Binary and Ternary Searches

Mike Mirzayanov, Codeforces

Binary Search: Intro

Binary search is a search algorithm that finds the position of a target value within a sorted data.

On each step try middle and reduce the search space by 2.

Invariant: Given $a[0...n-1]$. Maintain l and r, that $a[l \dots r]$ contains the required value.

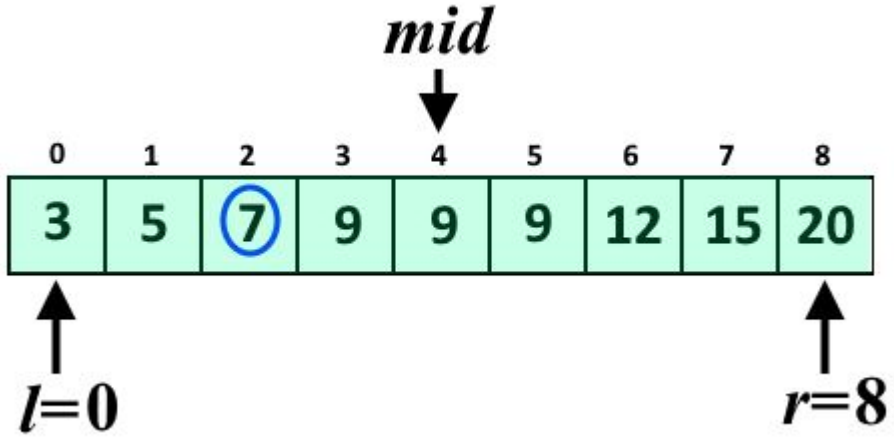
Step: Try middle $mid := l + (r-l)/2$, do $l := mid$ or $r := mid$.

Complexity: $O(\log n)$.

Binary Search: Example

Data: $a = [3, 5, 7, 9, 9, 9, 12, 15, 20], x = 7$

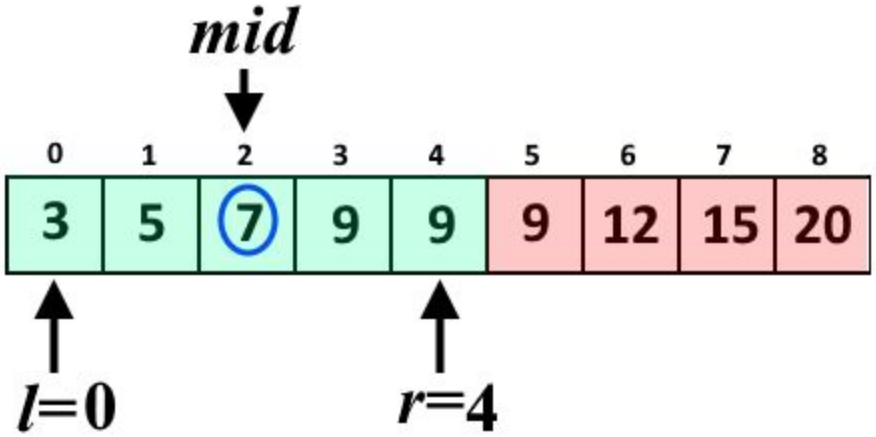
Do binary search until search space is trivial (has length at most 2)



