

# Binary Search

beOI Training



OLYMPIADE BELGE D'INFORMATIQUE  
BELGISCHE INFORMATICA-OLYMPIADE

April 15, 2016

# Table of Contents

Basic algorithm

Other applications

# Why do we need it?

## **Searching a sorted list:**

Given a sorted array  $A$ , find the index of an element  $v$  in this array.

# The algorithm

Look at the middle element and compare it with the number you're looking for.

Is it:

- ▶ The same → Congratulations!
- ▶ Smaller → It can only be after this element
- ▶ Bigger → It can only be before this element

Repeat!

This is a divide-and-conquer algorithm

## An example

$A =$ 

-5	-3	1	5	10	12	15	20	44	45
----	----	---	---	----	----	----	----	----	----

  
 $v = 12$

$low = 0$

$high = 9$

$mid = 4$

## An example

$A =$ 

-5	-3	1	5	10	12	15	20	44	45
----	----	---	---	----	----	----	----	----	----

  
 $v = 12$

$low = 0$

$high = 9$

$mid = 4$

## An example

$A =$ 

-5	-3	1	5	10	12	15	20	44	45
----	----	---	---	----	----	----	----	----	----

  
 $v = 12$

$low = 5$

$high = 9$

$mid = 7$

## An example

$$A = \begin{bmatrix} -5 & -3 & 1 & 5 & 10 & 12 & 15 & 20 & 44 & 45 \end{bmatrix}$$

$v = 12$

*low* = 5

*high* = 6

*mid* = 5





## What if it's not there?

We just stop when there's no more items to consider  
→  $low > high$

# The code

```
1 int binary_search(vector<int> A, int v) {
2     int low = 0;
3     int high = A.size() - 1;
4     while(low <= high) {
5         int mid = (low + high)/2;
6         if(A[mid]==v)
7             return mid;
8         if(A[mid] < v)
9             low = mid + 1;
10        if(A[mid] > v)
11            high = mid - 1;
12    }
13    return -1; // Not found
14 }
```

# The code

```
1 int binary_search(vector<int> A, int v) {  
2     int low = 0;  
3     int high = A.size() - 1;  
4     while(low <= high) {  
5         int mid = (low + high)/2;  
6         if(A[mid]==v)  
7             return mid;  
8         if(A[mid] < v)  
9             low = mid + 1;  
10        if(A[mid] > v)  
11            high = mid - 1;  
12    }  
13    return -1; // Not found  
14 }
```

Complexity?  $\mathcal{O}(\log(n))$

# The devil's in the details

Beware for off-by-one errors when changing boundaries!

How to handle duplicate values?

Watch out for overflow when computing indices.



# Table of Contents

Basic algorithm

Other applications

So...just searching?

Yep, that's it!

# So...just searching?

Yep, that's it!

Well, not only in an array.

# So...just searching?

Yep, that's it!

Well, not only in an array.

Only condition: static sorted *sequence*



# Bisection method

Find a hard root of a function.

Binary search on real numbers.

Example:  $x^3 - 4x - 9 = 0$

# The code: iterative

```
1 double f(double x) {  
2     return x*x*x - 4*x - 9;  
3 }  
4  
5 double bisection(double low, double high, double EPS) {  
6     while(true) { // depends on your problem  
7         double mid = (low + high)/2;  
8         double val = f(mid);  
9         if(abs(val) < EPS)  
10            return mid;  
11         if(val > 0)  
12            high = mid;  
13         else if(val < 0)  
14            low = mid;  
15     }  
16 }
```

# The code: recursive

```
1 double f(double x) {  
2     return x*x*x - 4*x - 9;  
3 }  
4  
5 double bisection(double low, double high, double EPS) {  
6     double mid = (low + high)/2;  
7     double val = f(mid);  
8     if(abs(val) < EPS)  
9         return mid;  
10    if(val > 0)  
11        return bisection(low, mid, EPS);  
12    else if(val < 0)  
13        return bisection(mid, high, EPS);  
14 }
```

# Binary search the answer: possibility 1

You don't know how to find the answer directly.  
BUT: you do know how to check how close you are.



## Binary search the answer: possibility 2

You don't know how to find the answer directly.  
BUT: you do know how to check if it's possible.



$A =$ 

0
---

0
---

0
---

1
---

1
---

1
---

1
---

# Table of Contents

Basic algorithm

Other applications

# Exercises

- ▶ Code a binary search!
- ▶ UVA 11057
- ▶ UVA 10567
- ▶ (UVA 957)
  
- ▶ UVA 10341
- ▶ UVA 11413
- ▶ UVA 11935
  
- ▶ UVA 12190 (very tedious!)