Introduction

Data Structures

Nick Smallbone

Course organisation

Lecturer: Nick Smallbone (me)

- nicsma@chalmers.se

Course assistants:

- Nachiappan Valliappan (nacval@chalmers.se)
- Adi Hrustic (hrustic@student.chalmers.se)
- Karin Wibergh (karin.wibergh@gmail.com)
- Oskar Lyrstrand (guslyros@student.gu.se)
- Daniel Willim (willim@student.chalmers.se)
- Robin Lilius-Lundmark (robin.lilius@gmail.com)

Student representatives:

- Adam Oliv, TKDAT (adamoli@student.chalmers.se)
- Hugo Simonsson, TKDAT (hugosi@student.chalmers.se)
- David Hedgren, TKDAT (davhedg@student.chalmers.se)
- Hugo Ölund, TKTEM (ohugo@student.chalmers.se)

Website: http://www.cse.chalmers.se/edu/course/DAT037

- Linked from student portal

Course organisation

Lectures twice a week

- Monday 8am (HC4)
- First three weeks: Friday 1pm (HA4)
- After that: Wednesday 8am (HB2)

Examination: exam + labs

- Grade determined by exam grade
- But need to pass labs too!

Labs

Three programming labs and one hand-in

- Do them in pairs (ask a course assistant for permission if you need to work alone)

Part of the course examination

- Copying of code strictly forbidden!
- Copying of hand-in answers forbidden too!
- But you are welcome to discuss ideas with each other
  - If you're not sure what's appropriate, ask

Come to the lab sessions if you need help

- Tuesdays 13-15 and 15-17, Thursdays 15-17
- No need to come to all lab sessions!
- Electronic queueing system Waglys (see website)

Exercises

A set of exercises will appear on the website each week

- Recommended to try them – they're there for your benefit!
- Answers will appear at some point

Two exercise sessions per week

- Monday 10am
- Thursday 8am

Where on earth is …?

Exercises and lab sessions are sprawled across many rooms

- EA, EC, EF, 3358, 3582, 2480
- ML1, ML3, ML16

Directions to all of these on the website!

Slack

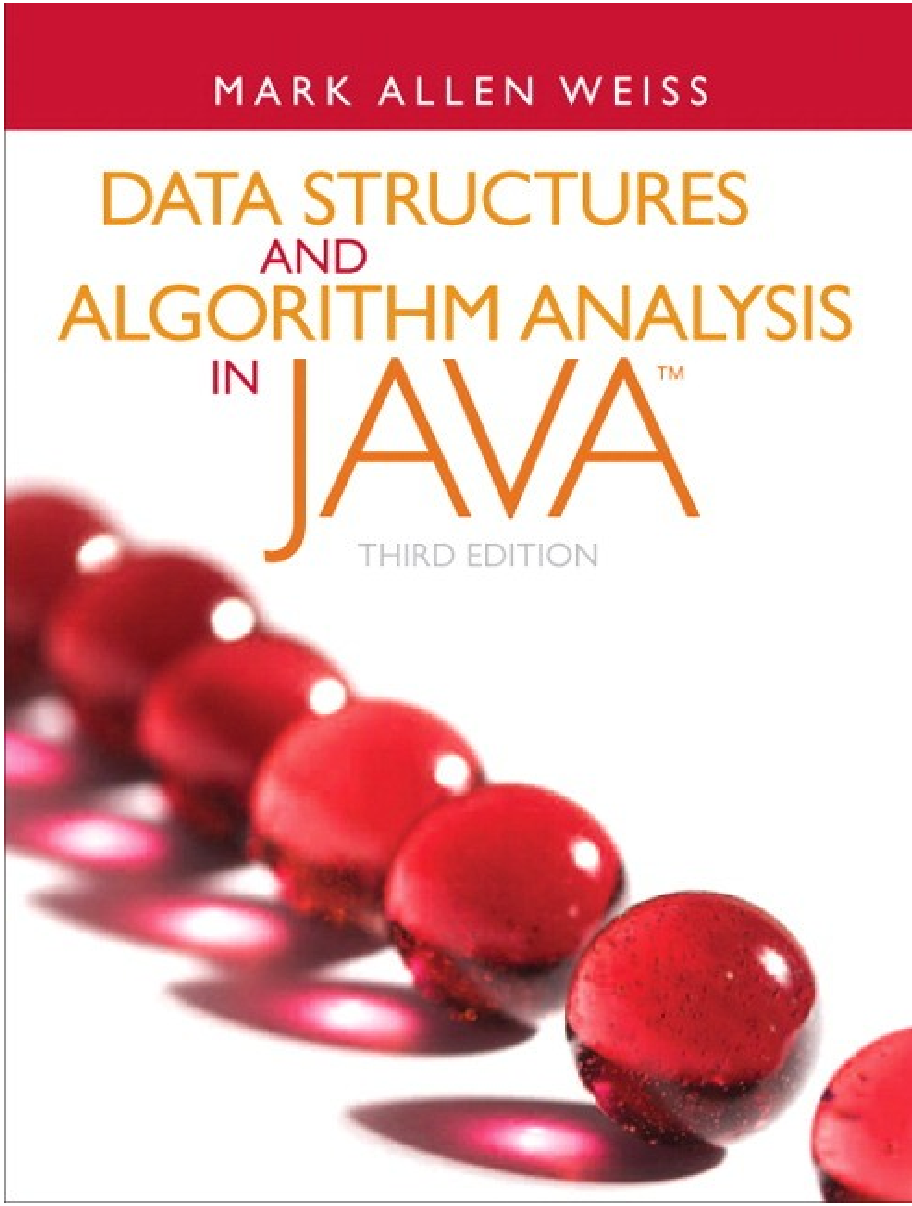
There is a course Slack instance (see website)

