CONTEST #2:
POINTERS & BINARY SEARCH
SOLUTIONS

Link contest: gdoj.eu.org/contest/gg9_003

I. Hướng dẫn đọc lời giải:

1. Đọc kĩ và chú ý những chỗ có **định dạng đặc biệt** (in đậm, in nghiêng, gạch chân, tô màu,...).
2. Chép lại code rồi **debug** kết hợp **đọc lời giải** cho đến khi ***hiểu code***.
3. Khi hiểu code rồi thì cố gắng ***code lại mà không nhìn*** (không học thuộc lòng mà phải hiểu).
4. Đối với những **bài khó**, bạn *không cần phải cố gắng quá sức* để hiểu. Nếu cảm thấy **hiểu gần hết** thì hỏi những phần còn khúc mắc trong chatbox; nếu cảm thấy **không hiểu gì** thì luyện cho trình lên rồi quay lại đọc sau. Nên nhớ các **bài khó** được sinh ra để làm các bạn *"trầy da tróc vẩy"*, vì vậy đừng nản lòng khi **chưa** AC nhé!

II. Lời giải:

1. Tính tổng - SUM. *

```cpp
#include <bits/stdc++.h>
using namespace std;
int a[1005];
int n, k, ans = 0;
int main() {
    freopen ("SUM.INP", "r", stdin);
    freopen ("SUM.OUT", "w", stdout);
    cin >> n >> k;
    for (int i=1; i<=n; i++) cin >> a[i];

    for (int i=1; i<=n; i++) {
        for (int j=i+1; j<=n; j++)
            if (a[i] + a[j] == k) ans++;
    }
    cout << ans;
    return 0;
}
```

- Ý tưởng: đề bài đặt giới hạn $n \leqslant 10^3$ nên ta chỉ cần dùng giải thuật duyệt trâu vẫn có thể giải quyết được bài toán.

- Thực hiện: dùng 2 vòng lặp lồng nhau duyệt qua và đếm tất cả cặp phần tử *(a[i], a[j])* với $1 \leqslant i < j \leqslant n$ và thoả mãn **a[i] + a[j] = x**.

→ **Độ phức tạp: $O(n^2)$.**

2. Vườn rau - VUONRAU. *

- Ý tưởng: ta dễ dàng biết được một số thông tin sau:

- Giây đầu tiên, các vòi nước sẽ tưới rau trong ô của chính nó.
- Từ giây thứ hai, tất cả các vòi nước sẽ tưới ô liền trước và liền sau.
- Như vậy, để tưới hết luống rau, ta cần canh thời gian sao cho ô cách xa các vòi nước nhất trong vườn vẫn phải được tưới.

- Thực hiện:

- Gọi ***dist*** là khoảng cách xa nhất giữa hai vòi nước kế nhau. Ví dụ: có 3 vòi nước được đặt ở các vị trí 2, 7, 8 thì ***dist*** là 5 (vì 7 - 2 = 5).
- Vì sau từng giây, mỗi vòi đều lan ra xung quanh nó một ô, nên sau ***dist/2*** giây thì đảm bảo tất cả rau nằm trong khoảng từ vòi đầu tiên đến vòi cuối cùng đều được tưới.
- Gọi ***first_last*** là max giữa khoảng cách từ điểm đầu luống rau đến vòi nước đầu tiên và khoảng cách từ vòi nước cuối cùng đến điểm kết thúc của luống rau. Tức là ***max (0 → v[1], v[k] → n)***.
- Kết quả là **max (dist, first_last)**.

→ Độ phức tạp: O(k).

