



vovuh's blog

Codeforces Round #636 (Div. 3) Editorial

By [vovuh](#), [history](#), 13 hours ago,  

1343A - Candies

Idea: [vovuh](#)[Tutorial](#)

1343A - Candies

Notice that $\sum_{i=0}^{k-1} 2^i = 2^k - 1$. Thus we can replace the initial equation with the following:

$(2^k - 1)x = n$. So we can iterate over all possible k in range $[2; 29]$ (because $2^{30} - 1 > 10^9$) and check if n is divisible by $2^k - 1$. If it is then we can print $x = \frac{n}{2^k - 1}$.

P.S. I know that so many participants found the formula $\sum_{i=0}^{k-1} 2^i = 2^k - 1$ using geometric

progression sum but there is the other way to understand this and it is a way more intuitive for me. Just take a look at the binary representation of numbers: we can notice that $2^0 = 1, 2^1 = 10, 2^2 = 100$ and so on. Thus $2^0 = 1, 2^0 + 2^1 = 11, 2^0 + 2^1 + 2^2 = 111$ and so on. And if we add one to this number consisting of k ones then we get 2^k .

[Solution](#)

```
#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;
        cin >> n;
        for (int pw = 2; pw < 30; ++pw) {
            int val = (1 << pw) - 1;
            if (n % val == 0) {
                cerr << val << endl;
                cout << n / val << endl;
                break;
            }
        }

        return 0;
    }
}
```

→ Pay attention

Before contest

[Codeforces Round #637 \(Div. 1\) - Thanks, Ivan Belonogov!](#)

24:54:18

Before contest

[Codeforces Round #637 \(Div. 2\) - Thanks, Ivan Belonogov!](#)

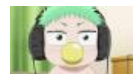
24:54:17

[Register now »](#)

*has extra registration

 Like 136 people like this. [Sign Up](#) to see what your friends like.

→ hawkvine

 Rating: **1444**
 Contribution: 0


hawkvine

- [Settings](#)
- [Blog](#)
- [Teams](#)
- [Submissions](#)
- [Talks](#)
- [Contests](#)

→ Top rated

#	User	Rating
1	MiFaFaOvO	3681
2	tourist	3541
3	Um_nik	3434
4	apiadu	3397
5	maroonrk	3353
6	300iq	3317
7	ecnerwala	3260
8	LHiC	3229
9	TLE	3223
10	Benq	3220

[Countries](#) | [Cities](#) | [Organizations](#)
[View all →](#)

→ Top contributors

#	User	Contrib.
1	antontrygubO_o	193
2	Errichto	187
3	vovuh	174
4	pikmike	170
5	tourist	166
6	ko_osaga	164
6	McDic	164
6	Um_nik	164
9	Radewoosh	163
10	300iq	155

[View all →](#)

→ Find user

1343B - Balanced Array

Idea: **vovuh**[Tutorial](#)

1343B - Balanced Array

Firstly, if n is not divisible by 4 then the answer is "NO" because the parities of halves won't match. Otherwise, the answer is always "YES". Let's construct it as follows: firstly, let's create the array $[2, 4, 6, \dots, n, 1, 3, 5, \dots, n-1]$. This array is almost good except one thing: the sum in the right half is exactly $\frac{n}{2}$ less than the sum in the left half. So we can fix it easily: just add $\frac{n}{2}$ to the last element.

[Solution](#)

```
#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;
        cin >> n;
        n /= 2;
        if (n & 1) {
            cout << "NO" << endl;
            continue;
        }
        cout << "YES" << endl;
        for (int i = 1; i <= n; ++i) {
            cout << i * 2 << " ";
        }
        for (int i = 1; i < n; ++i) {
            cout << i * 2 - 1 << " ";
        }
        cout << 3 * n - 1 << endl;
    }

    return 0;
}
```

1343C - Alternating Subsequence

Idea: **vovuh** and **MikeMirzayanov**[Tutorial](#)

1343C - Alternating Subsequence

Firstly, let's extract maximum by inclusion segments of the array that consists of the numbers with the same sign. For example, if the array is $[1, 1, 2, -1, -5, 2, 1, -3]$ then these segments are $[1, 1, 2]$, $[-1, -5]$, $[2, 1]$ and $[-3]$. We can do it with any "two pointers"-like algorithm. The number of these segments is the maximum possible length of the alternating subsequence because we can take only one element from each block. And as we want to maximize the sum, we need to take the maximum element from each block.

