ITMO UNIVERSITY

How to Win Coding Competitions: Secrets of Champions

Week 3: Sorting and Search Algorithms
Lecture 11: Implementations of binary search

Maxim Buzdalov
Saint Petersburg 2016

Binary search: Recall

```
function $\textsc{BinarySearch}(F, \textsc{Avg}, D_{\min}, D_{\max})$
    $L \leftarrow D_{\min}, R \leftarrow D_{\max}, V_{\min} \leftarrow F(L), V_{\max} \leftarrow F(R)$
    if $V_{\min} = 1$ then return $\langle \textsc{Null}, D_{\min} \rangle$ end if
    if $V_{\max} = -1$ then return $\langle D_{\max}, \textsc{Null} \rangle$ end if
    if $V_{\min} = 0$ then return $\langle D_{\min}, D_{\min} \rangle$ end if
    if $V_{\max} = 0$ then return $\langle D_{\max}, D_{\max} \rangle$ end if
    for ever do
        $M \leftarrow \textsc{Avg}(L, R)$
        if $M = L$ or $M = R$ then return $\langle L, R \rangle$ end if
        $v \leftarrow F(M)$
        if $v = 0$ then return $\langle M, M \rangle$ end if
        if $v = -1$ then $L \leftarrow M$ else $R \leftarrow M$ end if
    end for
end function
```

Binary search: Example integer implementation

Let's implement the pseudocode for searching an element in an array

Binary search: Example integer implementation

Let’s implement the pseudocode for searching an element in an array

```
pair<int, int> bin_search(int *a, int size, int what) {
    int l = 0, r = size - 1, vMin = a[l], vMax = a[r];
    if (vMin > what) return make_pair(-1, l);
    if (vMax < what) return make_pair(r, size);
    if (vMin == what) return make_pair(l, l);
    if (vMax == what) return make_pair(r, r);
    while (true) {
        int m = (l + r) / 2;
        if (l == m || r == m) return make_pair(l, r);
        int v = a[m];
        if (v == what) return make_pair(m, m);
        if (v < what) l = m; else r = m;
    }
}
```

Binary search: Example integer implementation

Let’s implement the pseudocode for searching an element in an array

```
pair<int, int> bin_search(int *a, int size, int what) {
    int l = 0, r = size - 1, vMin = a[l], vMax = a[r];
    if (vMin > what) return make_pair(-1, l);
    if (vMax < what) return make_pair(r, size);
    if (vMin == what) return make_pair(l, l);
    if (vMax == what) return make_pair(r, r);
    while (true) {
        int m = (l + r) / 2;
        if (l == m || r == m) return make_pair(l, r);
        int v = a[m];
        if (v == what) return make_pair(m, m);
        if (v < what) l = m; else r = m;
    }
}
```

Okay, let’s test it!

Testing procedure[1]:

- Generate a random int array of size N and sort it
- Generate 10^7 random ints for querying them
- Perform all queries and check their answers for correctness
- Measure and report the time for all queries

[1] Source code: `binsearch-1.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementation

Testing procedure[1]:

- Generate a random int array of size N and sort it
- Generate 10^7 random ints for querying them
- Perform all queries and check their answers for correctness
- Measure and report the time for all queries

```
maxbuzz $ ./binsearch-1 50000
Generating array... done!
Sorting array... done!
Generating queries... done!
Doing 10000000 binary searches... done!
Time: 1033
```

[1] Source code: `binsearch-1.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementation

Testing procedure[1]:

- Generate a random int array of size N and sort it
- Generate 10^7 random ints for querying them
- Perform all queries and check their answers for correctness
- Measure and report the time for all queries

```
maxbuzz $ ./binsearch-1 50000
Generating array...  done!
Sorting array...  done!
Generating queries...  done!
Doing 10000000 binary searches...  done!
Time:  1033
```

50000	1033

[1] Source code: `binsearch-1.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementation

Testing procedure[1]:

- Generate a random int array of size N and sort it
- Generate 10^7 random ints for querying them
- Perform all queries and check their answers for correctness
- Measure and report the time for all queries

```
maxbuzz $ ./binsearch-1 100000
Generating array... done!
Sorting array... done!
Generating queries... done!
Doing 10000000 binary searches... done!
Time: 1124
```