- Code AC:

```
#include <bits/stdc++.h>
using namespace std;
int n, k, ans = 0, dist = 0;
int v[100001];
int main() {
    freopen ("VUONRAU.INP", "r", stdin);
    freopen ("VUONRAU.OUT", "w", stdout);
    cin >> n >> k;
    for (int i=1; i<=k; i++) cin >> v[i];

    int first_last = max(v[1], n-v[k]+1);
    for (int i=1; i<=k; i++)
        dist = max(dist, ((v[i]-v[i-1])/2) + 1);
        // +1 vì giây đầu tiên phải tưới chính nó
    int result = max(dist, first_last);
    cout << result;
    return 0;
}
```

3. Dãy không tăng - DKT. *

- Ý tưởng: Đề đã cho dãy ***a*** và ***b*** sắp xếp không tăng, dãy ***c*** không giảm, nên ta chỉ cần 3 con trỏ, trong đó 2 con trỏ của dãy ***a*** và ***b*** chạy từ **trái sang phải**, con trỏ của dãy **c** chạy từ **phải sang trái** rồi so sánh để lấy ra phần tử lớn nhất và di chuyển con trỏ lên sang trái (phải).

- Thực hiện:

- Vì dãy ***a*** được sắp xếp không giảm nên ta đặt con trỏ ***x*** của dãy ***a*** ở vị trí **1**.
- Tương tự dãy ***a***, con trỏ ***y*** của dãy ***b*** được đặt ở vị trí **1**.
- Dãy ***c*** được sắp xếp không tăng nên ta đặt con trỏ ***z*** của dãy ***c*** ở vị trí **m**.

- Để tạo dãy ***d*** không tăng, ở mỗi lần duyệt, ta tìm **con trỏ đang trỏ vào phần tử lớn nhất** để đưa phần tử đó vào mảng ***d*** và **di chuyển con trỏ**.
- Dùng biến ***k*** để theo dõi chỉ số của dãy ***d***, khi k đạt *2n + m* thì dừng vòng lặp.

→ Độ phức tạp: O(n + m).

- Code AC:

```
#include <bits/stdc++.h>
using namespace std;
int a[10005], b[10005], c[10005], d[50005];

int main() {
    freopen ("DKT.INP", "r", stdin);
    freopen ("DKT.OUT", "w", stdout);
    int m, n; cin >> n >> m;
    for (int i=1; i<=n; i++) cin >> a[i];
    for (int i=1; i<=n; i++) cin >> b[i];
    for (int i=1; i<=m; i++) cin >> c[i];

    int x = 1, y = 1, z = m, k = 1;
    while (k <= n+n+m) {
        int Max;
        if (x<=n && a[x]>=b[y] && a[x]>=c[z])
            { Max = a[x]; x++; }
        else if (y<=n && b[y]>=a[x] && b[y]>=c[z])
            { Max = b[y]; y++; }
        else if (z>=1 && c[z]>=a[x] && c[z]>=b[y])
            { Max = c[z]; z--; }
        d[k] = Max; k++;
    }
    for (int i=1; i<k; i++) cout << d[i] << " ";
    return 0;
}
```

4. Dãy đẹp - NICEROW. *

- Ý tưởng: để tìm ra số ***k*** thỏa mãn điều kiện đề bài, ta giải quyết bài toán bằng *Binary Search* như sau:

- Trước tiên, để tồn tại ***k*** thỏa mãn yêu cầu thì ***n*** *không được nhỏ hơn* ***3***.
- Sắp xếp lại dãy để thực hiện tìm kiếm nhị phân.
- Mỗi khi thu được ***k***, ta kiểm tra số ***k*** đó có thỏa mãn điều kiện để ***dãy số a*** là dãy đẹp hay không.

- Thực hiện:

- Ở mỗi lần chặt nhị phân, ta giả định ***phần tử giữa mảng*** là ***k*** *(k = a[mid])* và kiểm tra ***k*** có thỏa mãn đề bài không.
- Cách kiểm tra:
  + Gọi ***cnt_les*** và ***cnt_grt*** lần lượt là số lượng phần tử nhỏ hơn k và số lượng phần tử lớn hơn k.
  + Duyệt qua toàn bộ dãy ***a*** để đếm ***cnt_les*** và ***cnt_grt***.