- Slack is a discussion group/chat room thingy
- There are rooms for discussing the exercises and labs, finding lab partners, and more
- You can also create your own rooms and have private conversations with people
- You can easily reach me and the course assistants
- You can ask questions about random things in the course

Please sign up!

The course book

- Mark Weiss:
  Data Structures and
  Algorithm Analysis in Java
- Matches the course fairly closely
- Lots of good exercises
- But you might be able to manage without it, especially if you're good at Java programming
- Also look at resources section of website



A FIRST EXAMPLE: READING A FILE

A simple problem

Here is a program that reads a file into a string and then prints it out twice:

```
String result = “”;
Character c = readChar();
while(c != null) {
    result += c;
    c = readChar();
}
System.out.print(result);
System.out.print(result);
```

A simple problem

Here is a program that reads a file into a string and then prints it out twice:

```
String result = “”;
Character c = readChar();
while(c != null) {
    result += c;
    c = readChar();
}
System.out.print(result);
System.out.print(result);
```

Takes *one hour* to read in *War and Peace*

The right way to solve it?

Use a StringBuilder instead

```
StringBuilder result = new StringBuilder();
Character c = readChar();
while(c != null) {
    result.append(c);
    c = readChar();
}
System.out.print(result);
System.out.print(result);
```

Takes *half a second*
to read in
War and Peace

...but: **why is there such a big difference?**

Behind the scenes

A string is basically an *array of characters*

- String s = “hello” ↔ char[] s = {'h','e','l','l','o'}

This little line of code...

```
  result = result + c;
```

is:

- Creating a new array one character longer than before
- Copying the original string into the array, *one character at a time*
- Storing the new character at the end
- Setting `result` to the new array

(See CopyNaive.java)

w	o	r	d	+ s

1. Create a new array

2. Copy the old array, one character at a time

w	o	r	d	

3. Add the new character to the end

w	o	r	d	**s**

Calculating the performance

When we append a character to a string of length *i*, it copies *i* characters

When we read a file of length *n*, the string changes length like so:
0 → 1 → 2 → … → n-1 → n

Hence number of characters copied is:
$0 + 1 + 2 + \ldots + (n-1) = n(n-1)/2$

(Here is a more back-of-the-envelope calculation:

- The string starts off at length 0, finishes at length *n*
- …so average length throughout is *n/2*
- Total characters copied is roughly
  number of characters appended × *average length of string*
- Which is $n \times n/2 = n^2/2$ characters copied)

For “War and Peace”, n = 3600000

so 1800000 × 3600000 = **6,480,000,000,000** characters copied!

No wonder it's slow!

Improving it (take 1)

It's a bit silly to copy the whole array every time we append a character

Idea: add some slack to the array

- Whenever the array gets full, make a new array that's (say) 100 characters bigger
- Then we can add another 99 characters before we need to grow the array again!
- In the implementation, we need a variable which remembers how much of the array is currently used

(See Copy100.java)

h	e	l	l	o		w	o	r	l

Add an element:

h	e	l	l	o		w	o	r	l
d									

Add an element:

h	e	l	l	o		w	o	r	l
d	**!**								

Not good enough

If *n = 100k* is a multiple of 100, then the array grows in size like so:
0 → 100 → 200 → ... → 100k

The total number of characters copied is
0+100+200+...+(100k-100)
= 100(0+1+...+(k-1))
= 100k(k-1)/2
= n(n-100) / 200

Intuitively, we need to grow the array **1/100th** as often, so the total number of characters copied is reduced by a factor of 100

Instead of copying **6,480,000,000,000** characters, we will copy only (roughly) **64,800,000,000!**

That's still way too many...

Not good enough

If *n* = *100k* is a multiple of 100, th like so:

$0 \rightarrow 100 \rightarrow 200 \rightarrow \ldots \rightarrow 100k$

The total number of characters c

$0+100+200+\ldots+(100k-100)$

$= 100(0+1+\ldots+(k-1))$

$= 100k(k-1)/2$

$= n(n-100) / 200$

Brute force idea:
let's grow the array by
1 million characters.

Let's not do this! Why not?

Intuitively, we need to grow the array **1/100th** as often, so the total number of characters copied is reduced by a factor of 100

Instead of copying **6,480,000,000,000** characters, we will copy only (roughly) **64,800,000,000!**

That's still way too many...

Improving it (take 2)

The problem is that growing the array gets more and more expensive as it gets bigger

Another idea: whenever the array gets full, **double** its size

That way, we need to grow the array *less and less often* as it gets bigger

Does this work?

