the characters in it can be rearranged to form a palindrome.

Input:
S = "aabe"
Output:
3

Input:
S = "adbabd"
Output:
6

Your task is to complete the function longestSubstring() which takes a String S as input and returns the length of largest possible Palindrome.

115. Which will legally declare, construct, and initialize an array?

A. int [] myList = {"1", "2", "3"}; B. int [] myList = (5, 8, 2);

C. int myList [] [] = {4,9,7,0}; D. int myList [] = {4, 3, 7};

116. The function/ method arrayReverse modify the input list by reversing its element. The function/method arrayReverse accepts two arguments - len an integer representing the length of the list and arr, list of integers representing the input list respectively

For example, if the input list arr is {20 30 10 40 50}.

The function/method arrayReverse compiles code successfully but fails to get the desirable result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

Testcase 1:
Input:
4,[4,2,8,6]
Expected return value:
6 8 2 4

Testcase 2:

Input:
3,[11,20,17]
Expected return value:

```
                    17 0 11
```

117. The function/method printCharacterPattern accepts an integer num. It is supposed to print the first num(0<=num<=26) lines of the pattern as shown below:

For example, if num=4, the pattern is

```
a
ab
abc
abcd
```

The function/method printCharacterPattern compiles successfully but fails to get desirable output. Fix the logical errors.

```
void printCharacterPattern(int num)
{
int i,j;
char ch='a';
char print;
for(i=0;i<num;i++)
{
print=ch;
for(j=0;j<=i;j++)
{
printf("%c",ch++);
}
printf("\n");
}
}
```

118. The function/method manchester: for each element in the input array arr, a counter is incremented
if the bit arr[i] is same as arr[i-1]. Then the increment counter value is added to the output array to store the result.
If the bit arr[i] and arr[i-1] are different, then 0 is added to output array. for the first bit in the input array, assume its previous bit to be 0. For example, if arr is {0,1,0,0,1,1,1,0}, then it should print 1 0 0 2 0 3 4 0.

It accepts two arguments-size and arr-list of integers. Each element represents a bit 0 or 1.

It compiles successfully but fails to print the desirable result. Fix the code.

```
void manchester(int size,int *arr)
```

```
{
bool result;
int *res=(int *)malloc(sizeof(int)*size);
int count=0;
for(int i=0;i<size;i++)
{
if(i==0)
result=(arr[i]==0);
else
result=(arr[i]==arr[i-1]);
res[i]=(result)?0:(++count);
}
for(int i=0;i<size;i++)
{
printf("%d",res[i]);
}
}
```

119. The function/method print Fibonacci accepts an integer num, representing a number. The function/method print Fibonacci prints first num numbers of fibonacci series.

For example, given input 5, the function should print the string "01123"(without quotes).

It compiles successfully but fails to give the desirable result for some test cases. Debug the code.

```
void printFibonacci(int num)
{
long num1=0;
long num2=1;
for(int i=1;i<num;++i)
{
printf("%ld",num1);
long sum=num1+num2;
num2=sum;
num1=num2;
}
}
```

120. The function/method countDigits return an integer representing the remainder when the given number is divided by the number of digits in it.
The function/method countDigits accepts an argument-num,an integer representing the given number.

The function/method countDigits compiles code successfully but fails to get the desirable result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.
Test case 1:

Input: 782

Expected return value: 2

Test case 2:

Input: 21340

Expected return value: 0

```
int countDigits(int num)
{
int count=0; while(num!=0)
{
num=num/10; count++;
}
return (num%count);
}
```

121. The function/ method arrayReverse modify the input list by reversing its element. The function/method arrayReverse accepts two arguments - len an integer representing the length of the list and arr, list of integers representing the input list respectively

For example, if the input list arr is {20 30 10 40 50}.