Handle: [Recent actions](#)

- pikmike** → [Educational Codeforces Round 82 Editorial](#)
- vovuh** → [Codeforces Round #636 \(Div. 3\) Editorial](#)
- I love myself** → [Codeforces Round #637 — Thanks, Ivan Belongov!](#)
- JoeSherif** → [Runtime error on test 1!!!!](#)
- maroonrk** → [XX Open Cup GP of Tokyo](#)
- scaler_april_batch → [Scaler Academy Review with Proof](#)
- decoder_1671** → [plz help in finding error in 1343C problem](#)
- vovuh** → [Codeforces Round #634 \(Div. 3\)](#)
- msd07 → [Youtube Channels of Competitive Programmers](#)
- insert_cool_handle** → [\[Feature Request\] Option to never see a blog again](#)
- Jatana** → [Sublime Text \(FastOlympicCoding\) — tools for competitive programming](#)
- MikeMirzayanov** → [Rule about third-party code is changing](#)
- PetarV** → [\(Graph\) Neural Networks for Algorithmic Reasoning](#)
- spakk9 → [Is there any Platform to run past ICPC .IOL problems???](#)
- SPyofgame** → [Problem with std::map](#)
- SHOYEB97 → [Change account setting](#)
- NiceBaseballCoder** → [Our New Game: yuvarLUCK](#)
- vovuh** → [Codeforces Round #636 \(Div. 3\)](#)
- hackerbaba** → [Knapsack1 and knapsack2 atcoder solutions](#)
- Silver** → [YouTube Channels for Competitive Programmers](#)
- deathnaught → [Shorter Questions for shorter formats](#)
- sugim48** → [\[AtCoder\] Educational DP Contest](#)
- preda2or** → [How to prevent Infinite loop in Sublime Text \(Solution\).](#)
- vjudge6699 → [Feature request!!](#)
- elements** → [How is contributions calculated?](#)

[Detailed →](#)

Time complexity: $O(n)$.Solution

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

using namespace std;

int main() {
#ifdef _DEBUG
    freopen("input.txt", "r", stdin);
    //    freopen("output.txt", "w", stdout);
#endif

    auto sgn = [&](int x) {
        if (x > 0) return 1;
        else return -1;
    };

    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        vector<int> a(n);
        for (auto &it : a) cin >> it;
        long long sum = 0;
        for (int i = 0; i < n; ++i) {
            int cur = a[i];
            int j = i;
            while (j < n && sgn(a[i]) == sgn(a[j])) {
                cur = max(cur, a[j]);
                ++j;
            }
            sum += cur;
            i = j - 1;
        }
        cout << sum << endl;
    }

    return 0;
}
```

1343D - Constant Palindrome Sum

Idea: **MikeMirzayanov**Tutorial

1343D - Constant Palindrome Sum

It is obvious that if we fix the value of x then there are three cases for the pair of elements:

1. We don't need to change anything in this pair;
2. we can replace one element to fix this pair;
3. we need to replace both elements to fix this pair.

The first part can be calculated easily in $O(n + k)$, we just need to create the array of frequencies cnt , where cnt_x is the number of such pairs (a_i, a_{n-i+1}) that $a_i + a_{n-i+1} = x$.

The second part is a bit tricky but still doable in $O(n + k)$. For each pair, let's understand the minimum and the maximum sum we can obtain using at most one replacement. For the i -th pair, all such sums belong to the segment $[\min(a_i, a_{n-i+1}) + 1; \max(a_i, a_{n-i+1}) + k]$. Let's make $+1$ on this segment using prefix sums (make $+1$ in the left border, -1 in the right border plus one and then just

compute prefix sums on this array). Let this array be $pref$. Then the value $pref_x$ tells the number of such pairs that we need to replace **at most one** element in this pair to make it sum equals x .

And the last part can be calculated as $\frac{n}{2} - pref_x$. So, for the sum x the answer is $(pref_x - cnt_x) + (\frac{n}{2} - pref_x) \cdot 2$. We just need to take the minimum such value among all possible sums from 2 to $2k$.

There is another one solution that uses scanline, not depends on k and works in $O(n \log n)$ but it has no cool ideas to explain it here (anyway the main idea is almost the same as in the solution above).

Solution