Analysing it

Consider the case when we have just expanded the array

- This is the worst case for performance
- The string must contain $2^n + 1$ characters for some *n*

The array has grown like so:
$1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow \ldots \rightarrow 2^n \rightarrow 2^{n+1}$

The number of characters copied is then

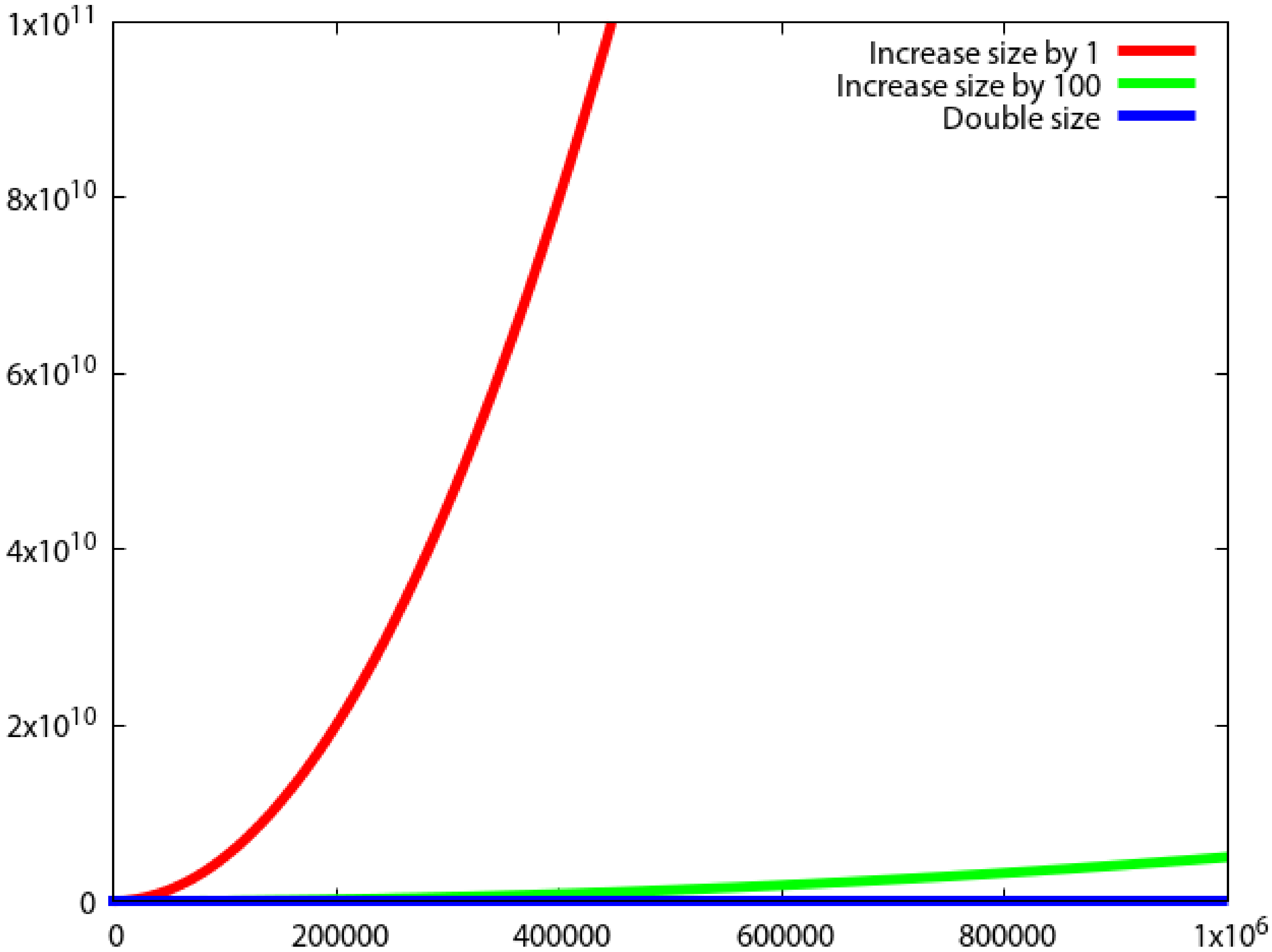
$1 + 2 + 4 + 8 + \ldots + 2^n$

$= 2^{n+1} - 1$

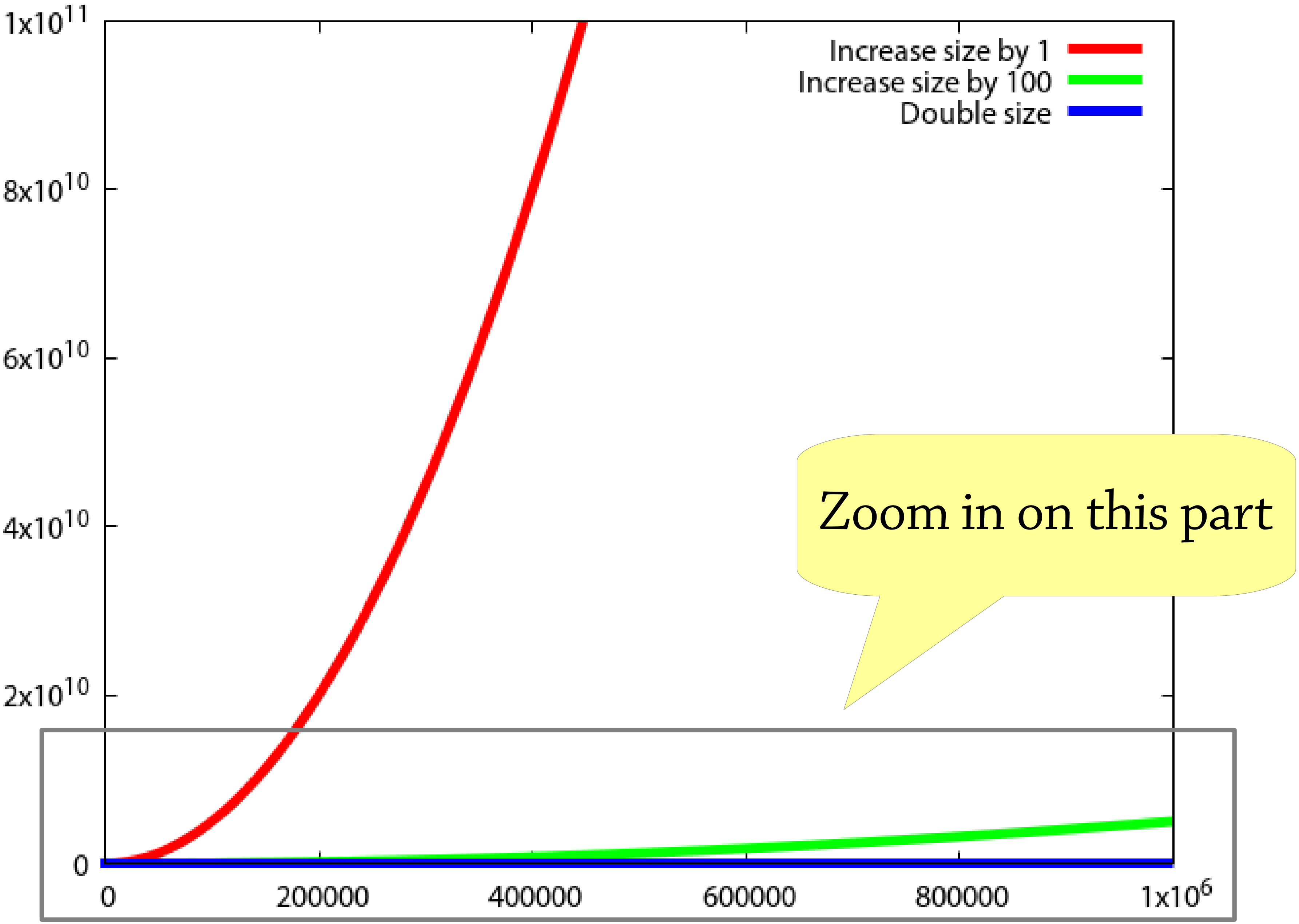
$< 2 \times (2^n + 1)$, i.e. twice the length of the string

For “War and Peace”, we copy **~7,200,000** characters. A million times less than the first version!