The function/method arrayReverse compiles code successfully but fails to get the desirable result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

Testcase 1:

Input:
4,[4,2,8,6]

Expected return value: 6 8 2 4

Testcase 2:

Input:
3,[11,20,17]

Expected return value: 17 20 11

```
void arrayReverse(int len,int *arr)
{
int i,temp,originalLen=len; for(i=0;i<=originalLen/2;i++)
{
temp=arr[len-1]; arr[len-1]=arr[i]; arr[i]=temp;
len-=1;
}
}
```

122. The function/method printCharacterPattern accepts an integer num. It is supposed to print the first num(0<=num<=26) lines of the pattern as shown below:

For example, if num=4, the pattern is
a ab abc
abcd

The function/method printCharacterPattern compiles successfully but fails to get desirable output. Fix the logical errors.

```
void printCharacterPattern(int num)
{
int i,j;
char ch='a'; char print;
for(i=0;i<num;i++)
{
print=ch; for(j=0;j<=i;j++)
{
printf("%c",ch++);
}
printf("\n");
}
}
```

123. The function/method remove Element prints space separated integer that remains after removing the integer at the given index from the input list.

The function/method remove Element accepts three arguments, size an integer representing the size of input list, index Value an integer representing given index and input List, a list of integers representing the input list.

The function/method remove Element compiles code successfully but fails to get the desirable result for some test cases due to incorrect implementation of function/method remove Element. Your task is to fix the code so that it passes all the test cases.

Note: zero based indexing is followed to access list elements

Test case 1:

Status: wrong Expected:
1 2 3 5 6 7 8 9

Returned
1 2 3 4 4 6 6 8

Test case 1:

Status: correct Expected:
11 23 12 34 54 32

Returned
11 23 12 34 54 32

```
void removeElement(int size,int indexValue,int *inputList)
{
int i,j; if(indexValue<size)
{
for(i=indexValue;i<size;i++)
{
inputList[i]=inputList[i++];
}
for(i=0;i<size-1;i++)
{
printf("%d",inputList[i]);
}
}
else
{
for(i=0;i<size;i++)
{
printf("%d",inputList[i]);
}
```

124. The function calculateMatrixSum(int ** matrix, int m, int n) accepts a two dimensional array matrix of dimensions m, n as input and returns the sum of odd elements whose ith and jth index are same.

The function compiles successfully but fails to return the desired result for some test cases

PROGRAM

```
int calculateMatrixSum(int rows,int columns,int ** matrix))
{
int i,j,sum=0; if((row>0)&&(column>0))
{
for(i=0;i<row;i++)
{
sum=0; for(j=0;j<column;j++)
{
if(i==j)
{
if(matrix[i][j]/2!=0)
sum+=matrix[i][j];
}
}
}
return sum;

}
else
return sum;
}
```

125. The function/method sortArray modify the input list by sorting its elements in descending order. It accepts two arguments-Len, representing the length of the list and arr, a list of integers representing the input list respectively.

It compiles successfully but fails for some test case. Fix the code.

```
void sortArray(int len,int *arr)
{
int i,max,location,j,temp; for(i=0;i<len;i++)
{
max=arr[i]; location=i; for(j=i;j<len;j++)
{
if(max>arr[j])
{
max=arr[j]; location=j;
}
}
temp=arr[i]; arr[i]=arr[location]; arr[location]=temp;
}
}
```

126. The function/method replace Values is modifying the input list in such a way - if the length of the input list is odd, then all the elements of the input list are supposed to be replaced by 1s and in case it is even, the elements should be replaced by 0s.

For example, given the input list [0,1,2], the function will modify the input list like [1 1 1].It accepts two arguments size, an integer representing the size of input list and input list , a list of integers.

The function complies successfully but fails to return the desired results due to logical errors

Your task is to debug the program to pass all the test cases

```
void replaceValues(int size,int *inputList)
{
int i,j; if(size%2==0)
{
i=0;
while(i<size)
{
inputList[i]=0; i+=2;
}
}
else
{
j=0;
while(j<size)
{

inputList[j]=1; j+=2;
}
}
}
```

127. The function product Matrix accepts a two-dimensional array matrix of dimensions m, n as input and returns the sum of odd elements whose ith and jth index are same.