- Nếu ***cnt_les = cnt_grt*** tức là số lượng phần tử nhỏ hơn k bằng số lượng phần tử lớn hơn k. Ta in ra "1" và kết thúc chương trình.
- Nếu ***cnt_les ≠ cnt_grt***:
  + **TH1**: ***cnt_les < cnt_grt*** tức là ***số k hiện tại*** đang bị **lệch về bên trái** (vì dãy đã được sắp xếp). Do đó, ta thu hẹp phạm vi tìm kiếm **về phần bên phải** dãy.
  + **TH2**: ***cnt_les > cnt_grt*** tức là ***số k hiện tại*** đang bị **lệch về bên phải** (vì dãy đã được sắp xếp). Do đó, ta thu hẹp phạm vi tìm kiếm **về phần bên trái** dãy.
- Nếu duyệt qua hết dãy mà chương trình không in ra "1" tức là không tìm được số k thích hợp => in ra "0".

→ Độ phức tạp: O(n * log(n)).

- Code AC:

```
#include <bits/stdc++.h>
using namespace std;
int n, a[1000005];

int main() {
    freopen ("NICEROW.INP", "r", stdin);
    freopen ("NICEROW.OUT", "w", stdout);
    cin >> n;
    if (n<3) { cout<<"0"; return 0; }
    for (int i=1; i<=n; i++) cin >> a[i];

    int L = 0, R = n-1, k = -1;
    sort (a+1, a+n+1);
    while (L <= R) {
        int mid = L + (R-1) / 2;
        k = a[mid];
        int cnt_les = 0, cnt_grt = 0;
        for (int i=1; i<=n; i++) {
            if (a[i] < k) cnt_les++;
            else if (a[i] > k) cnt_grt++;
        }
        if (cnt_les == cnt_grt) {
            cout << "1";
            return 0;
        }
        else if (cnt_les < cnt_grt) L = mid+1;
        else R = mid-1;
    }
    cout << "0"; return 0;
}
```

5. Dãy số lòng chảo - DAYSOLC. *

- Ý tưởng: duyệt qua từng phần tử trong mảng và kiểm tra phần tử đó có thể là đáy của một chảo hay không.

- **TH1** *(phần tử đó có thể làm đáy)*: ta duyệt ***sang 2 bên*** đến khi ***chạm miệng chảo***. Từ đó tính được độ dài dãy số lòng chảo này.
- **TH2** *(phần tử đó không thể làm đáy)*: ta tiếp tục vòng lặp.

- Thực hiện:

- Điều kiện xác định dãy số lòng chảo: có từ ***3 phần tử trở lên*** và phần tử đáy thỏa mãn ***a[i-1] < a[i] < a[i+1]***.
- Lưu ý: tính từ đáy, dãy lòng chảo phải ***tăng dần sang hai bên***, do đó khi duyệt, nếu gặp ***a[i-1] >= a[i]*** hoặc ***a[i] ⩾ a[i+1]*** tức là dãy đang duyệt không có tính chất lòng chảo, ta đến lần lặp tiếp theo để tìm các đáy khác.
- Từ phần tử đáy, ta mở rộng thành chảo dần sang hai bên:
  + Trái: ***a[i-1] < a[i]*** → tiếp tục mở rộng.
  + Phải: ***a[i] < a[i+1]*** → tiếp tục mở rộng.
- Khi không còn mở rộng được nữa, ***L*** và ***R*** chính là hai bên miệng chảo. Khi đó, độ dài dãy lòng chảo này là ***R - L + 1***.
- So sánh độ dài dãy vừa tìm được với dãy đang lưu và cập nhật kết quả.

→ Độ phức tạp: O(n²).