Performance – a graph

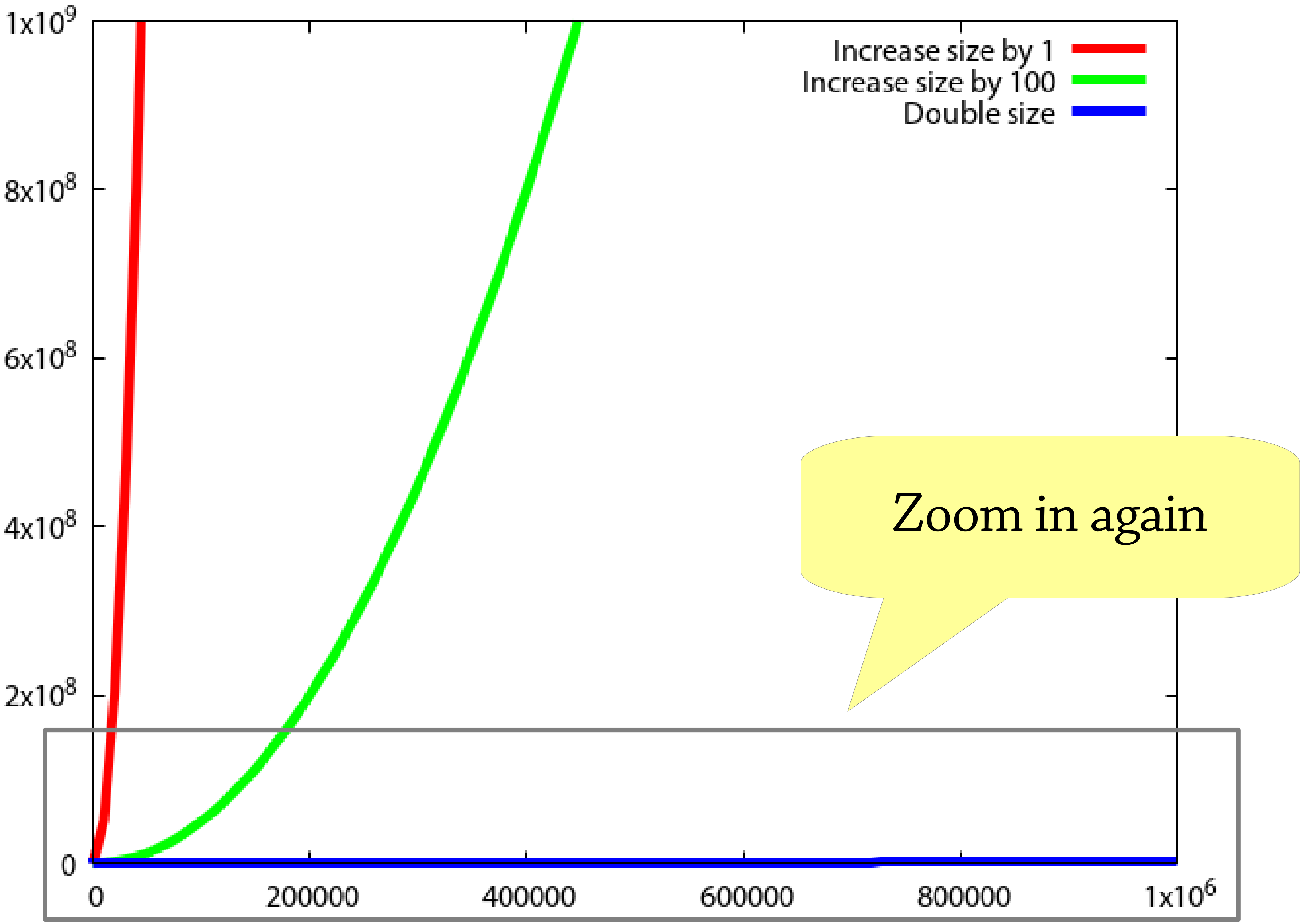


Performance – a graph



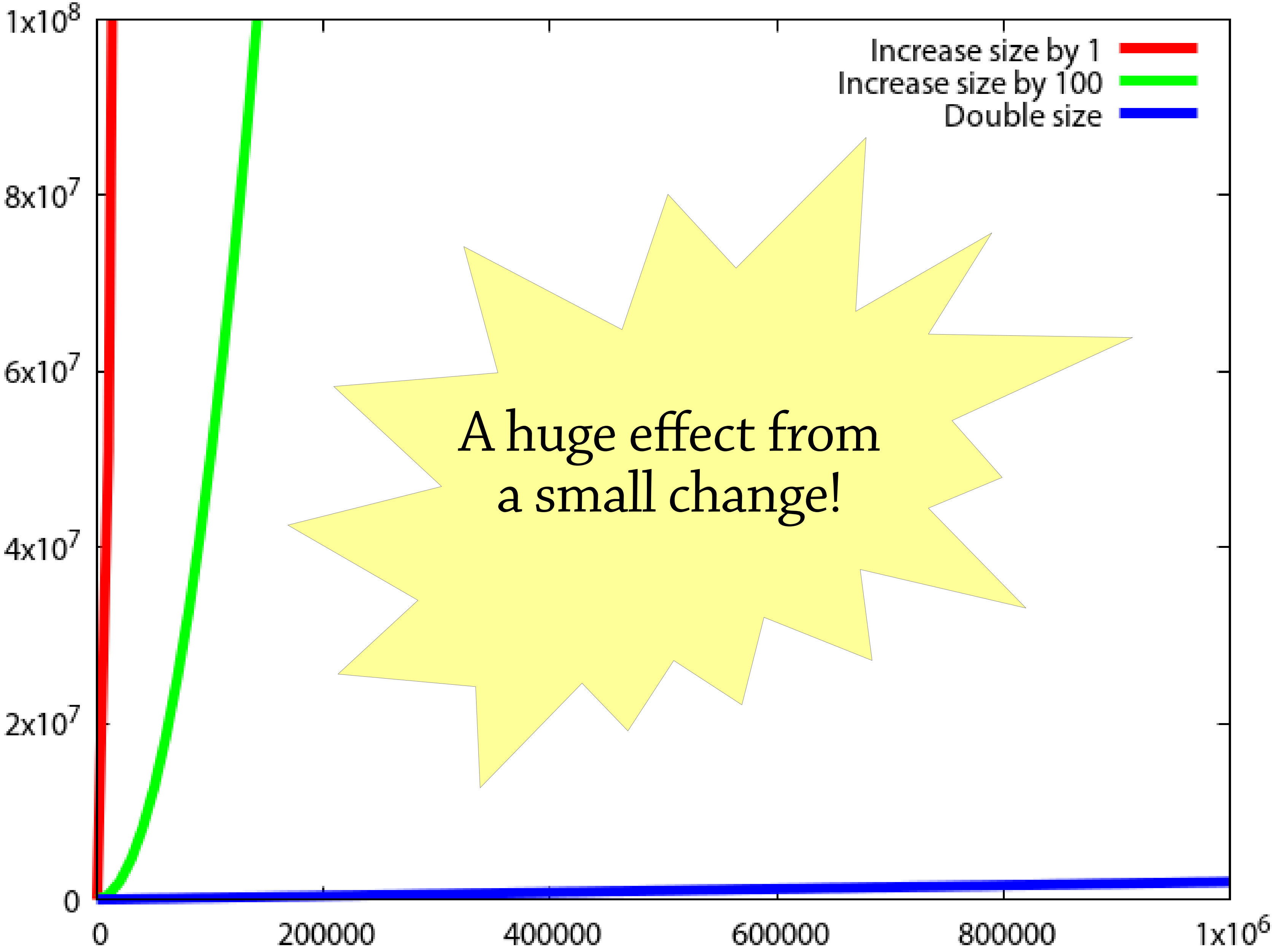
Zoom in on this part

Zoom in!