The function compiles line but fails to return the desired result for some test cases

PROGRAM

```
int productMatrix(int rows,int columns,int **matrix)
```

```
{
int result=0;
for(i=0;i<row;i++)
{

for(j=0;j<column;j++)
{
if((i==j)||matrix[i][j]%2!=0)
result*=matrix[i][j];
}
}
if(result<=1)  return 0;

else
return result;
}
```

128. The function maxReplace is supposed to replace every element of the input array arr with the maximum element of arr.

The function complies unsuccessfully due to compilation errors.

Your task is to debug the program to pass all the test cases

PROGRAM:

```
void maxReplace(int size,int &inputList)
{
int i; if(size>0)
{
int max=inputList[0]; for(i=0;i<size;i++)
{
if(max<inputList[i])
{
max=inputList[i];      }
}
}
for(i=0;i<size,i++)
{
inputList[i]=max printf("%d",inputList[i]);
}
}
```

129. The function/method sumElement return an integer representing the sum of the elements in the input array that are greater than twice the input number K and present at the even index. It accepts three augments-size of array, numK representing input number and input array list of integers.

It compiles successfully but fails to return the desirable result.

```
int sumElement(int size, int numK,int *inputArray)
{

int i,sum=0; for(i=0;i<size;i++)
{
if(inputArray[i]>2*numK && i/2==0)
{
sum=inputArray[i];
}
}
return sum;
}
```

130. The function/method manchester: for each element in the input array arr, a counter is incremented if the bit arr[i] is same as arr[i-1]. Then the increment counter value is added to the output array to store the result.

If the bit arr[i] and arr[i-1] are different, then 0 is added to output array. for the first bit in the input array, assume its previous bit to be 0. For example, if arr is {0,1,0,0,1,1,1,0}, then it should print 1 0 0 2 0
3 4 0.

It accepts two arguments-size and arr-list of integers. Each element represents a bit 0 or 1.

It compiles successfully but fails to print the desirable result. Fix the code.

```
void manchester(int size,int *arr)
{
bool result.
int *res=(int *)malloc(sizeof(int)*size); int count=0;
for(int i=0;i<size;i++)
{
if(i==0)
result=(arr[i]==0);
else
result=(arr[i]==arr[i-1]);
res[i]=(result)?0:(++count);
}
for(int i=0;i<size;i++)
{
printf("%d",res[i]);
}
}
```

131. The function/method printFibonacci accepts an integer num, representing a number. The function/method printFibonacci prints first num numbers of Fibonacci series.

For example, given input 5, the function should print the string "01123"(without quotes).

It compiles successfully but fails to give the desirable result for some test cases. Debug the code.

```
void printFibonacci(int num)
{
long num1=0; long num2=1;
for(int i=1;i<num;++i)
{
printf("%ld",num1); long sum=num1+num2; num2=sum; num1=num2;
}
}
```

132. The function/method selectionSortArray performs an in-place selection sort on the given input list which will be sorted in ascending order.

It accepts two arguments-len,an integer representing the length of input list and arr, alist of integers representing the input list respectively.

It compiles successfully but fails to get the desired result.

```
void selectionSortArray(int len,int *arr)
{
int x=0,y=0; for(x=0;x<len;x++)
{
int index_of_min=x; for(y=x;y<len;y++)
{
if(arr[index_of_min]>arr[x]){ index_of_min=y;
}
}
int temp=arr[x]; arr[x]=arr[index_of_min]; arr[index_of_min]=temp;
}
}
```

133. The function/method descendingSortArray performs an in-place sort on the given input list which will be sorted in descending order.

It accepts two argument-len of an array and array of elements.

