Explanation of Klee's SUPER DUPER LARGE Array Solution

Problem Restatement

You are given an array a of length n where:

$$a = [k, k+1, k+2, \ldots, k+n-1]$$

We want to pick an index i ($1 \leq i \leq n$) and compute:

$$x = \left| \sum_{j=1}^{i} a_j - \sum_{j=i+1}^{n} a_j \right|$$

Our goal is to minimize x.

Mathematical Insight

Since the array is an arithmetic progression (AP) starting from k with common difference 1, the j-th element of the array is:

$$a_j = k + j - 1$$

We divide the array at some position i, so the two parts become:

$$\text{First half: } a_1 + a_2 + \cdots + a_i \text{Second half: } a_{i+1} + \cdots + a_n$$

We aim to minimize:

$$x = \left| \sum_{j=1}^{i} a_j - \sum_{j=i+1}^{n} a_j \right|$$

Let us define:

$$\text{Let } m = k + i - 1 \quad \text{(value at index } i\text{)}$$

We do binary search on m, ranging from k to $k + n - 1$.

Let us denote:

- $\text{len}_1 = m - k \quad$ (number of elements in first half)
- $\text{len}_2 = k + n - m \quad$ (number of elements in second half)

The sum of an arithmetic progression is:

$$\text{Sum} = \frac{\text{first term} + \text{last term}}{2} \times \text{number of terms}$$

First half sum:

$$\text{half}_1 = \frac{(k + (m-1)) \times (m-k)}{2}$$

Second half sum:

$$\text{half}_2 = \frac{(m + (k + n - 1)) \times (k + n - m)}{2}$$

Then, compute:

$$\text{diff} = |\text{half}_1 - \text{half}_2|$$

And minimize this over all possible m via binary search.

C Code Implementation

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <math.h>

typedef long long ll;

int main() {
    int t;
    scanf("%d", &t);

    while (t--) {
        ll n, k;
        scanf("%lld %lld", &n, &k);

        ll low = k;
        ll high = k + n - 1;
        ll ans = LLONG_MAX;

        while (low <= high) {
            ll mid = (low + high) / 2;

            ll len1 = mid - k;
            ll half1 = 0;
            if (len1 > 0) {
                ll first = k;
                ll last = mid - 1;
                half1 = (first + last) * len1 / 2;
            }

            ll len2 = k + n - mid;
            ll half2 = 0;
            if (len2 > 0) {
                ll first = mid;
                ll last = k + n - 1;
                half2 = (first + last) * len2 / 2;
            }

            ll diff = llabs(half1 - half2);
            if (diff < ans) ans = diff;

            if (half1 == half2) {
                break;
```

```
            } else if (half1 > half2) {
                high = mid - 1;
            } else {
                low = mid + 1;
            }
        }

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

    return 0;
}
```

Time Complexity

- Each test case runs in $O(\log n)$ due to binary search.
- Efficient for large n and k up to 10^9.