Zoom in again

Zoom in!



A huge effect from a small change!

Why does it work really?

The important property:

- After growing the array, the new array is no more than half full
- Let's say: the array's size is *2n*, but it contains *n* characters
- Before the next "expensive" append, which copies *2n* characters, there are *n* "cheap" appends with no copying => constant cost of 2 characters copied per step

Also works if we e.g. increase array size by 50% instead of doubling!

- Can trade off memory use vs execution speed

Dynamic arrays

A dynamic array is like an array, but can be resized – very useful data structure:

- `E get(int i);`
- `void set(int i, E e);`
- `void add(E e);`

Implementation is just as in our file-reading example:

- An array
- A variable storing the size of the used part of the array
- `add` copies the array when it gets full, but doubles the size of the array each time

Called `ArrayList` in Java

Exercise – also implement `void deleteLast();`

Back to strings and StringBuilder

String: array of characters

- Fixed size
- Immutable (can't modify once created)

StringBuilder: *dynamic* array of characters

- Can be resized and modified efficiently

Why can't the String class use a dynamic array?

DATA STRUCTURES

What is a data structure?

A design for organising the data in a program, so that particular operations can be performed efficiently

- An array: get and set
- A dynamic array (ArrayList): get, set, add new elements, ...
- A (hash) set: add, remove, check for membership
- A (hash) map: associate keys with values, look up keys
- Many hundreds more! Stacks, queues, priority queues, prefix trees, ...

Interface vs implementation

As a user, you are mostly interested in *what operations* the data structure supports, not how it works

Terminology:

- The set of operations is an *abstract data type (ADT)* (roughly corresponds to an interface in Java)
- The data structure *implements* the ADT
- Example: *map* is an ADT which can be implemented by a binary search tree, a skip list, a hash table, … (we will come across all these later)
- Why multiple implementations of the same ADT? Because they may support slightly different operations or have different performance characteristics

This course

How to design data structures

- Lectures, exercises, labs

How to reason about them

- Lectures, exercises, hand-in

How to use them and pick the right one

- Labs and exercises

Why learn this?

Why study how data structures work inside? Can't we just use them?

- Sometimes you need to *adapt* an existing data structure, which you can only do if you understand it
- The best way to learn how to *design your own* data structures is to study lots of existing ones

Plus, learning how things work is valuable in itself!

ANOTHER EXAMPLE: BINARY SEARCH

Searching

Suppose I give you an array, and ask you to find if a particular value is in it, say 4.

5	3	9	2	8	7	3	2	1	4

The only way is to look at each element in turn.

This is called *linear search*.

You might have to look at every element before you find the right one.

Searching

But what if the array is sorted?

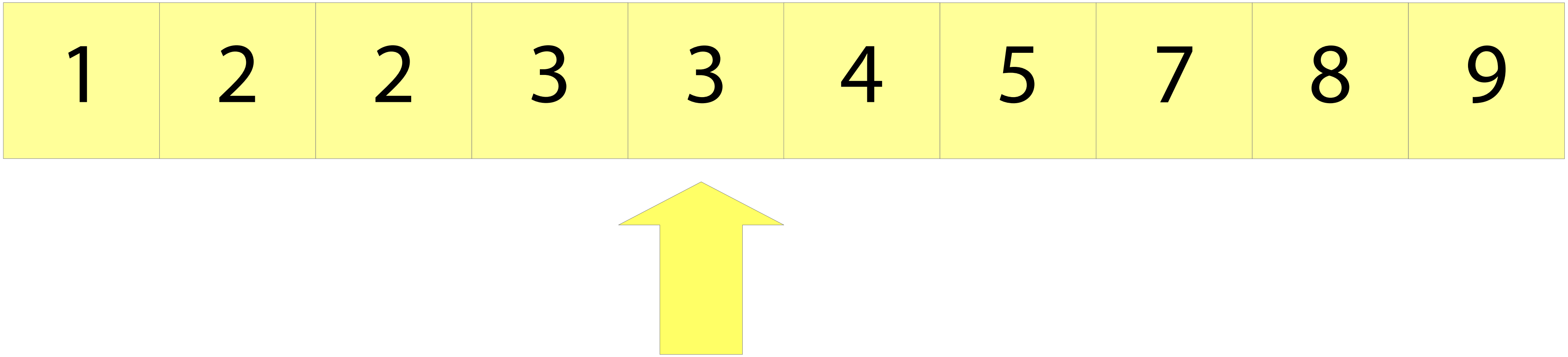
1	2	2	3	3	4	5	7	8	9

Then we can use *binary search*.

Binary search

Suppose we want to look for 4.

We start by looking at the element half way along the array, which happens to be 3.



1	2	2	3	3	4	5	7	8	9

Binary search

3 is less than 4.

Since the array is sorted, we know that 4 must come after 3.