50000	1033

[1] Source code: `binsearch-1.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementation

Testing procedure[1]:

- Generate a random int array of size N and sort it
- Generate 10^7 random ints for querying them
- Perform all queries and check their answers for correctness
- Measure and report the time for all queries

```
maxbuzz $ ./binsearch-1 100000
Generating array... done!
Sorting array... done!
Generating queries... done!
Doing 10000000 binary searches... done!
Time: 1124
```

50000	1033
100000	1124

[1] Source code: binsearch-1.cpp at https://github.com/mbuzdalov/i2cp-code

Binary search: Testing example implementation

Testing procedure[1]:

- Generate a random int array of size N and sort it
- Generate 10^7 random ints for querying them
- Perform all queries and check their answers for correctness
- Measure and report the time for all queries

```
maxbuzz $ ./binsearch-1 200000
Generating array... done!
Sorting array... done!
Generating queries... done!
Doing 10000000 binary searches... done!
Time: 1252
```

50000	1033
100000	1124

[1] Source code: `binsearch-1.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementation

Testing procedure[1]:

- Generate a random int array of size N and sort it
- Generate 10^7 random ints for querying them
- Perform all queries and check their answers for correctness
- Measure and report the time for all queries

```
maxbuzz $ ./binsearch-1 200000
Generating array...  done!
Sorting array...  done!
Generating queries...  done!
Doing 10000000 binary searches...  done!
Time:  1252
```

50000	1033
100000	1124
200000	1252

[1] Source code: `binsearch-1.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementation

Testing procedure[1]:

- Generate a random int array of size N and sort it
- Generate 10^7 random ints for querying them
- Perform all queries and check their answers for correctness
- Measure and report the time for all queries

```
maxbuzz $ ./binsearch-1 400000
Generating array...  done!
Sorting array...  done!
Generating queries...  done!
Doing 10000000 binary searches...  done!
Time:  1371
```

50000	1033
100000	1124
200000	1252

[1] Source code: `binsearch-1.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementation

Testing procedure[1]:

- Generate a random int array of size N and sort it
- Generate 10^7 random ints for querying them
- Perform all queries and check their answers for correctness
- Measure and report the time for all queries

```
maxbuzz $ ./binsearch-1 400000
Generating array...  done!
Sorting array...  done!
Generating queries...  done!
Doing 10000000 binary searches...  done!
Time:  1371
```

50000	1033
100000	1124
200000	1252
400000	1371

[1] Source code: binsearch-1.cpp at https://github.com/mbuzdalov/i2cp-code

Binary search: Testing example implementation

Testing procedure[1]:

- Generate a random int array of size N and sort it
- Generate 10^7 random ints for querying them
- Perform all queries and check their answers for correctness
- Measure and report the time for all queries

```
maxbuzz $ ./binsearch-1 800000
Generating array... done!
Sorting array... done!
Generating queries... done!
Doing 10000000 binary searches... done!
Time: 1598
```

50000	1033
100000	1124
200000	1252
400000	1371

[1] Source code: `binsearch-1.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementation

Testing procedure[1]:

- Generate a random int array of size N and sort it
- Generate 10^7 random ints for querying them
- Perform all queries and check their answers for correctness
- Measure and report the time for all queries

```
maxbuzz $ ./binsearch-1 800000
Generating array... done!
Sorting array... done!
Generating queries... done!
Doing 10000000 binary searches... done!
Time: 1598
```

50000	1033
100000	1124
200000	1252
400000	1371
800000	1598

[1] Source code: `binsearch-1.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementation

Testing procedure[1]:

- Generate a random int array of size N and sort it
- Generate 10^7 random ints for querying them
- Perform all queries and check their answers for correctness
- Measure and report the time for all queries

```
maxbuzz $ ./binsearch-1 1600000
Generating array...  done!
Sorting array...  done!
Generating queries...  done!
Doing 10000000 binary searches...  done!
Time:  2231
```

50000	1033
100000	1124
200000	1252
400000	1371
800000	1598

[1] Source code: `binsearch-1.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementation

Testing procedure[1]:

- Generate a random int array of size N and sort it
- Generate 10^7 random ints for querying them
- Perform all queries and check their answers for correctness
- Measure and report the time for all queries

```
maxbuzz $ ./binsearch-1 1600000
Generating array...  done!
Sorting array...  done!
Generating queries...  done!
Doing 10000000 binary searches...  done!
Time:  2231
```

50000	1033
100000	1124
200000	1252
400000	1371
800000	1598
1600000	2231

[1] Source code: `binsearch-1.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementation

Testing procedure[1]:

- Generate a random int array of size N and sort it
- Generate 10^7 random ints for querying them
- Perform all queries and check their answers for correctness
- Measure and report the time for all queries

```
maxbuzz $ ./binsearch-1 16000000
Generating array...  done!
Sorting array...  done!
Generating queries...  done!
Doing 10000000 binary searches...  done!
Time:  4268
```

50000	1033
100000	1124
200000	1252
400000	1371
800000	1598
1600000	2231

[1] Source code: `binsearch-1.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementation

Testing procedure[1]:

- Generate a random int array of size N and sort it
- Generate 10^7 random ints for querying them
- Perform all queries and check their answers for correctness
- Measure and report the time for all queries

```
maxbuzz $ ./binsearch-1 16000000
Generating array...  done!
Sorting array...  done!
Generating queries...  done!
Doing 10000000 binary searches...  done!
Time:  4268
```

50000	1033
100000	1124
200000	1252
400000	1371
800000	1598
1600000	2231
16000000	4268

[1] Source code: `binsearch-1.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementation

Testing procedure[1]:

- Generate a random int array of size N and sort it
- Generate 10^7 random ints for querying them
- Perform all queries and check their answers for correctness
- Measure and report the time for all queries

```
maxbuzz $ ./binsearch-1 160000000
Generating array...  done!
Sorting array...  done!
Generating queries...  done!
Doing 10000000 binary searches...  done!
Time:  7529
```

50000	1033
100000	1124
200000	1252
400000	1371
800000	1598
1600000	2231
16000000	4268

[1] Source code: `binsearch-1.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementation

Testing procedure[1]:

- Generate a random int array of size N and sort it
- Generate 10^7 random ints for querying them
- Perform all queries and check their answers for correctness
- Measure and report the time for all queries

```
maxbuzz $ ./binsearch-1 160000000
Generating array...  done!
Sorting array...  done!
Generating queries...  done!
Doing 10000000 binary searches...  done!
Time:  7529
```

50000	1033
100000	1124
200000	1252
400000	1371
800000	1598
1600000	2231
16000000	4268
160000000	7529

[1] Source code: `binsearch-1.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementation

Testing procedure[1]:

- Generate a random int array of size N and sort it
- Generate 10^7 random ints for querying them
- Perform all queries and check their answers for correctness
- Measure and report the time for all queries

```
maxbuzz $ ./binsearch-1 1600000000
Generating array...  done!
Sorting array...  done!
Generating queries...  done!
Doing 10000000 binary searches...Segmentation fault
```

50000	1033
100000	1124
200000	1252
400000	1371
800000	1598
1600000	2231
16000000	4268
160000000	7529

[1] Source code: `binsearch-1.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementation

Testing procedure[1]:

- Generate a random int array of size N and sort it
- Generate 10^7 random ints for querying them
- Perform all queries and check their answers for correctness
- Measure and report the time for all queries

```
maxbuzz $ ./binsearch-1 1600000000
Generating array...  done!
Sorting array...  done!
Generating queries...  done!
Doing 10000000 binary searches...Segmentation fault
```

What has just happened?

