BRAC University Competitive Programming Workshop

Handouts for Day 3

A.J.M Istiaque

Department of CSE
BRAC University
18 April 2025

Binary & Ternary Search Boilerplates

Binary Search (Discrete)

```cpp
bool ok(int m) {
    // ...
    // Write your logic based on the problem
}

int l = 0, r = INF;
//Set a custom upper bound instead of using builtin constants like INT_MAX or LLONG_MAX, because they can cause overflow when calculating mid and other cases. Use a large enough value according to the maximum input or logic of the problem.

int ans = 1; // Default if no valid answer found

while (l <= r) {
    int m = (l + r) / 2;

    if (ok(m)) {
        ans = m; //Update answer (might be the best so far)

        //If you are minimizing (want the smallest m that satisfies ok)
        r = m  1;

        //If you are maximizing (want the largest m that satisfies ok)
        // l = m + 1;
    }
    else {
        //If minimizing:
        l = m + 1;

        //If maximizing:
        // r = m  1;
    }
}
```

Binary Search (Continuous)

```
double l = 0, r = INF, ans=1;

// EPS should be small enough for desired precision.
// If you need x digits of precision, use: EPS = 10^(-x - 1)
const double EPS = 1e-7; // For 6digit precision
// EPS = 1e-4 for up to 3 decimal digits

while (r  l >= EPS) {
    double m = (l + r) / 2.;

    if (ok(m)) {
        ans = m; // Update answer

        // If minimizing:
        r = m;

        // If maximizing:
        // l = m;
    }
    else {
        // If minimizing:
        l = m;

        // If maximizing:
        // r = m;
    }
}
```

Ternary Search (Discrete)

```
int f(int x) {
    // Compute and return value to optimize (e.g., cost, score)
}

int l = 0, r = INF, ans = 1;
while (r  l >= 3) {
    int m1 = l + (r  l) / 3, m2 = r  (r  l) / 3;
    int val1 = f(m1), val2 = f(m2);

    //For Minimization:
    if (val1 > val2) {
        l = m1 + 1;
        if (val2 < ans) {
            ans = val2; // or ans = m2; (depending on the problem)
        }
    }
    else {
        r = m2  1;
        if (val1 < ans) {
            ans = val1; // or ans = m1; (depending on the problem)
        }
    }

    //For Maximization:
    /*
    if (val1 < val2) {
        l = m1 + 1;
        if (val2 > ans) {
            ans = val2; // or ans = m2; (depending on the problem)
        }
    }
    else {
        r = m2  1;
        if (val1 > ans) {
            ans = val1; // or ans = m1; (depending on the problem)
        }
    }
    */
}
```

Ternary Search (Continuous)

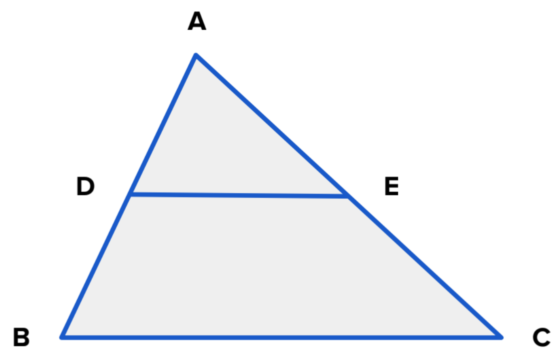
```
const double EPS = 1e-7;

double f(double x) {
    //Do necessary calculation
}

double l = 0, r = INF, ans = 1;

while (r  l > EPS) {
    double m1 = l + (r  l) / 3., m2 = r  (r  l) / 3.;
    double val1 = f(m1), val2 = f(m2);

    //For Minimization:
    if (val1 > val2) {
        l = m1 + 1;
        if (val2 < ans) {
            ans = val2; // or ans = m2; (depending on the problem)
        }
    }
    else {
        r = m2  1;
        if (val1 < ans) {
            ans = val1; // or ans = m1; (depending on the problem)
        }
    }

    //For Maximization:
    /*
    if (val1 < val2) {
        l = m1 + 1;
        if (val2 > ans) {
            ans = val2; // or ans = m2; (depending on the problem)
        }
    }
    else {
        r = m2  1;
        if (val1 > ans) {
            ans = val1; // or ans = m1; (depending on the problem)
        }
    }
    */
}
```

Triangle Partitioning

Problem Setup

You are given:

- Triangle $\triangle ABC$ with sides AB, AC, and BC.
- A line segment DE such that $DE \parallel BC$.
- Triangle $\triangle ADE$ and quadrilateral $BDEC$ share the area ratio $\dfrac{\text{Area of } \triangle ADE}{\text{Area of } BDEC} = k$.