It compiles successfully but fails to get the desirable output.

```
void descendingSortArray(int len, int *arr)
```

```
{
int small,pos, i,j,temp; for(i=0;i<=len-1;i++)
{
for(j=i;j<len;j++)
{
temp=0; if(arr[i]>arr[j])
{
temp=arr[i]; arr[i]=arr[j]; arr[j]=temp;
}
}
}
}
```

134. The function/method patternPrint accepts an argument num,an integer.

The function/method patternPrint prints num lines in the following pattern.

For example, num=4, the pattern should be

```
1
11
111
1111
```

It compiles successfully but fails to print desirable result.

```
void patternPrint(int num)
{
int print=1,i,j; for(i=0;i<num;i++)
{
for(j=0;j<=i;j++);
{
printf("%d", print);
}
printf("\n");
}
}
```

135. Lisa always forgets her birthday which is on the 5th of july. So,develop a function/method which will be helpful to remember her birthday.

It returns an integer "1" if it is her birthday else return 0. It accepts two argument-month of her birthday, day of her birthday.

It compiles successfully but fails to return the desirable output.

```
int checkBirthday(char *month,int day)
{
if(strcmp(month,"July"))||(day==5)) return 1;
else
return 0;
}
```

136. The function/method sameElementCount returns an integer representing the number of elements of the input list which are even numbers and equal to the element to its right. For eg, if the input list is [4 [4 4 1 8 4 1 1 2 2] then the function should return the output 3 as it has three similar groups i.e., (4,4),
(4,4),(2,2)

It compiles successfully but fails to return desirable output.

```
int sameElementCount(int size,int inputList)
{
int i,count=0; for(i=0;i<size;i++)
{
if((inputList[i]%2==0)&&(inputList[i]==inputList[i++])
{
count++;
}
}
return count;
}
```

137. The function/method manchester: for each element in the input array arr, a counter is incremented if the bit arr[i] is same as arr[i-1]. Then the element of the list is 0, otherwise 1.

for the first bit in the input array, assume its previous bit to be 0. For example, if arr is {0,1,0,0,1,1,1,0}, then it should print {0,1,1,0,1,0,0,1}

It accepts two arguments-size and arr-list of integers. Each element represents a bit 0 or 1.
It compiles successfully but fails to print the desirable result. Fix the code.

```
void manchester(int len,int *arr)
{
int res[len]; res[0]=arr[0]; for(int i=1;i<len;i++)
{
res[i]=(arr[i]==arr[i-1]);
}
for(int i=0;i<len;i++)
{
printf("%d",res[i]);
}
}
```

138. The function/method patternPrint accepts an argument num,an integer.

The function/method patternPrint prints num lines in the following pattern.

For example, num=4, the pattern should be

```
11
1111
111111
11111111
```

It compiles successfully but fails to print desirable result.

```
void drawPrintPattern(int num)
{
int i,j,print=1; for(i=1;i<=num;i++)
{
for(j=1;j<=2*i;j++);
{
printf("%d",print);
}
printf("\n");
}
}
```

139. Reverse the words in a sentence

Input: "I love programming"

Output: "programming love I"

```
def reverse_words(sentence):    return ' '.join(sentence.split()[::-1])print(reverse_words("I love programming"))  # Output: programming love I
```

140. Count vowels in a string

Input: "OpenAI ChatGPT"

Output: 5

```
def count_vowels(s):    return sum(1 for char in s.lower() if char in 'aeiou')print(count_vowels("OpenAI ChatGPT"))  # Output: 5
```

141. Move all zeros to the end of the list (maintaining order)

Input: [0, 1, 0, 3, 12]Output: [1, 3, 12, 0, 0]

```
def move_zeros(nums):    result = [x for x in nums if x != 0]    zeros = [0] * (len(nums) - len(result))    return result + zerosprint(move_zeros([0, 1, 0, 3, 12]))  # Output: [1, 3, 12, 0, 0]
```