50000	1033
100000	1124
200000	1252
400000	1371
800000	1598
1600000	2231
16000000	4268
160000000	7529

[1] Source code: `binsearch-1.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Correcting example implementation

```
pair<int, int> bin_search(int *a, int size, int what) {
    int l = 0, r = size - 1, vMin = a[l], vMax = a[r];
    if (vMin > what) return make_pair(-1, l);
    if (vMax < what) return make_pair(r, size);
    if (vMin == what) return make_pair(l, l);
    if (vMax == what) return make_pair(r, r);
    while (true) {
        int m = (l + r) / 2;
        if (l == m || r == m) return make_pair(l, r);
        int v = a[m];
        if (v == what) return make_pair(m, m);
        if (v < what) l = m; else r = m;
    }
}
```

Binary search: Correcting example implementation

```
pair<int, int> bin_search(int *a, int size, int what) {
    int l = 0, r = size - 1, vMin = a[l], vMax = a[r];
    if (vMin > what) return make_pair(-1, l);
    if (vMax < what) return make_pair(r, size);
    if (vMin == what) return make_pair(l, l);
    if (vMax == what) return make_pair(r, r);
    while (true) {
        int m = (l + r) / 2;
        if (l == m || r == m) return make_pair(l, r);
        int v = a[m];
        if (v == what) return make_pair(m, m);
        if (v < what) l = m; else r = m;
    }
}
```

Here is the problem: integer overflow!

Binary search: Correcting example implementation

```
pair<int, int> bin_search(int *a, int size, int what) {
    int l = 0, r = size - 1, vMin = a[l], vMax = a[r];
    if (vMin > what) return make_pair(-1, l);
    if (vMax < what) return make_pair(r, size);
    if (vMin == what) return make_pair(l, l);
    if (vMax == what) return make_pair(r, r);
    while (true) {
        int m = (l + r) / 2;
        if (l == m || r == m) return make_pair(l, r);
        int v = a[m];
        if (v == what) return make_pair(m, m);
        if (v < what) l = m; else r = m;
    }
}
```

Here is the problem: integer overflow!

- $(1500000000 + 1600000000) / 2 =$ -597483648

Binary search: Correcting example implementation

```
pair<int, int> bin_search(int *a, int size, int what) {
    int l = 0, r = size - 1, vMin = a[l], vMax = a[r];
    if (vMin > what) return make_pair(-1, l);
    if (vMax < what) return make_pair(r, size);
    if (vMin == what) return make_pair(l, l);
    if (vMax == what) return make_pair(r, r);
    while (true) {
        int m = l + (r - l) / 2;
        if (l == m || r == m) return make_pair(l, r);
        int v = a[m];
        if (v == what) return make_pair(m, m);
        if (v < what) l = m; else r = m;
    }
}
```

Here is the problem: integer overflow!

- (1500000000 + 1600000000) / 2 = -597483648
- Example for how to fix it

Binary search: Testing corrected implementation

Testing procedure[1]:

- Generate a random int array of size N and sort it
- Generate 10^7 random ints for querying them
- Perform all queries and check their answers for correctness
- Measure and report the time for all queries

```
maxbuzz $ ./binsearch-2 1600000000
Generating array... done!
Sorting array... done!
Generating queries... done!
Doing 10000000 binary searches... done!
Time: 11428
```

50000	1033
100000	1124
200000	1252
400000	1371
800000	1598
1600000	2231
16000000	4268
160000000	7529
1600000000	11428

[1] Source code: `binsearch-2.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Example double implementation

The pseudocode works only with finite search domains. What about real numbers?

Binary search: Example double implementation

The pseudocode works only with finite search domains. What about real numbers?

```
pair<double, double> bin_search(double (*f)(double), double l, double r) {
    double vMin = f(l), vMax = f(r);
    if (vMin > 0) return make_pair(l - 1, l);
    if (vMax < 0) return make_pair(r, r + 1);
    if (vMin == 0) return make_pair(l, l);
    if (vMax == 0) return make_pair(r, r);
    while (true) {
        double m = (l + r) / 2;
        if (l == m || r == m) return make_pair(l, r);
        double v = f(m);
        if (v == 0) return make_pair(m, m);
        if (v < 0) l = m; else r = m;
    }
}
```

Binary search: Example double implementation

The pseudocode works only with finite search domains. What about real numbers?

```
pair<double, double> bin_search(double (*f)(double), double l, double r) {
    double vMin = f(l), vMax = f(r);
    if (vMin > 0) return make_pair(l - 1, l);
    if (vMax < 0) return make_pair(r, r + 1);
    if (vMin == 0) return make_pair(l, l);
    if (vMax == 0) return make_pair(r, r);
    while (true) {
        double m = (l + r) / 2;
        if (l == m || r == m) return make_pair(l, r);
        double v = f(m);
        if (v == 0) return make_pair(m, m);
        if (v < 0) l = m; else r = m;
    }
}
```

This code searches for a root of the given $\mathbb{R} \to \mathbb{R}$ function. Will it terminate?