- Code AC:

```
#include <bits/stdc++.h>
using namespace std;
int n, a[10005];
vector <int> v;

int main()
{
    freopen ("DAYSOLC.INP", "r", stdin);
    freopen ("DAYSOLC.OUT", "w", stdout);
    cin >> n;
    for (int i=1; i<=n; i++) cin >> a[i];

    for (int mid=2; mid<n; mid++)
    {
      if (a[mid-1] == a[mid] || a[mid] == a[mid+1]) continue;
      else {
        int l = mid, r=mid;
        while (a[l] < a[l-1]) l--;
        while (a[r] < a[r+1]) r++;
        if (l < mid && r > mid && r-l+1 > v.size())
        {
            v.clear();
            for (int i=l; i<=r; i++) v.push_back(a[i]);
        }
      }
    }
    if (v.size() < 3) cout << -1;
    else for (int i=0; i<v.size(); i++) cout << it << " ";
    return 0;
}
```

6. Nổ mìn - NOMIN. *

- Ý tưởng: hướng giải quyết khá giống bài dãy số lòng chảo, nhưng thay vì tìm các đáy chảo, ta sẽ tìm và đếm số lượng các đỉnh.

- Thực hiện:

- Điều kiện đặt mìn là phải đặt ở chướng ngài vật cao hơn những chướng ngại xung quanh nó.
- (*) Duyệt dần từ trái sang phải: nếu phần tử sau lớn hơn phần tử trước thì chắc chắn vị trí đặt mìn phải nằm bên phải so với vị trí hiện tại → dịch con trỏ L sang phải.
- Sau thao tác trên, tại vị trí i, nếu H[i+1] không lớn hơn H[i] → H[i] là một vị trí đặt mìn. Ta tăng số lượng mìn cần đặt (ans++) và tiếp tục nổ sang phải với điều kiện H[i] > H[i+1] đến khi không còn nổ được nữa thì lặp lại thao tác (*).
- Sau mỗi đợt nổ, đặt lại L = R+1 để tìm đỉnh tiếp theo.

→ Độ phức tạp: O(n).

- Code AC:

```
#include<bits/stdc++.h>
using namespace std;
int n, h[500005];

int main()
{
    freopen ("NOMIN.INP", "r", stdin);
    freopen ("NOMIN.OUT", "w", stdout);
    cin >> n;
    for (int i=1; i<=n; i++) cin >> h[i];

    int ans=0, L=1, R;
    while (L <= n) {
        if (h[L+1] > h[L]) L++;
        else {
            ans++; R = L;
            while (R <= n) {
                if (h[R] > h[R+1]) R++;
                else break;
            }
            L = R+1;
        }
    }
    cout << ans; return 0;
}
```

7. Đào vàng - DAOVANG. *

- Ý tưởng tổng quát:

- Đối tượng chặt nhị phân: lực đập của máy khoan.
- Phạm vi: theo giới hạn của X[i], L = 1 và R = 10^9.
- Sắp xếp lại dãy từ bé đến lớn.
- Mỗi lần tìm được lực đập, kiểm tra với K lần khoan có đào được hết vàng hay không. Từ đó thu hẹp phạm vi một cách hợp lý.

- Thực hiện:

- **Hàm check:**
  + Biến ***x*** là lực đập đang xét (từ ***mid*** truyền vào).
  + Gọi ***t = a[1] + x*** là điểm đặt máy khoan. Tức phạm vi đào là ***a[1]*** ←***t*** →***t + x***.
  + Duyệt qua vị trí của các thỏi vàng. Nếu ***a[i] > t+x*** tức là ***a[i]*** nằm ngoài phạm vi đào của máy khoan. Do đó, ta cần đặt thêm một máy khác tại vị trí ***a[i] + x*** để đào khoảng ***a[i]*** ←***a[i] + x*** →***a[i] + 2x***.
  + Sau khi duyệt hết, nếu số lượng máy khoan phải đặt (***cnt***) vượt quá ***k*** có nghĩa là lực đập ***x*** này không thể đào hết ***n*** thỏi vàng trong ***k*** lần.