142. Find the first non-repeating character in a string

Input: "swiss"Output: "w"

```
from collections import Counterdef first_unique_char(s):    freq = Counter(s)    for char in s:        if freq[char] == 1:            return char    return Noneprint(first_unique_char("swiss"))  # Output: w
```

143. Sudoku Validator

Given a 9x9 Sudoku board, check whether it is valid (not necessarily solved).

```
def is_valid_sudoku(board):    for i in range(9):        row = set()        col = set()        square = set()        for j in range(9):            if board[i][j] != '.' and board[i][j] in row:                return False            row.add(board[i][j])            if board[j][i] != '.' and board[j][i] in col:                return False            col.add(board[j][i])            r, c = 3*(i//3) + j//3, 3*(i%3) + j%3            if board[r][c] != '.' and board[r][c] in square:                return False            square.add(board[r][c])    return True
```

144. Word Break Problem (DP)

Input: s = "leetcode", wordDict = ["leet", "code"]Output: True

```
def word_break(s, wordDict):    dp = [False]*(len(s)+1)    dp[0] = True    for i in range(1, len(s)+1):        for j in range(i):            if dp[j] and s[j:i] in wordDict:                dp[i] = True                break    return dp[-1]print(word_break("leetcode", ["leet", "code"]))  # Output: True
```

145. Write a java program to Swap two numbers without using a third variable

Input: a = 5, b = 10

Program:-

```
public class SwapNumbers {
  public static void main(String[] args) {
      int a = 5, b = 10;
      System.out.println("Before swap: a = " + a + ", b = " + b);
      a = a + b;
      b = a - b;
      a = a - b;
      System.out.println("After swap: a = " + a + ", b = " + b);
    }
 }
```

Output: a = 10, b = 5

146. Write a java program to count vowels in a string

Input: str = "Hello World"

Program:-

```
            public class CountVowels {
    public static void main(String[] args) {
        String str = "Hello World";
        int count = 0;

        str = str.toLowerCase();

        for (int i = 0; i < str.length(); i++) {
            char c = str.charAt(i);
            if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u') {
                count++;
            }
        }

        System.out.println("Number of vowels = " + count);
    }
}
```

Output: Number of vowels = 3

147. Write a java program to reverse each word in a sentence

Input: sentence = "Java is fun"

Program:-

```
      public class ReverseWords {
            public static void main(String[] args) {
             String sentence = "Java is fun";
                 String[] words = sentence.split(" ");

              for (String word : words) {
                  StringBuilder rev = new StringBuilder(word);
                   System.out.print(rev.reverse() + " ");
                  }
             }
            }
```

Output: avaJ si nuf

148. Write a java program to sort an array without using built-in methods

Input: arr = {5, 2, 8, 1, 3}

Program:-

```
    public class SortArray {
  public static void main(String[] args) {
     int[] arr = {5, 2, 8, 1, 3};

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

            System.out.print("Sorted Array: ");
              for (int num : arr) {
                    System.out.print(num + " ");
         }
     }
  }
```

Output: Sorted Array: 1 2 3 5 8

149**.** Write a java program to find all prime numbers up to n using Sieve of Eratosthenes

Input: n = 30

Program:-

```
public class SievePrimes {
 public static void main(String[] args) {
     int n = 30;
    boolean[] prime = new boolean[n + 1];

       for (int i = 2; i <= n; i++) {
        prime[i] = true;
     }

      for (int p = 2; p * p <= n; p++) {
            if (prime[p]) {
          for (int i = p * p; i <= n; i += p) {
            prime[i] = false;
          }
        }
    }

      System.out.print("Prime numbers up to " + n + ": ");
           for (int i = 2; i <= n; i++) {
             if (prime[i]) {
             System.out.print(i + " ");
        }
      }
   }
}
```

Output: Prime numbers up to 30: 2 3 5 7 11 13 17 19 23 29

150. Write a java program to implement a simple LRU (Least Recently Used) cache

Input:

