E

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
struct NODE{
    int p, v;
    bool operator < (const NODE & a) const {
        return this->v < a.v;
    }
    bool operator > (const NODE & a) const {
        return a < (*this);
    }
} a[N];
int d[N];
int c[N];
int n, cnt;
int lowbit(int x){
    return x & (-x);
}
void add(int x){
    for (int i = x; i <= cnt;i+=lowbit(i))
    {
        c[i]++;
    }
}
int query(int x){
    int sum = 0;
    for (int i = x; i > 0;i-=lowbit(i))
    {
        sum += c[i];
    }
    return sum;
}
int main(){
    int t;
    scanf("%d",&t);
    while (t--){
        memset(c,0,sizeof(c));
        scanf("%d",&n);
        for (int i = 1; i <= n;i++)
        {
            scanf("%d",&a[i].v);
            a[i].p = i;
        }
        sort(a+1,a+1+n);
        cnt = 0;
        for (int i = 1; i <= n;i++)
        {
            if (i==1 || a[i].v!=a[i-1].v){
                cnt++;
            }
            d[a[i].p] = cnt;
        }
        int ans=0;
        int ma = 0;
        for (int i = 1; i <= n;i++)
        {
            add(d[i]);
            if (query(d[i])<i){
                ma = max(ma,query(d[i]-1)-ans);
                ans++;
            }
        }
        printf("%d\n", ans + ma);
    }
    return 0;
}
```

B

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e5 + 5;
struct NODE{
    ll p;
    ll v;
    NODE (){}
    NODE (ll p,ll v):p(p),v(v){}
} b[N];
bool cmp(NODE a,NODE b){
    return (a.v == b.v ? a.p < b.p : a.v < b.v);
}
int n;
ll x, y;
ll getIndex(ll p,ll v){
    int l = 1, r = n, mid;
    ll ans = -1;
    while (l<=r){
        mid = l + r >> 1;
        if (b[mid].v<v){
            l = mid + 1;
        }else if (b[mid].v>v){
            r = mid - 1;
        }else if (b[mid].p<p){
            l = mid + 1;
        }else if (b[mid].p>p){
            r = mid - 1;
        }else{
            ans = mid;
            break;
        }
    }
    return ans;
}
void work(){
    scanf("%d%lld%lld",&n,&x,&y);
    ll x1 = 0, y1 = 0;
    for (int i = 1; i <= n;i++)
    {
        scanf("%lld",&b[i].v);
        b[i].p = i;
        x1 += i * b[i].v;
        y1 += i * b[i].v * b[i].v;
    }

    sort(b+1,b+1+n,cmp);
    ll ans = 0;
    if (x1==x||y1==y){
        if (x1==x && y1==y){
            ll cnt = 0;
            for (int i = 1; i <= n;i++)
            {
                if (i==1 || b[i].v==b[i-1].v)
                    cnt++;
                else {
                    ans += cnt * (cnt - 1) / 2;
                    cnt = 1;
                }
            }
            if (cnt)
                ans += cnt * (cnt - 1) / 2;
        }else
```

```
            ans = 0;
    }else{
        ll tmp = (y1 - y) % (x1 - x);
        if (tmp == 0 && (y1-y)/(x1-x)>0){
            ll C = (y1 - y) / (x1 - x);
            for (int i = 1; i <= n;i++)
            {
                ll v = C - b[i].v;//aj
                if (b[i].v==v)
                    continue;
                ll pos = b[i].p - (x1 - x) / (b[i].v - v);//j
                ll ind = getIndex(pos, v);
                if (ind!=-1 && ind>=i)
                    ans++;
            }
        }
    }
    printf("%lld\n",ans);
}
int main(){
    int t;
    scanf("%d",&t);
    while (t--){
        work();
    }
    return 0;
}
```

I

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

namespace high_precision
{
    const int INITIAL_SIZE = 50000;                  // sizeof(Big
Integer) = sizeof(int) * (UNITIAL_SIZE + 1)
    const int UNIT_LENGTH = 8;                      // Maximum Leng
th of BigInteger = INITIAL_SIZE * UNIT_LENGTH
    const int UNIT_SIZE = 100000000;                // 10 ** UNIT_L
ENGTH
    const int UNIT_MAX = UNIT_SIZE - 1;
    char buffer[INITIAL_SIZE * UNIT_LENGTH + 1];  // inner char a
rray for BigInteger2charArray
```