Binary search: Example double implementation

The pseudocode works only with finite search domains. What about real numbers?

```
pair<double, double> bin_search(double (*f)(double), double l, double r) {
    double vMin = f(l), vMax = f(r);
    if (vMin > 0) return make_pair(l - 1, l);
    if (vMax < 0) return make_pair(r, r + 1);
    if (vMin == 0) return make_pair(l, l);
    if (vMax == 0) return make_pair(r, r);
    while (true) {
        double m = (l + r) / 2;
        if (l == m || r == m) return make_pair(l, r);
        double v = f(m);
        if (v == 0) return make_pair(m, m);
        if (v < 0) l = m; else r = m;
    }
}
```

This code searches for a root of the given $\mathbb{R} \to \mathbb{R}$ function. Will it terminate?

Yes it will, because computer real numbers are finite!

Let us also examine two common ways to implement real-valued binary search.

Binary search: Example double implementation

Let us also examine two common ways to implement real-valued binary search.

Epsilon-based.

```
pair<double, double> bin_search_eps(double (*f)(double), double l, double r) {
    const double epsilon = 1e-9;
    while (r - l > epsilon) {
        double m = (l + r) / 2;
        if (f(m) < 0) l = m; else r = m;
    }
    return make_pair(l, r);
}
```

Binary search: Example double implementation

Let us also examine two common ways to implement real-valued binary search.

Epsilon-based.

```
pair<double, double> bin_search_eps(double (*f)(double), double l, double r) {
    const double epsilon = 1e-9;
    while (r - l > epsilon) {
        double m = (l + r) / 2;
        if (f(m) < 0) l = m; else r = m;
    }
    return make_pair(l, r);
}
```

Iteration limit.

```
pair<double, double> bin_search_iter(double (*f)(double), double l, double r) {
    const int max_iterations = 50;
    for (int iter = 0; iter < max_iterations; ++iter) {
        double m = (l + r) / 2;
        if (f(m) < 0) l = m; else r = m;
    }
    return make_pair(l, r);
}
```

Binary search: Testing example implementations

Setup[1]:

- $f_1(x) = x$
- $f_2(x) = x + 412349128419.77615$
- $f_3(x) = \operatorname{atan}(x) + x + 17$

- Left bound: -10^{12}
- Right bound: 10^{11}
- Output precision: 17 digits

[1] Source code: `binsearch-3.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementations

Setup[1]:

- $f_1(x) = x$
- $f_2(x) = x + 412349128419.77615$
- $f_3(x) = \mathrm{atan}(x) + x + 17$

- Left bound: -10^{12}
- Right bound: 10^{11}
- Output precision: 17 digits

```
Binary search with exact termination
  Function x: 1110 iterations
    f0(0) = 0
    f0(0) = 0
  Function x + 412349128419.77615: 53 iterations
    f1(-412349128419.77612) = 0
    f1(-412349128419.77612) = 0
  Function atan(x) + x + 17: 87 iterations
    f2(-15.493656816339765) = 0
    f2(-15.493656816339765) = 0
```

[1] Source code: `binsearch-3.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementations

Setup[1]:

- $f_1(x) = x$
- $f_2(x) = x + 412349128419.77615$
- $f_3(x) = \mathrm{atan}(x) + x + 17$

- Left bound: -10^{12}
- Right bound: 10^{11}
- Output precision: 17 digits

```
Binary search with exact termination
  Function x: 1110 iterations
    f0(0) = 0
    f0(0) = 0
  Function x + 412349128419.77615: 53 iterations
    f1(-412349128419.77612) = 0
    f1(-412349128419.77612) = 0
  Function atan(x) + x + 17: 87 iterations
    f2(-15.493656816339765) = 0
    f2(-15.493656816339765) = 0
```