Capacity = 3

Operations:

put(1, 100)

put(2, 200)

put(3, 300)

get(1)   // Access key 1

put(4, 400) // Key 2 should be removed (least recently used)

Program:-

```
 import java.util.*;


public class LRUCache {
private final int capacity;
   private final LinkedHashMap<Integer, Integer> map;


     public LRUCache(int capacity) {
          this.capacity = capacity;
         this.map = new LinkedHashMap<>(capacity, 0.75f, true) {
              protected boolean removeEldestEntry(Map.Entry eldest) {
              return size() > capacity;
              }
          } ;
     }
```

```
    public int get(int key) {
        return map.getOrDefault(key, -1);
    }

    public void put(int key, int value) {
        map.put(key, value);
    }

    public static void main(String[] args) {
        LRUCache cache = new LRUCache(3);

        cache.put(1, 100);
        cache.put(2, 200);
        cache.put(3, 300);
        cache.get(1);
        cache.put(4, 400);

        System.out.println("Cache contents: " + cache.map);
    }
}
```

Output:

Cache contents: {3=300, 1=100, 4=400}

151**.** Count the number of vowels in a string

Input: “hello world”

```
Def count_vowels(s):
   Return sum(1 for char in s.lower() if char in ‘aeiou’)
```

```
Print(count_vowels("hello world"))  # Output: 3
```

152**.** Reverse a given number

Input: 12345

```
Def reverse_number(n):
   Return int(str(n)[::-1])
Print(reverse_number(12345))  # Output: 54321
```

153**.** Find the first non-repeating character in a string

Input: "swiss"

```
From collections import Counter
Def first_non_repeating(s):
   Count = Counter(s)
   For char in s:
      If count[char] == 1:
         Return char
   Return None
Print(first_non_repeating("swiss"))  # Output: 'w'
```

154**.** Merge two sorted arrays into one sorted array

Input: [1,3,5], [2,4,6]

```
Def merge_sorted(arr1, arr2):
   Return sorted(arr1 + arr2)
Print(merge_sorted([1, 3, 5], [2, 4, 6]))  # Output: [1, 2, 3, 4, 5, 6]
```

155**.** Word Break Problem (Dynamic Programming)

Input: “applepenapple”, [“apple”, “pen”]

```
Def word_break(s, wordDict):
   Dp = [False] * (len(s) + 1)
   Dp[0] = True
For I in range(1, len(s)+1):
      For word in wordDict:
         If dp[I – len(word)] and s[i-len(word):i] == word:
            Dp[i] = True
            Break
   Return dp[-1]
Print(word_break(“applepenapple”, [“apple”, “pen”]))  # Output: True
```

156**.** N-Queens Problem (Backtracking)

Input: N = 4

```
Def solve_n_queens(n):
   Res = []
   Board = [-1] * n
Def is_safe(row, col):
      For I in range(row):
         If board[i] == col or abs(board[i] – col) == row – i:
            Return False
      Return True
Def solve(row):
      If row == n:
         Res.append(board[:])
```

```
        Else:
            For col in range(n):
                If is_safe(row, col):
                    Board[row] = col
                    Solve(row + 1)
 Solve(0)
    Return res
Print(solve_n_queens(4))  # Output: Positions of queens in valid solutions
```

157. Given an array of integers nums and an integer target, return *indices of the two numbers such that they add up to target*.

You may assume that each input would have ***exactly*** **one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

Input: nums = [2,7,11,15], target = 9

Answer :

```
class Solution:
    def twoSum(nums):
          n = len(nums)
          for i in range(n):
              for j in range(i + 1, n):
                    if nums[i] + nums[j] == target:
                              return (i, j)
```

158. Valid Parentheses

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.