You are required to compute the length of AD such that the ratio $\frac{\text{Area of } \triangle ADE}{\text{Area of } BDEC} = k$ holds, with errors less than 10^6 being ignored.

StepbyStep Solution

Step 1: Geometric Observation

Since $DE \parallel BC$, triangle $\triangle ADE$ is **similar** to triangle $\triangle ABC$ by **AA similarity**:

- $\angle ADE = \angle ABC$ (alternate interior angles)
- $\angle AED = \angle ACB$ (alternate interior angles)

This similarity gives:

$$\frac{AD}{AB} = \frac{AE}{AC} = \frac{DE}{BC}$$

Let $AD = x$. Then, due to similarity:

$$AE = \frac{x}{AB} \cdot AC, \quad DE = \frac{x}{AB} \cdot BC$$

Step 2: Use Heron's Formula to Calculate Areas

Area of $\triangle ABC$: Let $a = AB$, $b = AC$, $c = BC$, and $s = \dfrac{a+b+c}{2}$ be the semiperimeter. Then:

$$\text{Area}_{ABC} = \sqrt{s(sa)(sb)(sc)}$$

Area of $\triangle ADE$: Use the previously scaled sides:

$$AD = x, \quad AE = \frac{x}{AB} \cdot AC, \quad DE = \frac{x}{AB} \cdot BC$$

Let $s' = \dfrac{AD + AE + DE}{2}$, then:

$$\text{Area}_{ADE} = \sqrt{s'(s'AD)(s'AE)(s'DE)}$$

Area of BDEC:

$$\text{Area}_{BDEC} = \text{Area}_{ABC}\text{Area}_{ADE}$$

Step 3: Why Binary Search Works - Monotonic Property

We define a function:

$$f(x) = \frac{\text{Area}_{ADE}}{\text{Area}_{BDEC}}$$

As x (i.e., AD) increases:

- The scaling ratio $\frac{x}{AB}$ increases.
- So, the sides AE and DE grow proportionally.
- Hence, $\triangle ADE$ becomes a larger portion of $\triangle ABC$.
- Therefore, Area_{ADE} increases, and $\text{Area}_{BDEC} = \text{Area}_{ABC}\text{Area}_{ADE}$ decreases.

This means the ratio $f(x)$ is a **strictly increasing function**. In other words:

$$x_1 < x_2 \quad \Rightarrow \quad f(x_1) < f(x_2)$$

This monotonicity is key to applying binary search, as it guarantees that once the area ratio exceeds k, any further increase in x will only push the ratio further up. Hence, we can:

- Move left if $f(x) \geq k$
- Move right if $f(x) < k$

Time Complexity Analysis

We perform binary search on the length of segment AD in the range $[0, AB]$, and we stop when the search range becomes less than a small threshold ε.

Since we perform binary search for each of the T test cases, and each binary search runs for $\mathcal{O}\left(\log_2\left(\frac{AB}{\varepsilon}\right)\right)$ iterations, the overall time complexity is:

$$\boxed{\mathcal{O}\left(T \cdot \log\left(\frac{AB}{\varepsilon}\right)\right)}$$

CPP Code

```cpp
#include <bits/stdc++.h>
using namespace std;

const double EPS = 1e-7; // Precision threshold: errors less than 1e-6 are ignored

void solveCase() {
    double AB, AC, BC, k;
    cin >> AB >> AC >> BC >> k;

    // Calculate total area of triangle ABC using Heron's formula
    double s = (AB + AC + BC) / 2.0;
    double totalArea = sqrt(s * (s  AB) * (s  AC) * (s  BC));

    double l = 0.0, r = AB;

    while (r  l >= EPS) {
        double AD = (l + r) / 2.0;
        double scale = AD / AB;

        double AE = scale * AC;
        double DE = scale * BC;

        // Area of triangle ADE using Heron's formula
        double sADE = (AD + AE + DE) / 2.0;
        double areaADE = sqrt(sADE * (sADE  AD) * (sADE  AE) * (sADE  DE));

        double areaBDEC = totalArea  areaADE;

        if (areaADE / areaBDEC >= k) r = AD;
        else l = AD;
```

```
    }

    cout << fixed << setprecision(7) << l << '\n';
}

int main() {
    cin.tie(0)>sync_with_stdio(0);

    int testCases;
    cin >> testCases;

    for (int caseNum = 1; caseNum <= testCases; ++caseNum) {
        cout << "Case " << caseNum << ": ";
        solveCase();
    }

    return 0;
}
```