Binary Search: Example

Data: $a = [3, 5, 7, 9, 9, 9, 12, 15, 20], x = 7$

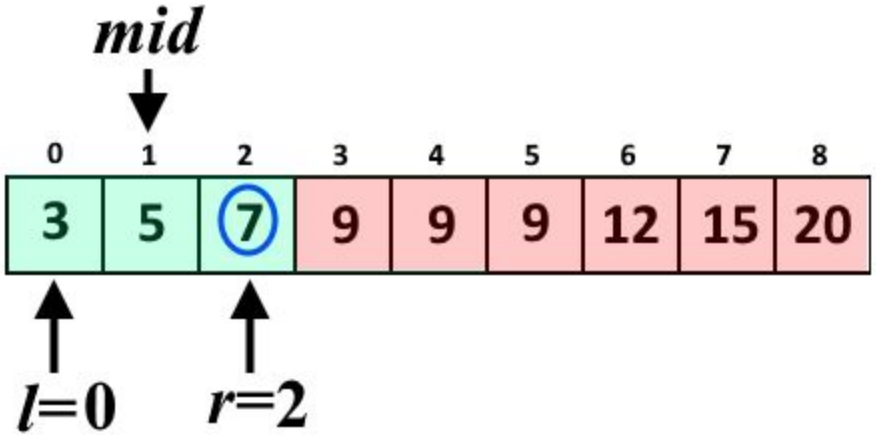
Do binary search until search space is trivial (has length at most 2)



Binary Search: Example

Data: $a = [3, 5, 7, 9, 9, 9, 12, 15, 20], x = 7$

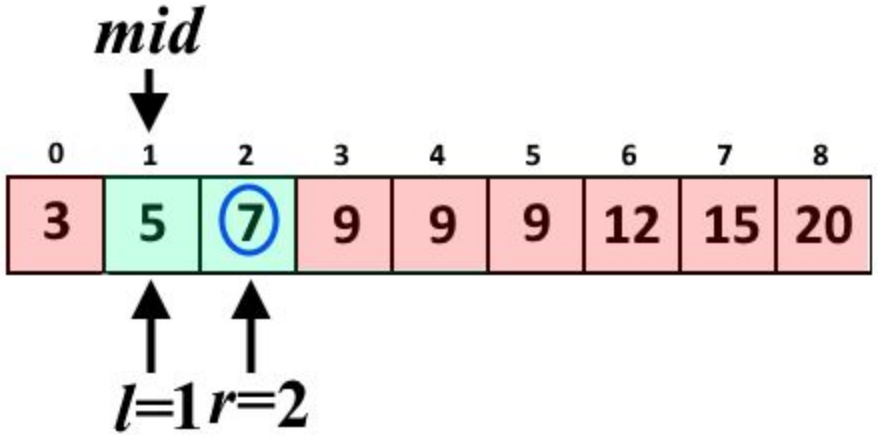
Do binary search until search space is trivial (has length at most 2)



Binary Search: Example

Data: $a = [3, 5, 7, 9, 9, 9, 12, 15, 20], x = 7$

Do binary search until search space is trivial (has length at most 2)



Binary Search: Implementation

```
l = 0, r = n - 1
while r - l > 1
    mid = l + (r - l) / 2
    if a[mid] < b
        l = mid
    else
        r = mid
for i = l ... r
    if a[i] == b
        return i
return NO_SUCH_VALUE
```

// Initialize search space with the whole range
// Is search space greater than 2?

// Choose left or right part

// Search in naive way on trivial search space

Binary Search: Implementation

```
l = 0, r = n - 1
while r - l > 1
    mid = l + (r - l) / 2
    if a[mid] < b
        l = mid
    else
        r = mid
for i = l ... r
    if a[i] == b
        return i
return NO_SUCH_VALUE
```

Such implementation can be easily adopted to find:

- Find first (leftmost) occurrence
- Find last (rightmost) occurrence
- Find first greater or equal (lower bound)
- Find first strictly greater (upper bound)

Binary Search: Standard C++ library

- **binary_search(a.begin(), a.end(), value):** true or false
- **lower_bound(a.begin(), a.end(), value)**
  Returns iterator to the first element greater or equal than value (or end-iterator)
- **upper_bound(a.begin(), a.end(), value)**
  Returns iterator to the first element strictly greater than value (or end-iterator)
- **equal_range(a.begin(), a.end(), value)**: pair of iterators (lower and upper bounds)

Binary Search: Standard C++ library

Example

Renumerate numbers from 0: **[10, 6, 10, 101] => [1, 0, 1, 2]**

```
vector<int> b(a);
sort(b.begin(), b.end());
erase(unique(b.begin(), b.end()), b.end())
for (size_t i = 0; i < a.size(); i++)
    a[i] = lower_bound(b.begin(), b.end(), a[i]) - b.begin();
```

Binary Search: Standard C++ library

Example

Given a, for each $b[i]$ find number of occurrences:

```
sort(a.begin(), a.end());
for (int b_i: b) {
    auto range = equal_range(a.begin(), a.end(), b_i);
    cout << range.second - range.first << endl;
}
```

Binary Search: Binary Search Answer

In many problems you can “guess” the answer using a binary search. In such problems it should be some monotonic nature while you increase candidate to answer.