```
    char raw_sqrt[INITIAL_SIZE * UNIT_LENGTH + 1];
    int SQRT_L, SQRT_CNT;

    class BigInteger
    {
    public:
        int a[INITIAL_SIZE];
        int length;

        BigInteger() : a() { length = 1; memset(a, 0, sizeof(a));
 }
        BigInteger(long long);
        BigInteger(int);
        explicit BigInteger(const char *);
        BigInteger(const BigInteger &T) : a(), length(T.length)
{ memcpy(a, T.a, sizeof a); }

        BigInteger &operator =  (const BigInteger &T) { length =
T.length; memcpy(a, T.a, sizeof a); return *this; }
        BigInteger  operator +  (const BigInteger &) const;
        BigInteger &operator += (const BigInteger &T) { return *t
his = *this + T; }
        BigInteger  operator -  (const BigInteger &) const;
        BigInteger &operator -= (const BigInteger &T) { return *t
his = *this - T; }
        BigInteger  operator *  (const BigInteger &) const;
        BigInteger &operator *= (const BigInteger &T) { return *t
his = *this * T; }
        BigInteger  operator /  (const long long &) const;
        BigInteger &operator /= (const long long &b) { return *th
is = *this / b; }
        long long   operator %  (const long long &) const;
        BigInteger  operator ^  (const long long &) const;
        BigInteger &operator ^= (const long long &b) { return *th
is = *this ^ b; }
        bool        operator >  (const BigInteger &T) const;
        bool        operator <  (const BigInteger &T) const { ret
urn T > *this; }
        bool        operator >= (const BigInteger &T) const { ret
urn !(T > *this); }
        bool        operator <= (const BigInteger &T) const { ret
urn !(*this > T); }
        bool        operator == (const BigInteger &T) const;

        void to_inner_char() const;
        char * get_char() const { return buffer + (buffer[0] == '
0'); }
        char * to_char() const { to_inner_char(); return get_char
(); }
        void get_raw_sqrt() const;
        BigInteger self_sqrt() const { get_raw_sqrt(); return Big
Integer(raw_sqrt); }

        void print(bool stdio=true);
        void println(bool stdio=true) { print(stdio); if (stdio)
puts(""); else cout << endl; }
    };

    BigInteger::BigInteger(const long long b) : a()
    {
        long long c, d = b;
        length = 0;
        memset(a, 0, sizeof a);
        while (d > UNIT_MAX)
        {
            c = d - d / UNIT_SIZE * UNIT_SIZE;
            d = d / UNIT_SIZE;
            a[length++] = (int) c;
        }
        a[length++] = (int) d;
    }

    BigInteger::BigInteger(const int b) : a()
    {
        int c, d = b;
        length = 0;
        memset(a, 0, sizeof a);
        while (d > UNIT_MAX)
        {
            c = d - d / UNIT_SIZE * UNIT_SIZE;
            d = d / UNIT_SIZE;
            a[length++] = c;
        }
        a[length++] = d;
    }

    BigInteger::BigInteger(const char *s) : a()
    {
        int t, k, index, l, i;
        bool flag = false;
        memset(a, 0, sizeof a);
        if (s[0] == '-') flag = true, ++s;
        l = (int) strlen(s);
        length = l / UNIT_LENGTH;
        if (l % UNIT_LENGTH) ++length;
        index = 0;
        for (i = l - 1; i >= 0; i -= UNIT_LENGTH)
        {
            t = 0;
            k = i - UNIT_LENGTH + 1;
            if (k < 0) k = 0;
            for (int j = k; j <= i; ++j) t = t * 10 + s[j] - '0';
            a[index++] = t;
        }
        if (flag) a[index - 1] = -a[index - 1];
    }

    BigInteger BigInteger::operator+(const BigInteger &T) const
    {
        BigInteger t(*this);
        int i, big;
        big = T.length > length ? T.length : length;
        for (i = 0; i < big; ++i)
        {
            t.a[i] += T.a[i];
            if (t.a[i] > UNIT_MAX) ++t.a[i + 1], t.a[i] -= UNIT_M
AX + 1;
        }
        if (t.a[big] != 0) t.length = big + 1;
        else t.length = big;
        return t;
    }