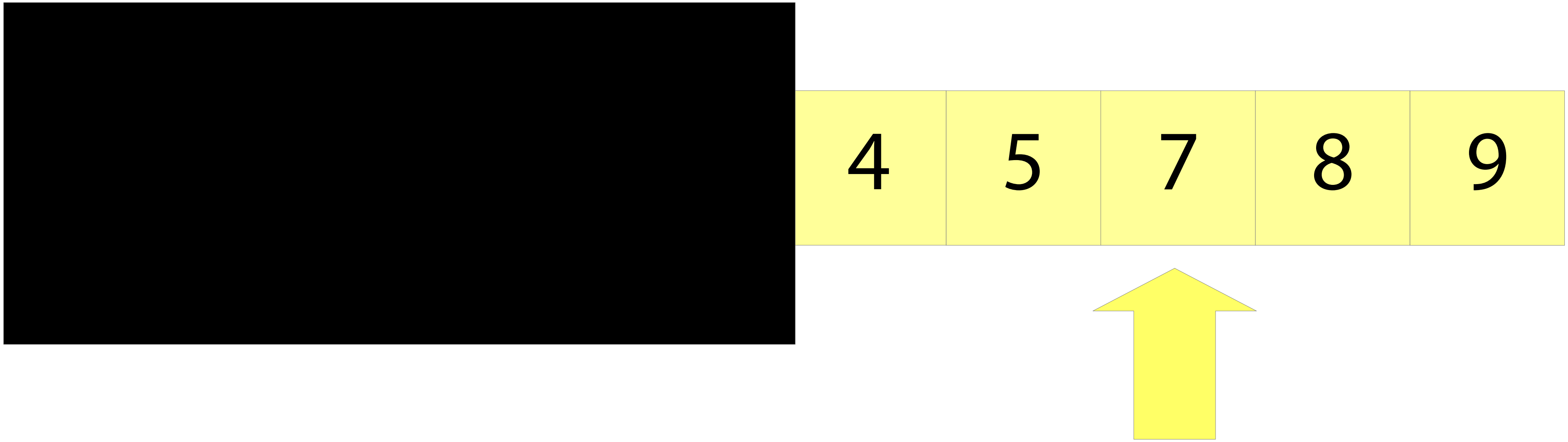
We can ignore everything before 3.

[redacted]	4	5	7	8	9

Binary search

Now we repeat the process.

We look at the element half way along what's left of the array. This happens to be 7.

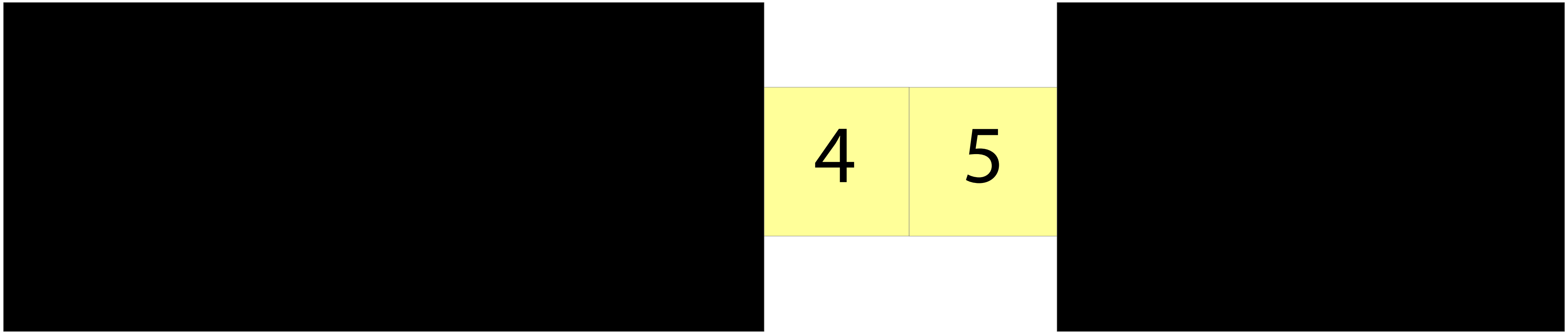


Binary search

7 is greater than 4.

Since the array is sorted, we know that 4 must come before 7.

We can ignore everything after 7.