- Always terminates

[1] Source code: `binsearch-3.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementations

Setup[1]:

- $f_1(x) = x$
- $f_2(x) = x + 412349128419.77615$
- $f_3(x) = \mathrm{atan}(x) + x + 17$

- Left bound: -10^{12}
- Right bound: 10^{11}
- Output precision: 17 digits

```
Binary search with exact termination
  Function x: 1110 iterations
    f0(0) = 0
    f0(0) = 0
  Function x + 412349128419.77615: 53 iterations
    f1(-412349128419.77612) = 0
    f1(-412349128419.77612) = 0
  Function atan(x) + x + 17: 87 iterations
    f2(-15.493656816339765) = 0
    f2(-15.493656816339765) = 0
```

- Always terminates
- May require a lot of work around zero

[1] Source code: `binsearch-3.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementations

Setup[1]:

- $f_1(x) = x$
- $f_2(x) = x + 412349128419.77615$
- $f_3(x) = \mathrm{atan}(x) + x + 17$

- Left bound: -10^{12}
- Right bound: 10^{11}
- Output precision: 17 digits

```
Binary search with exact termination
  Function x: 1110 iterations
    f0(0) = 0
    f0(0) = 0
  Function x + 412349128419.77615: 53 iterations
    f1(-412349128419.77612) = 0
    f1(-412349128419.77612) = 0
  Function atan(x) + x + 17: 87 iterations
    f2(-15.493656816339765) = 0
    f2(-15.493656816339765) = 0
```

- Always terminates
- May require a lot of work around zero
  - And remember, doubles very close to zero may be very slow

[1] Source code: `binsearch-3.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Setup[1]:

- $f_1(x) = x$
- $f_2(x) = x + 412349128419.77615$
- $f_3(x) = \text{atan}(x) + x + 17$

- Left bound: -10^{12}
- Right bound: 10^{11}
- Output precision: 17 digits

```
Binary search with exact termination
  Function x: 1110 iterations
    f0(0) = 0
    f0(0) = 0
  Function x + 412349128419.77615: 53 iterations
    f1(-412349128419.77612) = 0
    f1(-412349128419.77612) = 0
  Function atan(x) + x + 17: 87 iterations
    f2(-15.49365816339765) = 0
    f2(-15.49365816339765) = 0
```

- Always terminates
- May require a lot of work around zero
  - And remember, doubles very close to zero may be very slow
  - IEEE 754 subnormal values for double: $[-2.225 \cdot 10^{-308}; 2.225 \cdot 10^{-308}]$, excluding zero

[1] Source code: `binsearch-3.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementations

Setup[1]:

- $f_1(x) = x$
- $f_2(x) = x + 412349128419.77615$
- $f_3(x) = \mathrm{atan}(x) + x + 17$

- Left bound: -10^{12}
- Right bound: 10^{11}
- Output precision: 17 digits

```
Binary search with epsilon
  Function x: 70 iterations
    f0(-8.4703294725430034e-10) = -8.4703294725430034e-10
    f0(8.4703294725430034e-11) = 8.4703294725430034e-11
  Function x + 412349128419.77615: FAILED TO CONVERGE
    f1(-412349128419.77618) = -6.103515625e-05
    f1(-412349128419.77612) = 0
  Function atan(x) + x + 17: 70 iterations
    f2(-15.493656816897174) = -5.5972293466766132e-10
    f2(-15.493656815965437) = 3.7588065993077718e-10
```

[1] Source code: `binsearch-3.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementations

Setup[1]:

- $f_1(x) = x$
- $f_2(x) = x + 412349128419.77615$
- $f_3(x) = \text{atan}(x) + x + 17$

- Left bound: -10^{12}
- Right bound: 10^{11}
- Output precision: 17 digits

```
Binary search with epsilon
  Function x: 70 iterations
    f0(-8.470329472543003e-10) = -8.470329472543003e-10
    f0(8.470329472543003e-11) = 8.470329472543003e-11
  Function x + 412349128419.77615: FAILED TO CONVERGE
    f1(-412349128419.77618) = -6.103515625e-05
    f1(-412349128419.77612) = 0
  Function atan(x) + x + 17: 70 iterations
    f2(-15.493656816897174) = -5.597229346676613e-10
    f2(-15.493656815965437) = 3.758806599307718e-10
```