    BigInteger BigInteger::operator-(const BigInteger &T) const
    {
        int i, j, big;
        bool flag;
        BigInteger t1, t2;
        if (*this > T) t1 = *this, t2 = T, flag = false;
        else t1 = T, t2 = *this, flag = true;
        big = t1.length;
        for (i = 0; i < big; ++i)
        {
            if (t1.a[i] < t2.a[i])
            {
                j = i + 1;
                while (t1.a[j] == 0) ++j;
                --t1.a[j--];
                while (j > i) t1.a[j--] += UNIT_MAX;
                t1.a[i] += UNIT_SIZE - t2.a[i];
            }
            else t1.a[i] -= t2.a[i];
        }
        t1.length = big;
        while (t1.a[t1.length - 1] == 0 && t1.length > 1) --t1.le
ngth, --big;
        if (flag) t1.a[big - 1] = -t1.a[big - 1];
        return t1;
    }

    BigInteger BigInteger::operator*(const BigInteger &T) const
    {
        BigInteger ret;
        int i = 0, j = 0, up, temp1;
        long long temp;
        for (i = 0; i < length; ++i)
        {
            up = 0;
            for (j = 0; j < T.length; ++j)
            {
                temp = (long long) a[i] * T.a[j] + ret.a[i + j] +
 up;
                if (temp > UNIT_MAX)
                {
                    temp1 = (int) (temp - temp / UNIT_SIZE * UNIT
_SIZE);
                    up = (int) (temp / UNIT_SIZE);
                    ret.a[i + j] = temp1;
                }
                else up = 0, ret.a[i + j] = (int) temp;
            }
            if (up != 0) ret.a[i + j] = up;
        }
        ret.length = i + j;
        while (ret.a[ret.length - 1] == 0 && ret.length > 1) ret.
length--;
        return ret;
    }

    BigInteger BigInteger::operator/(const long long &b) const
    {
        assert(b != 0);
        BigInteger ret;
        long long i, down = 0;
        for (i = length - 1; i >= 0; i--)
        {
            ret.a[i] = (int) ((down * UNIT_SIZE + a[i]) / b);
            down = a[i] + down * UNIT_SIZE - b * ret.a[i] ;
        }
        ret.length = length;
        while (ret.a[ret.length - 1] == 0 && ret.length > 1) --re
t.length;
        return ret;
    }

    long long BigInteger::operator%(const long long &b) const
    {
        long long d = 0;
        for (int i = length - 1; i >= 0; --i) d = (d * UNIT_SIZE %
 b + a[i]) % b;
        return d;
    }

    BigInteger BigInteger::operator^(const long long &n) const
    {
        assert(n >= 0);
        BigInteger res(1);
        if (n == 0) return res;
        if (n == 1) return *this;
        BigInteger tmp = *this;
        for (long long b = n; b; b >>= 1, tmp = tmp * tmp) if (b
& 1) res *= tmp;
        return res;
    }

    bool BigInteger::operator>(const BigInteger &T) const
    {
        if (a[length - 1] < 0)
        {
            if (T.a[T.length - 1] < 0)
            {
                if (length < T.length) return true;
                if (length == T.length)
                {
                    int ln = length - 1;
                    if (a[ln] > T.a[ln]) return true;
                    if (a[ln] < T.a[ln]) return false;
                    --ln;
                    while (a[ln] == T.a[ln] && ln >= 0) --ln;
                    return ln >= 0 && a[ln] < T.a[ln];
                }
            }
            return false;
        }
        if (T.a[length - 1] < 0 || length > T.length) return true;
        if (length == T.length)
        {
            int ln = length - 1;
            while (a[ln] == T.a[ln] && ln >= 0) --ln;
            return ln >= 0 && a[ln] > T.a[ln];
        }
        return false;
    }

    bool BigInteger::operator==(const BigInteger &T) const
    {
        if (length == T.length)
        {
            int ln = length - 1;
            while (a[ln] == T.a[ln] && ln >= 0) --ln;
            return ln == -1;
        }
        return false;
    }

    void BigInteger::to_inner_char() const
    {
        int ln = length - 1, cnt = 1, tmp, i;
        if (a[ln] < 0) buffer[0] = '-', tmp = -a[ln];
        else buffer[0] = '0', tmp = a[ln];
        for (i = UNIT_SIZE / 10; a[ln] / i == 0; i /= 10);
        for (; i; i /= 10)
            buffer[cnt++] = (char) ('0' + tmp / i), tmp %= i;
        for (--ln; ln >= 0; --ln)
        {
            tmp = a[ln];
            for (i = UNIT_SIZE / 10; i; i /= 10)
                buffer[cnt++] = (char) ('0' + tmp / i), tmp %= i;
        }
        buffer[cnt] = '\0';
    }