2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

1.Input : s = “( )”   2. input: s = “[ )”

Output: true   output: False

Answer :

```
def isValid(s):
    while "()" in s or "[]" in s or "{}" in s:
        s = s.replace("()", "")
        s = s.replace("[]", "")
        s = s.replace("{}", "")
    return s == ""
```

159**.** Longest Substring Without Repeating Characters

Given a string s, find the length of the **longest substring** without duplicate characters.

Input: s = "abcabcbb"

Output: 3

Answer :

```
def lengthOfLongestSubstring(s):
    max_len = 0
    n = len(s)
    for i in range(n):
        seen = set()
        for j in range(i, n):
            if s[j] in seen:
                break
            seen.add(s[j])
            max_len = max(max_len, j - i + 1)
    return max_len
```

160. Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.

Notice that the solution set must not contain duplicate triplets.

Input: nums = [-1,0,1,2,-1,-4]

Output: [[-1,-1,2],[-1,0,1]]

Answer :

```
def threeSum(nums):
    n = len(nums)
    res = set()
    for i in range(n):
        for j in range(i+1, n):
            for k in range(j+1, n):
                if nums[i] + nums[j] + nums[k] == 0:
                    triplet = tuple(sorted([nums[i], nums[j], nums[k]]))
                    res.add(triplet)
    return list(res)
```

161. Given an unsorted integer array nums. Return the *smallest positive integer* that is *not present* in nums.

You must implement an algorithm that runs in O(n) time and uses O(1) auxiliary space.

Input: nums = [1,2,0]

Output: 3

Answer :

```
def firstMissingPositive(nums):
    i = 1
    while True:
        if i not in nums:
```

```
        return i
    i += 1
```

162. Given an input string s and a pattern p, implement regular expression matching with support for '.' and '*' where:

- '.' Matches any single character.
- '*' Matches zero or more of the preceding element.

The matching should cover the **entire** input string (not partial).

Input: s = "aa", p = "a"

Output : false

Input: s = "aa", p = "a*"

Output : true

Answer :

```
def isMatch(s , p):
    if not p:
        return not s

    first_match = bool(s) and (s[0] == p[0] or p[0] == '.')

    if len(p) >= 2 and p[1] == '*':
        return self.isMatch(s, p[2:]) or (first_match and self.isMatch(s[1:], p))
    else:
        return first_match and self.isMatch(s[1:], p[1:])
```

163. Given an array of integers nums and an integer target, return *indices of the two numbers such that they add up to target*.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

Input: nums = [2,7,11,15], target = 9

Answer :

```
class Solution:
    def twoSum(nums):
        n = len(nums)
        for i in range(n):
            for j in range(i + 1, n):
                if nums[i] + nums[j] == target:
                    return (i, j)
```

164. Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

4. Open brackets must be closed by the same type of brackets.
5. Open brackets must be closed in the correct order.
6. Every close bracket has a corresponding open bracket of the same type.

1.Input : s = “( )”   2. input: s = “[ )”

Output: true   output: False

Answer :

```
def isValid(s):
    while "()" in s or "[]" in s or "{}" in s:
        s = s.replace("()", "")
        s = s.replace("[]", "")
        s = s.replace("{}", "")
    return s == ""
```

165. Given a string s, find the length of the **longest substring** without duplicate characters.

Input: s = "abcabcbb"

Output: 3

Answer :

```
def lengthOfLongestSubstring(s):
    max_len = 0
    n = len(s)
    for i in range(n):
        seen = set()
        for j in range(i, n):
            if s[j] in seen:
                break
            seen.add(s[j])
            max_len = max(max_len, j - i + 1)
    return max_len
```

166. Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0.

Notice that the solution set must not contain duplicate triplets.