- Rather precise when converges

[1] Source code: `binsearch-3.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementations

Setup[1]:

- $f_1(x) = x$
- $f_2(x) = x + 412349128419.77615$
- $f_3(x) = \text{atan}(x) + x + 17$

- Left bound: -10^{12}
- Right bound: 10^{11}
- Output precision: 17 digits

```
Binary search with epsilon
  Function x: 70 iterations
    f0(-8.4703294725430034e-10) = -8.4703294725430034e-10
    f0(8.4703294725430034e-11) = 8.4703294725430034e-11
  Function x + 412349128419.77615: FAILED TO CONVERGE
    f1(-412349128419.77618) = -6.103515625e-05
    f1(-412349128419.77612) = 0
  Function atan(x) + x + 17: 70 iterations
    f2(-15.493656816897174) = -5.5972293466766132e-10
    f2(-15.493656815965437) = 3.758806599307718e-10
```

- Rather precise when converges
- But may not converge :(

[1] Source code: `binsearch-3.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementations

Setup[1]:

- $f_1(x) = x$
- $f_2(x) = x + 412349128419.77615$
- $f_3(x) = \text{atan}(x) + x + 17$

- Left bound: -10^{12}
- Right bound: 10^{11}
- Output precision: 17 digits

```
Binary search with epsilon
  Function x: 70 iterations
    f0(-8.4703294725430034e-10) = -8.4703294725430034e-10
    f0(8.4703294725430034e-11) = 8.4703294725430034e-11
  Function x + 412349128419.77615: FAILED TO CONVERGE
    f1(-412349128419.77618) = -6.103515625e-05
    f1(-412349128419.77612) = 0
  Function atan(x) + x + 17: 70 iterations
    f2(-15.493656816897174) = -5.5972293466766132e-10
    f2(-15.493656815965437) = 3.7588065993077718e-10
```

- Rather precise when converges
- But may not converge :(
  - Two adjacent doubles may have a difference bigger than your epsilon

[1] Source code: `binsearch-3.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementations

Setup[1]:

- $f_1(x) = x$
- $f_2(x) = x + 412349128419.77615$
- $f_3(x) = \mathrm{atan}(x) + x + 17$

- Left bound: -10^{12}
- Right bound: 10^{11}
- Output precision: 17 digits

```
Binary search with iteration limit 50
  Function x: 50 iterations
    f0(-0.00088817841970012523) = -0.00088817841970012523
    f0(8.8817841970012523e-05) = 8.8817841970012523e-05
  Function x + 412349128419.77615: 50 iterations
    f1(-412349128419.77625) = -0.0001220703125
    f1(-412349128419.77527) = 0.0008544921875
  Function atan(x) + x + 17: 50 iterations
    f2(-15.494094895984745) = -0.00043989694897561549
    f2(-15.493117899723075) = 0.00054115236727980687
```

[1] Source code: `binsearch-3.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementations

Setup[1]:

- $f_1(x) = x$
- $f_2(x) = x + 412349128419.77615$
- $f_3(x) = \text{atan}(x) + x + 17$

- Left bound: -10^{12}
- Right bound: 10^{11}
- Output precision: 17 digits

```
Binary search with iteration limit 50
  Function x: 50 iterations
    f0(-0.00088817841970012523) = -0.00088817841970012523
    f0(8.8817841970012523e-05) = 8.8817841970012523e-05
  Function x + 412349128419.77615: 50 iterations
    f1(-412349128419.77625) = -0.0001220703125
    f1(-412349128419.77527) = 0.0008544921875
  Function atan(x) + x + 17: 50 iterations
    f2(-15.494094895984745) = -0.00043989694897561549
    f2(-15.493117899723075) = 0.00054115236727980687
```

- Has a predictable running time

[1] Source code: `binsearch-3.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementations

Setup[1]:

- $f_1(x) = x$
- $f_2(x) = x + 412349128419.77615$
- $f_3(x) = \mathrm{atan}(x) + x + 17$