    int sqrt_dfs(int o, char *O, int I)
    {
        char c, *D = O;
        if(o > 0)
        {
            for(SQRT_L = 0; D[SQRT_L]; D[SQRT_L++] -= 10)
            {
                D[SQRT_L++] -= 120;
                D[SQRT_L] -= 110;
                while(!sqrt_dfs(0, O, SQRT_L)) D[SQRT_L] += 20;
                raw_sqrt[SQRT_CNT++] = (char) ((D[SQRT_L] + 1032)
 / 20);
            }
            raw_sqrt[SQRT_CNT] = '\0';
        }
        else
        {
            c = (char) (o + (D[I] + 82) % 10 - (I > SQRT_L >> 1)
* (D[I - SQRT_L + I] + 72) / 10 - 9);
            D[I] += I < 0 ? 0 : !(o = sqrt_dfs(c / 10, O, I - 1))
 * ((c + 999) % 10 - (D[I] + 92) % 10);
        }
        return o;
    }

    void raw2raw_sqrt()
    {
        SQRT_CNT = 0;
        sqrt_dfs(2, buffer + (strlen(buffer) & 1), 0);
    }

    void BigInteger::get_raw_sqrt() const
    {
        assert(a[length - 1] >= 0);
        to_inner_char();
        raw2raw_sqrt();
    }

    void BigInteger::print(bool stdio)
    {
        int i;
        if (stdio)
        {
            int j;
            int seg;
            char ch[UNIT_LENGTH + 1];
            ch[UNIT_LENGTH] = '\0';
            printf("%d", a[length - 1]);
            for (i = length - 2; i >= 0; --i)
            {
                seg = a[i];
                for (j = 0; j < UNIT_LENGTH; ++j) ch[j] = '0';
                while (seg)
                {
                    ch[--j] = (char) ('0' + seg % 10);
                    seg /= 10;
                }
                printf("%s", ch);
            }
        }
        else
        {
            cout << a[length - 1];
            for (i = length - 2; i >= 0; --i)
            {
                cout.width(UNIT_LENGTH);
                cout.fill('0');
                cout << a[i];
            }
        }
    }
}
using high_precision::BigInteger;
char s1[100005],s2[100005];
int main(){
    int t;
    scanf("%d",&t);
    while (t--){
        scanf(" %s %s",s1,s2);
        BigInteger a = BigInteger(s1);
        BigInteger b = BigInteger(s2);
        BigInteger tmp2 = BigInteger(2);
        BigInteger tmp1 = BigInteger(1);
        a = a + tmp1;
        b = b + tmp2;
        int a1 = a % 3;
        int b1 = b % 3;
        if (a1==0 && b1==0 || a1 && b1)
            printf("0\n");
        else
            printf("1\n");
    }
}
```

F

```
#include <bits/stdc++.h>
using namespace std;
char s[110];
int T;
int main(){
    scanf("%d",&T);
    while(T--){
        scanf("%s",s+0);
        int len= strlen(s);
        printf("%c",s[0]);
        for(int i=1;i<len;i++){
            if(s[i]!='a'&&s[i]!='e'&&s[i]!='i'&&s[i]!='o'&&s[i]!=
'u'&&s[i]!='y')
            printf("%c",s[i]);
        }


        printf("\n");
    }

    return 0;
}
```

G

```
#include<bits/stdc++.h>
using namespace std;
int n;
int main(){
    int t;
    scanf("%d",&t);
    while (t--){
        scanf("%d",&n);
        for (int i = n;;i++)
        {
            if (i%7==0 && i%4){
                printf("%d\n",i);
                break;
            }
        }
    }
    return 0;
}
```

H

```
#include<bits/stdc++.h>
using namespace std;
const int N = 1e5 + 5;
int a[N];
bool able[N];
int n;
int main(){
    int t;
    scanf("%d",&t);
    while (t--){
        scanf("%d",&n);
        for (int i = 1; i <= n;i++)
        {
            scanf("%d",&a[i]);
        }
        int cnt = 0;
        int minus = 0;
        for (int i = 2; i < n;i++)
        {
            if (a[i]>a[i-1] && a[i]>a[i+1]){
                able[i] = 1;
                cnt++;
                if (a[i-1]==a[i+1])
                    minus = 1;
                else if (a[i-1]>a[i+1]){
                    if (i-2<=0 || a[i-1]<=a[i-2])
                        minus = 1;
                }else {
                    if (i+1>n || a[i+1]<=a[i+2])
                        minus = 1;
                }
            }