Input: nums = [-1,0,1,2,-1,-4]

Output: [[-1,-1,2],[-1,0,1]]

Answer :

```
def threeSum(nums):
    n = len(nums)
    res = set()
    for i in range(n):
        for j in range(i+1, n):
            for k in range(j+1, n):
                if nums[i] + nums[j] + nums[k] == 0:
                    triplet = tuple(sorted([nums[i], nums[j], nums[k]]))
                    res.add(triplet)
    return list(res)
```

167. Given an unsorted integer array nums. Return the *smallest positive integer* that is *not present* in nums.

You must implement an algorithm that runs in O(n) time and uses O(1) auxiliary space.

Input: nums = [1,2,0]

Output: 3

Answer :

```
def firstMissingPositive(nums):
    i = 1
    while True:
        if i not in nums:
            return i
        i += 1
```

168. Given an input string s and a pattern p, implement regular expression matching with support for '.' and '*' where:

- '.' Matches any single character.
- '*' Matches zero or more of the preceding element.

The matching should cover the **entire** input string (not partial).

Input: s = "aa", p = "a"

Output : false

Input: s = "aa", p = "a*"

Output : true

Answer :

```
def isMatch(s , p):
        if not p:
            return not s

        first_match = bool(s) and (s[0] == p[0] or p[0] == '.')
        if len(p) >= 2 and p[1] == '*':
            return self.isMatch(s, p[2:]) or (first_match and self.isMatch(s[1:], p))
        else:
            return first_match and self.isMatch(s[1:], p[1:])
```

169. Reverse a string.

```
def reverse_string(s):
    reversed_string = ""
    for i in range(len(s) - 1, -1, -1):
        reversed_string += s[i]
    return reversed_string
# Example usage
input_string = "example"
output_string = reverse_string(input_string)
print("Input:", input_string)
print("Reversed string:", output_string)
```

Output: Input: example

Reversed string: elpmaxe

170. Check if a number is prime

Input: 29

```
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5)+1):
        if n % i == 0:
            return False
    return True
print(is_prime(29))
 Output: True
```

171**.** Find all pairs in a list that sum up to a given number

Input: arr = [1, 5, 7, -1, 5], sum = 6

```
def find_pairs(arr, target):
    seen = {}
    pairs = []
    for num in arr:
        if target - num in seen:
            pairs.append((target - num, num))
        seen[num] = True
    return pairs
print(find_pairs([1, 5, 7, -1, 5], 6))
Output: [(1, 5), (7, -1), (1, 5)]
```

172. Reverse a linked list.

Input: 1 -> 2 -> 3 -> 4 -> 5

```
def __init__(self, val=0, next=None):
                self.val = val
                self.next = next
def reverse_linked_list(head):
            prev = None
            current = head
            while current:
                next_node = current.next
                current.next = prev
                prev = current
                current = next_node
            return prev
```

Output: 5 -> 4 -> 3 -> 2 -> 1

173. Find the shortest path in a graph. (Using Dijkstra's Algorithm)

Input: Graph represented as an adjacency list.

```
import heapq
    def shortest_path_dijkstra(graph, start, end):
        distances = {node: float('inf') for node in graph}
        distances[start] = 0
        priority_queue = [(0, start)]
        while priority_queue:
            (current_distance, current_node) = heapq.heappop(priority_queue)
            if current_distance > distances[current_node]:
                continue
            if current_node == end:
                break
            for neighbor, weight in graph[current_node].items():
                distance = current_distance + weight
                if distance < distances[neighbor]:
                    distances[neighbor] = distance
                    heapq.heappush(priority_queue, (distance, neighbor))
        return distances[end]
```

Output: Shortest path from source to destination.

174. Word Break Problem (using recursion + memoization)

Input: s = "applepenapple", wordDict = ["apple", "pen"]

```
def word_break(s, wordDict):
    memo = {}
    def can_break(start):
        if start == len(s): return True
        if start in memo: return memo[start]
        for word in wordDict:
            if s.startswith(word, start) and can_break(start + len(word)):
                memo[start] = True
                return True
        memo[start] = False
        return False
    return can_break(0)
print(word_break("applepenapple", ["apple", "pen"]))
```

Output: True