- Left bound: -10^{12}
- Right bound: 10^{11}
- Output precision: 17 digits

```
Binary search with iteration limit 50
  Function x: 50 iterations
    f0(-0.00088817841970012523) = -0.00088817841970012523
    f0(8.8817841970012523e-05) = 8.8817841970012523e-05
  Function x + 412349128419.77615: 50 iterations
    f1(-412349128419.77625) = -0.0001220703125
    f1(-412349128419.77527) = 0.0008544921875
  Function atan(x) + x + 17: 50 iterations
    f2(-15.494094895984745) = -0.00043989694897561549
    f2(-15.493117899723075) = 0.00054115236727980687
```

- Has a predictable running time
- But the number of iterations should be accurately adjusted

[1] Source code: `binsearch-3.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search: Testing example implementations

Setup[1]:

- $f_1(x) = x$
- $f_2(x) = x + 412349128419.77615$
- $f_3(x) = \mathrm{atan}(x) + x + 17$

- Left bound: -10^{12}
- Right bound: 10^{11}
- Output precision: 17 digits

```
Binary search with iteration limit 70
  Function x: 70 iterations
    f0(-8.4703294725430034e-10) = -8.4703294725430034e-10
    f0(8.4703294725430034e-11) = 8.4703294725430034e-11
  Function x + 412349128419.77615: 70 iterations
    f1(-412349128419.77618) = -6.103515625e-05
    f1(-412349128419.77612) = 0
  Function atan(x) + x + 17: 70 iterations
    f2(-15.493656816897174) = -5.5972293466766132e-10
    f2(-15.493656815965437) = 3.7588065993077718e-10
```

- Has a predictable running time
- But the number of iterations should be accurately adjusted
- For $[-10^{12}; 10^{11}]$ 50 is not enough but 70 is quite good

[1] Source code: `binsearch-3.cpp` at `https://github.com/mbuzdalov/i2cp-code`

Binary search in standard libraries

- C:
  ```
  bsearch(const void *key, const void *base, size_t num,
          size_t size, int (*cmp)(const void *, const void *)
  ```
  - Searches for element pointed by `key` in an array pointed by `base` of size `num`, assuming that elements have byte size `size` and array is sorted using comparator `cmp`
  - If key is not found, `NULL` is returned – not useful for certain searches
  - Calls `cmp` with key as first argument – can do binary search for a different type of object!

Binary search in standard libraries

- C: `bsearch(const void *key, const void *base, size_t num, size_t size, int (*cmp)(const void *, const void *)`
  - Searches for element pointed by `key` in an array pointed by `base` of size `num`, assuming that elements have byte size `size` and array is sorted using comparator `cmp`
  - If key is not found, `NULL` is returned – not useful for certain searches
  - Calls `cmp` with key as first argument – can do binary search for a different type of object!
- C++: functions for random-access iterators (including support for comparators)
  - `std::binary_search` – searches if an element exists, returns `true` or `false`
  - `std::{lower,upper}_bound` – returns an iterator to the lower/upper bound
  - `std::equal_range` – unites lower and upper bounds

Binary search in standard libraries

- C: `bsearch(const void *key, const void *base, size_t num, size_t size, int (*cmp)(const void *, const void *)`
  - Searches for element pointed by `key` in an array pointed by `base` of size `num`, assuming that elements have byte size `size` and array is sorted using comparator `cmp`
  - If key is not found, `NULL` is returned – not useful for certain searches
  - Calls `cmp` with key as first argument – can do binary search for a different type of object!
- C++: functions for random-access iterators (including support for comparators)
  - `std::binary_search` – searches if an element exists, returns `true` or `false`
  - `std::{lower,upper}_bound` – returns an iterator to the lower/upper bound
  - `std::equal_range` – unites lower and upper bounds
- Java: functions for arrays and collections
  - `java.util.Arrays.binarySearch` – searches for a key in an array of primitives by a natural ordering, or in array of objects (including comparator version). Returns index of an element if it is found, $-i-1$ if element is not found but could be inserted at index i. Has variations with `fromIndex` and `toIndex`.
  - `java.util.Collections.binarySearch` – same for collections