```
#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, k;
        cin >> n >> k;
        vector<int> a(n);
        for (auto &it : a) cin >> it;
        vector<int> cnt(2 * k + 1);
        for (int i = 0; i < n / 2; ++i) {
            ++cnt[a[i] + a[n - i - 1]];
        }
        vector<int> pref(2 * k + 2);
        for (int i = 0; i < n / 2; ++i) {
            int l1 = 1 + a[i], r1 = k + a[i];
            int l2 = 1 + a[n - i - 1], r2 = k + a[n - i - 1];

            assert(max(l1, l2) <= min(r1, r2));
            ++pref[min(l1, l2)];
            --pref[max(r1, r2) + 1];
        }
        for (int i = 1; i <= 2 * k + 1; ++i) {
            pref[i] += pref[i - 1];
        }
        int ans = 1e9;
        for (int sum = 2; sum <= 2 * k; ++sum) {
            ans = min(ans, (pref[sum] - cnt[sum]) + (n / 2 - pref[sum]) * 2);
        }
        cout << ans << endl;
    }

    return 0;
}
```

1343E - Weights Distributing

Idea: **MikeMirzayanov**

Tutorial

1343E - Weights Distributing



If we distribute costs optimally, then this pair of paths ($a \rightarrow b$ and $b \rightarrow c$) can look like just a straight path that doesn't visit the same vertex twice or like three straight paths with one intersection point x . The first case is basically a subcase of the second one (with the intersection point a, b or c). So, if we fix the intersection point x then these two paths ($a \rightarrow b$ and $b \rightarrow c$) become four paths ($a \rightarrow x, x \rightarrow b, b \rightarrow x$ and $x \rightarrow c$). We can notice that each path we denoted should be the shortest possible because if it isn't the shortest one then we used some prices that we couldn't use.

Let the length of the shortest path from u to v be $dist(u, v)$. Then it is obvious that for the fixed intersection point x we don't need to use more than $dist(a, x) + dist(b, x) + dist(c, x)$ smallest costs. Now we want to distribute these costs between these three paths somehow. We can see that the path from b to x is used twice so it is more optimally to distribute the smallest costs along this part. So, let $pref_i$ be the sum of the first i smallest costs (just prefix sums on the sorted array p). Then for the intersection point x the answer is $pref_{dist(b,x)} + pref_{dist(a,x)+dist(b,x)+dist(c,x)}$ (if $dist(a, x) + dist(b, x) + dist(c, x) \leq m$). We can calculate distances from a, b and c to each vertex with three runs of bfs.

Time complexity: $O(m \log m)$.

Solution

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

using namespace std;

const int INF = 1e9;

vector<vector<int>> g;

void bfs(int s, vector<int> &d) {
    d[s] = 0;
    queue<int> q;
    q.push(s);
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        for (auto to : g[v]) {
            if (d[to] == INF) {
                d[to] = d[v] + 1;
                q.push(to);
            }
        }
    }
}

int main() {
#ifdef _DEBUG
    freopen("input.txt", "r", stdin);
    // freopen("output.txt", "w", stdout);
#endif

    int t;
    cin >> t;
    while (t--) {
        int n, m, a, b, c;
        cin >> n >> m >> a >> b >> c;
        --a, --b, --c;

        vector<int> p(m);
        for (auto &it : p) cin >> it;
        sort(p.begin(), p.end());
        vector<long long> pref(m + 1);
        for (int i = 0; i < m; ++i) {
            pref[i + 1] = pref[i] + p[i];
        }
    }
}
```

```

g = vector<vector<int>>(n);
for (int i = 0; i < m; ++i) {
    int x, y;
    cin >> x >> y;
    --x, --y;
    g[x].push_back(y);
    g[y].push_back(x);
}

vector<int> da(n, INF), db(n, INF), dc(n, INF);
bfs(a, da);
bfs(b, db);
bfs(c, dc);

long long ans = 1e18;
for (int i = 0; i < n; ++i) {
    if (da[i] + db[i] + dc[i] > m) continue;
    ans = min(ans, pref[db[i]] + pref[da[i] + db[i]
+ dc[i]]);
}
cout << ans << endl;

}

return 0;
}

```

1343F - Restore the Permutation by Sorted Segments

Idea: **MikeMirzayanov**

Tutorial

1343F - Restore the Permutation by Sorted Segments

Let's fix the first element and then try to restore permutation using this information. One interesting fact: if such permutation exists then it can be restored uniquely. Let's remove the first element from all segments containing it (we can use some logarithmic data structure for it). Then we just have a smaller problem but with one important condition: there is a segment consisting of one element (again, if such permutation exists). So, if the number of segments of length 1 is zero or more than one by some reason then there is no answer for this first element. Otherwise, let's place this segment (a single element) in second place, remove it from all segments containing it and just solve a smaller problem again.