Usage of lower_bound and upper_bound

Both `lower_bound` and `upper_bound` are efficient binary search utilities available in STL. They work in $\mathcal{O}(\log n)$ on sorted containers and associative containers.

1. In `vector` or sorted arrays

Precondition: The container must be sorted in nondecreasing order.

- `lower_bound(v.begin(), v.end(), x)`: Returns iterator to the first element $\geq x$
- `upper_bound(v.begin(), v.end(), x)`: Returns iterator to the first element $> x$

Example:

```
vector<int> v = {1, 3, 3, 5, 7, 9};

auto it1 = lower_bound(v.begin(), v.end(), 3); // Points to first 3
auto it2 = upper_bound(v.begin(), v.end(), 3); // Points to 5

int idx1 = it1  v.begin(); // Index of first >= 3 which is 1
int idx2 = it2  v.begin(); // Index of first > 3 which is 3
```

2. In `set` or `multiset`

Important : Many beginners mistakenly use `lower_bound(st.begin(), st.end(), x)` on a `set` or `multiset`. While this may compile, it results in $\mathcal{O}(n)$ time complexity.

Correct usage: Use member functions:

- `st.lower_bound(x)` - Returns iterator to the first element $\geq x$
- `st.upper_bound(x)` - Returns iterator to the first element $> x$

Example (Set):

```
set<int> s = {1, 3, 5, 7};

auto it = s.lower_bound(4); // Points to 5
if (it != s.end()) cout << *it << '\n';
```

Example (Multiset):

```
multiset<int> ms = {1, 3, 3, 5};

auto it1 = ms.lower_bound(3); // First 3
auto it2 = ms.upper_bound(3); // First 5
```

3. Common Use Cases

- Count number of occurrences of x in a vector:

  ```
  int count = upper_bound(v.begin(), v.end(), x)
              lower_bound(v.begin(), v.end(), x);
  ```

- Check if x exists in a set:

  ```
  auto it = st.lower_bound(x);
  if (it != st.end() && *it == x) {
      // x exists
  }
  ```

- Count number of elements strictly less than x:

  ```
  int count = lower_bound(v.begin(), v.end(), x)  v.begin();
  ```

Course and Learning Resources

- Codeforces EDU: Binary Search Course (Pilot)
- CSES Problem Set (Sorting and Searching Section only)

Practice Problems

1. TRICOIN - CodeChef
2. ABC326 C - Peak
3. Codeforces 1490C - Sum of Cubes
4. ABC246 D - 2-variable Function
5. ABC312 C - Invisible Hand
6. Codeforces 1902B - Getting Points
7. ABC255 C - ± 1 Operation 1 (Try to solve this using Ternary Search)
8. Codeforces 1873E - Building an Aquarium
9. Codeforces 1742E - Scuza
10. Codeforces 1574C - Slay the Dragon
11. Codeforces 1850E - Cardboard for Pictures
12. Codeforces 1928B - Equalize
13. Codeforces 1856C - To Become Max
14. LightOJ - Largest Box: Yet Another Geo Problem
15. Codeforces 1999G2 - Ruler (hard version, interactive)
16. ABC395 F - Smooth Occlusion
17. Codeforces 1996F - Bomb

The above problems are sorted in increasing order of difficulty.

Approach to solve a problem:

1. Read the statement carefully and understand what is being asked.
2. Check if the problem has a monotonic property.
3. Before writing code, analyze the time complexity and determine whether your solution will pass within the given time constraints.
4. Stuck for a long time? Take a glance at the editorial or hints and try again.
5. Getting a Wrong Answer (WA)? Check whether:
   - the search space (l, r) is set correctly,
   - any overflow is occurring,
   - any edge case is missing,
   - you have stress tested using a brute-force solution.
6. After solving a problem, see others' solutions to learn new techniques or approaches.