        }
        for (int i = 2; i < n;i++)
        {
            if (able[i-1] && able[i+1] && a[i-1]==a[i+1]){
                minus = 2;
            }
        }
        printf("%d\n",max(cnt-minus,0));
    }
    return 0;
}
```

J

```
#include<cstdio>
#include<queue>

using namespace std;
const int maxn=1000005;
struct node{
    int to;
    int next;
} a[maxn<<2];
int head[maxn];
int vis[maxn];

int tot;
int n,m;
int pre[maxn];

int ans[maxn];
int ansn;
int anss;




int find(int p) {
    while (pre[p]!=p) {
        pre[p]=pre[pre[p]];
        p=pre[p];
    }
    return p;
}

inline void addedge(int x,int y) {
    tot++;
    a[tot].next=head[x];
    a[tot].to=y;
    head[x]=tot;
    tot++;
    a[tot].next=head[y];
    a[tot].to=x;
    head[y]=tot;

    int xx=find(x);
    int yy=find(y);
    if (xx<yy) {
        pre[yy]=pre[xx];
    } else {
        pre[xx]=pre[yy];
    }
}

priority_queue<int, vector<int>, greater<int> > q;


inline void bfs() {
    while (!q.empty()) {
        int u=q.top();

//      printf("fuck: %d\n", u);

        q.pop();
        ansn++;
        ans[ansn]=u;
        for (int p=head[u];p!=0;p=a[p].next) {
            int v=a[p].to;
            if (vis[v]) {
                continue;
            }
            vis[v]=true;
            q.push(v);
        }
    }
}

inline void init() {
    for (int i=1;i<=n;i++) {
        vis[i]=false;
        head[i]=0;
        pre[i]=i;
    } tot=0;
    while (!q.empty()){
        q.pop();
    }
    anss=0;
    ansn=0;
}

int main() {
    int  T;
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n,&m);
        init();
        for (int i=1;i<=m;i++) {
            int x,y;
            scanf("%d%d", &x,&y);
            addedge(x,y);
        }

//      for (int i=1;i<=n;i++) {
//          printf("%d ", pre[i]);
//      }printf("\n");

        for (int i=1;i<=n;i++) {
            if (pre[i]!=i) {
                continue;
            }
            if (!vis[pre[i]]) {
                vis[pre[i]]=true;
                q.push(pre[i]);
                anss++;
            }
        }
        bfs();
        printf("%d\n", anss);
        for (int i=1;i<=ansn;i++) {
            printf("%d%s",ans[i], i==ansn? "\n":" ");
        }
    }

    return 0;
}

/*
```

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

*/
```

K

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 2e6 + 5;
char s[N], t[N];
ll countSubstrings(string s) {
    //预处理
    string t = "#";
    for (int i = 0; i < s.size(); ++i) {
        t += s[i];
        t += "#";
    }

    vector<ll> RL(t.size(), 0);
    ll MaxRight = 0, pos = 0;
    ll res = 0;
    for (int i = 0; i < t.size(); ++i) {
        RL[i] = MaxRight > i ? min(RL[2 * pos - i], MaxRight - i)
 : 1;

        while (i-RL[i] >=0 && i+RL[i] < t.size() && t[i + RL[i]]
== t[i - RL[i]])//扩展，注意边界
            ++RL[i];
        //更新最右端及其中心
        if (MaxRight < i + RL[i] -1) {
            MaxRight = i + RL[i] -1;
            pos = i;
        }

        res += RL[i]/2;
    }
    return res;
}
```

```
void work(){
    scanf(" %s",s);
    scanf(" %s",t);
    int left = -1, right = -1;
    for (int i = 0; i < strlen(s);i++)
    {
        if (s[i]!=t[i]){
            if (left==-1)
                left = i;
            right = i;
        }
    }

    ll ans = 0;
    if (left==-1 && right==-1){
        ans = countSubstrings(s);
    }else{
        bool flag = 0;
        for (int i = left, j = right; i <= right;i++,j--)
        {
            if (s[i]!=t[j]) {
                flag = 1;
                break;
            }
        }
        if (flag){
            printf("0\n");
            return;
        }
        ans = 1;
        for (int i = left-1, j = right+1; i >= 0 && j < strlen(s);
i--,j++)
        {
            if (s[i]!=s[j]){
                break;
            }
            ans++;
        }
    }
    printf("%lld\n",ans);
}
int main(){
    int t;
    scanf("%d",&t);
    while (t--){
        work();
    }
    return 0;
}
```