Example 1: Find the smallest size n of square that it can fit at least k non-overlapping axb rectangles (they can’t be rotated).

Solution: Find such minimal positive integer n that $(n/a)\cdot(n/b) \geq k$. Do binary search by function $f(n)=(n/a)\cdot(n/b)$ to find first (minimal) n that $f(n) \geq k$. Easy to see, that $f(n)$ monotonically increases.

Binary Search: Binary Search Answer

Example 1: Find the smallest size n of square that it can fit at least k non-overlapping axb rectangles (they can't be rotated).

Constraints: $1 \leq k, a, b \leq 10^6$.

```
l = 1, r = 1E9 // think, why 10^9
while r - l > 1
    long long mid = l + (r - l) / 2
    if (mid / a) * (mid / b) < k
        l = mid
    else
        r = mid
for i = l ... r
    if (i / a) * (i / b) >= k
        return i
```

Binary Search: Binary Search Answer

Example 2: Given n spool of thread. The i-th contains a_i of thread. You can cut thread to cutoff k pieces of thread of equal (same) length. What is the maximal possible length of k pieces?

Solution: $f(x)$ = maximal number of pieces to cut if their length is x. Now the problem is: find such maximal x that $f(x) \geq k$. Easy to see, that $f(x)$ monotonically decreases.

Binary Search: Floating-Point Numbers

Consider $f(x)$ is monotonically increases. You are to find the root of the equation $f(x)=C$ on the segment $[l, r]$. Use exactly the same idea, but other condition to stop the binary search.

```
for 100 times:
    double mid = (l + r) / 2
    if f(mid) < C
        l = mid
    else
        r = mid
```

Binary Search: Floating-Point Numbers

Using a binary search you can find the root of equation $f(x)=C$ on the segment $[l, r]$ for any increasing/decreasing function $f(x)$.

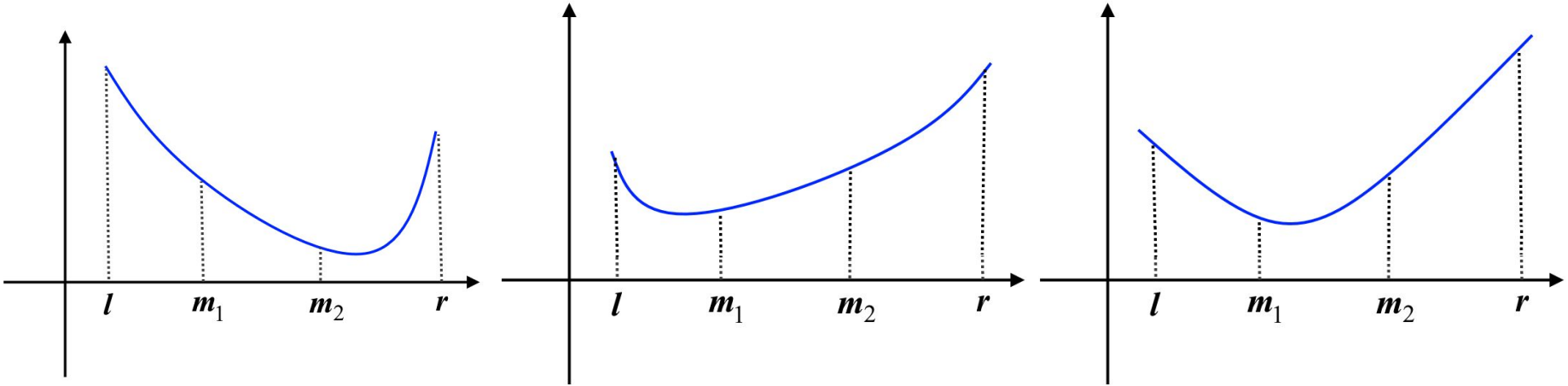
Ternary Search: Basics

You are given a convex downward function $f(x)$. Find the minimum value of the segment $[l, r]$.