- **Binary Search trong hàm main:**
  + Nếu hàm check trả về **FALSE** thì ta phải dùng máy khoan với một lực đập lớn hơn ==> Thu hẹp phạm vi tìm kiếm về bên phải.
  + Nếu hàm check trả về **TRUE** tức là lực đập này đã thỏa mãn điều kiện đề bài. Nhưng ta vẫn thử thu hẹp phạm vi về bên trái để xem có thể đào hết ***n*** thỏi vàng trong ***k*** lần với một lực đập bé hơn nữa hay không.

→ Độ phức tạp: O(n * log(10^9)).

- Code AC:

```
#include <bits/stdc++.h>
using namespace std;
int n, k, ans, a[500005];

bool check (int x) {
    int cnt = 1; // số lượng máy khoan cần đặt
    int t = a[1] + x; // vị trí đặt máy khoan
    for (int i=2; i<=n; i++) {
        if (a[i] > t+x) { // a[i] nằm ngoài phạm vi đào
            t = a[i]+x;
            cnt++;
        }
    }
    if(cnt > k) return false; // cần lực đập lớn hơn
    else return true; // thỏa mãn điều kiện
}
int main() {
    freopen ("DAOVANG.INP", "r", stdin);
    freopen ("DAOVANG.OUT", "w", stdout);
    cin >> n >> k;
    for (int i=1; i<=n; i++) cin >> a[i];
    sort (a+1, a+n+1);

    int L = 0, R = 1e9, mid = (l+r)/2;
    while (L <= R) {
        mid = (L + R)/2;
        if (!check(mid)) L = mid+1;
        else { R = mid-1;
               ans = mid;
        }
    }
    cout << ans; return 0;
}
```

8. Mua hàng - MUAHANG. *

- Thông báo:

Suốt thời gian diễn ra contest, chúng mình vẫn chưa thấy lượt AC nào ở bài 8. Có vẻ bài *Mua hàng* vẫn chưa phù hợp để đưa vào contest này. Do đó, chúng mình tạm thời chưa công khai lời giải và sẽ đưa bài này vào một contest khác ở thời điểm thích hợp hơn.

Tuy nhiên, bọn mình vẫn sẽ gợi ý hướng giải quyết trong phần bên dưới. Trong trường hợp bạn đã giải gần ra (***đạt 60+ điểm***) và không biết mình sai chỗ nào, vui lòng đặt câu hỏi, nêu vấn đề trong đoạn chat cộng đồng để được hỗ trợ. Chúc các bạn phá đảo thành công! 🐧

- Gợi ý:

- Đối tượng chặt nhị phân: số lượng suất ăn có thể làm (***mid***).
- Tạo hàm check để kiểm tra có thể làm được mid suất ăn hay không .
- Trong hàm check, bạn hãy sử dụng tất cả dữ kiện đã cho trong input để thực hiện tính toán (làm như một bài toán).
- Khuyến khích dùng nhiều chương trình con để dễ theo dõi code.
  Khuyến nghị: 1 hàm check, 1 hàm tìm kiếm nhị phân và 1 hàm tính số tiền phải trả của mỗi lần mua.

III. Lời kết:

Chân thành cảm ơn các bạn đã theo dõi và ủng hộ Guidelines for Grade 9. Đừng nản lòng nếu bạn chưa AC được các bài khó nha. Ở contest này, bạn chỉ cần đạt khoảng 240+ điểm là đã rất tuyệt vời rồi. Hãy cùng cố gắng và chăm chỉ luyện tập nhiều hơn để hoàn thành mục tiêu của bản thân mình nhé! Cố lên, GG9 luôn đồng hành cùng bạn! ❤️‍🔥

Thông tin liên hệ:

- Fanpage: **Căng TIN 2326**
- Group GG9: **Guidelines for Grade 9**
- Đoạn chat cộng đồng: m.me/ch/Aba9cpzNk_Td6_fl/
- Instagram: __cang_tin_2326__
- Gmail: cangtin2326@gmail.com
- Confessions: https://forms.gle/dvByc9UQ2bRsPQHn8

--- HẾT ---

Hẹn gặp lại các bạn vào "tháng của tuyết và tuần lộc"