If we succeed with restoring the permutation then we need to check if this permutation really satisfies the given input segments (see the first test case of the example to understand why this case appears). Let's just iterate over all i from 2 to n and then over all j from $i - 1$ to 1. If the segment a_j, a_{j+1}, \dots, a_i is in the list, remove it and go to the next i . If we can't find the segment for some i then this permutation is wrong.

Time complexity: $O(n^3 \log n)$ (or less, maybe?)

Solution

```

#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;
    cin >> n;
    vector<set<int>> segs;
    for (int i = 0; i < n - 1; ++i) {
        set<int> cur;
        int k;
        cin >> k;
        for (int j = 0; j < k; ++j) {
            int x;
            cin >> x;
            cur.insert(x);
        }
        segs.push_back(cur);
    }
    for (int fst = 1; fst <= n; ++fst) {
        vector<int> ans;
        bool ok = true;
        vector<set<int>> cur = segs;
        for (auto &it : cur) if (it.count(fst))
            it.erase(fst);

        ans.push_back(fst);
        for (int i = 1; i < n; ++i) {
            int cnt1 = 0;
            int nxt = -1;
            for (auto &it : cur) if (it.size() ==
1) {
                ++cnt1;
                nxt = *it.begin();
            }
            if (cnt1 != 1) {
                ok = false;
                break;
            }
            for (auto &it : cur) if (it.count(nxt))
                it.erase(nxt);

            ans.push_back(nxt);
        }
        if (ok) {
            set<set<int>> all(segs.begin(),
segs.end());

            for (int i = 1; i < n; ++i) {
                set<int> seg;
                seg.insert(ans[i]);
                bool found = false;
                for (int j = i - 1; j >= 0; --
j) {
                    seg.insert(ans[j]);
                    if (all.count(seg)) {
                        found = true;
                        all.erase(seg);
                        break;
                    }
                }
                if (!found) ok = false;
            }
        }
        if (ok) {
            for (auto it : ans) cout << it << " ";
            cout << endl;
            break;
        }
    }
}

```



```
return 0;
```

```
}
```

Tutorial of Codeforces Round #636 (Div. 3)

codeforces, 636, third division, editorial

+125



[vovuh](#)

13 hours ago

[177](#)



Comments (177)

[Write comment?](#)



[atishay127](#)

13 hours ago, <#> |

+12

How Problem D can be solved using scanline algo?

→ [Reply](#)



[vovuh](#)

13 hours ago, <#> |

+53

Well, I can explain it rough. The thing is that there are $O(n)$ (actually at most $2n$) endpoints of these segments (the second case) and $O(n)$ (at most n) sums of kind $a_i + a_{n-i+1}$ (the first case). And the idea is that you are only interested in these $3n$ points (maybe with some adjacent points also). This is because in other points values of *pref* and *cnt* don't change at all. So you can compress coordinates and then just consider all segments as events going from left to right and changing the value of *pref* correspondingly and store *cnt* in some logarithmic data structure like map in C++.

→ [Reply](#)



[grandesonnerie](#)

5 hours ago, <#> |

+5

I have implemented an $O(n \log n)$ solution using compression [here](#). For those of you who are confused, check it out!

→ [Reply](#)



[striver_79](#)

9 hours ago, <#> |

+22

video editorial of D. Click [here](#)

→ [Reply](#)



[Saiful6854](#)

7 hours ago, <#> |

0

Thanks for the video link. It's really helpful.

→ [Reply](#)



[decoder__](#)

3 hours ago, <#> |

+1

Can you explain what exactly is a scanline algorithm?

→ [Reply](#)



[Shikhar1](#)

95 minutes ago, <#> |

0

great one!

→ [Reply](#)



[starboy_99](#)

83 minutes ago, <#> |

0

Thank u so much... Really helpful!!

→ [Reply](#)



[spookywooky](#)

8 hours ago, <#> |

+3

Because I think compared to the tutorials solution it is much simpler code here a link to a submission [77584565](#)

→ [Reply](#)