Idea: divide the segment $[l, r]$ on 3 parts (with two extra points m_1 and m_2) and depending on $f(m_1)$ and $f(m_2)$ (just compare them) find continue with the segment $[l, m_2]$ or $[m_1, r]$.

Conclusion: if m_1 and m_2 divide $[l, r]$ on three equal parts, we reduce search space by 1.5 times on each iteration.

Ternary Search: Basics



Ternary Search: Implementation

```
for 200 times: // depends on initial l and r
    double m1 = l + (r - l) / 3
    double m2 = r - (r - l) / 3
    if f(m1) < f(m2):
        r = m2
    else:
        l = m1
```

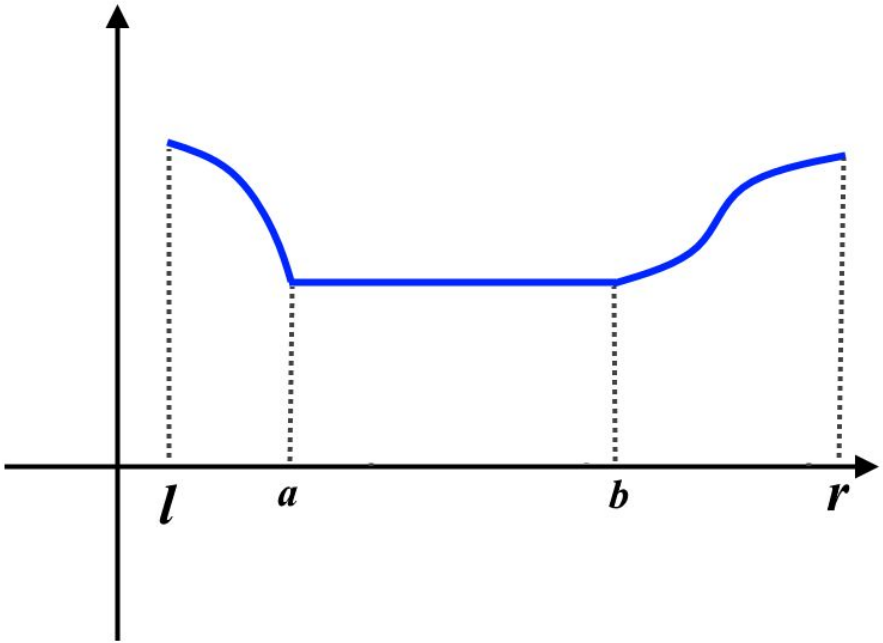
Ternary Search: Implementation

We **never** used that m_1 and m_2 divide segment on equal parts. Try m_1 and m_2 closer to the middle.

```
for 125 times: // depends on initial l and r
    double m1 = l + (r - l) * 0.45
    double m2 = r - (r - l) * 0.45
    if f(m1) < f(m2):
        r = m2
    else:
        l = m1
```

Ternary Search: Notes

Works on unimodal functions.



- Strictly decreases on $[l, a]$
- Constant on $[a, b]$
- Strictly increases in $[b, r]$

Ternary Search: Notes

- Inverse the sign to find maximum of a convex upward (upward unimodal) functions.
- Sum of convex functions is a convex function

Example (Supermarket Location Problem): given weighted points on the axis Ox, find the point to minimize weighted sum of distances.

Ternary Search: Minimum of a Smooth Function

Idea: split the segment on many small segments. On each of them with high probability the function is unimodal. Independently on each small segment find minimum with ternary search. Choose the global minimum among them.