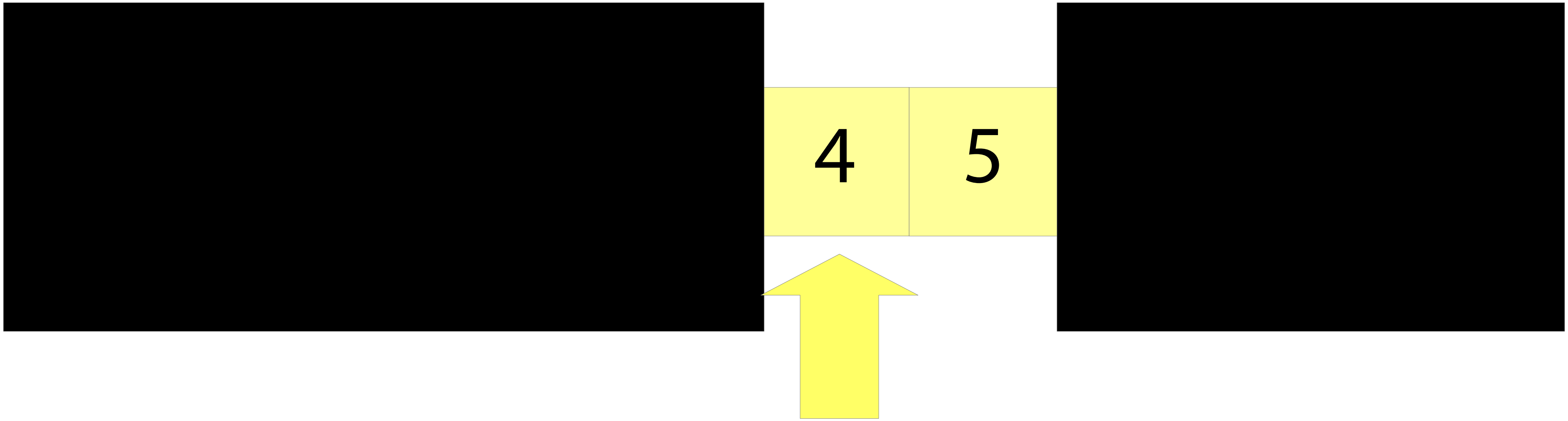


Binary search

We repeat the process.

We look half way along the array again.

We find 4!

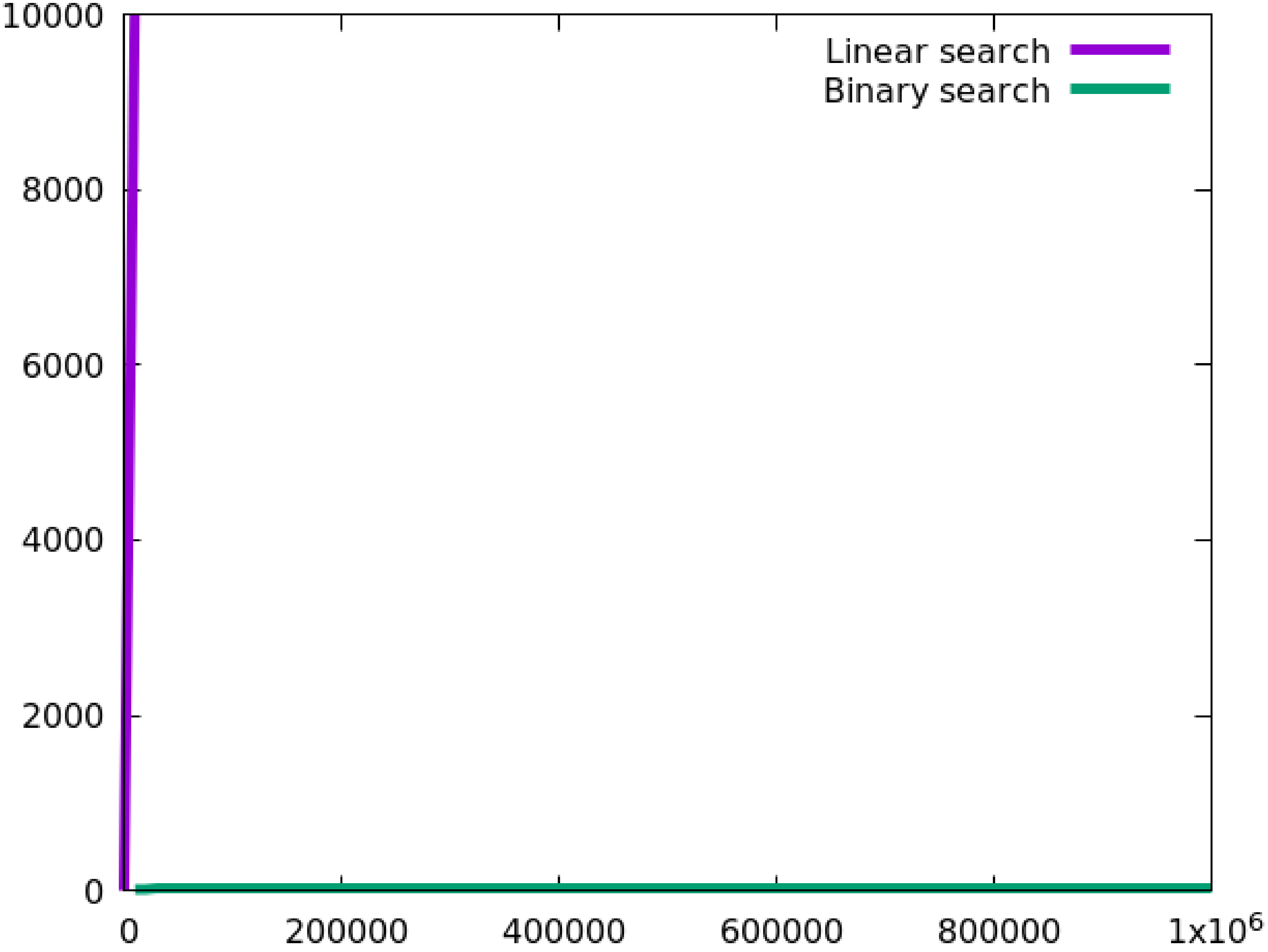


Performance of binary search

Binary search repeatedly *chops the array in half*

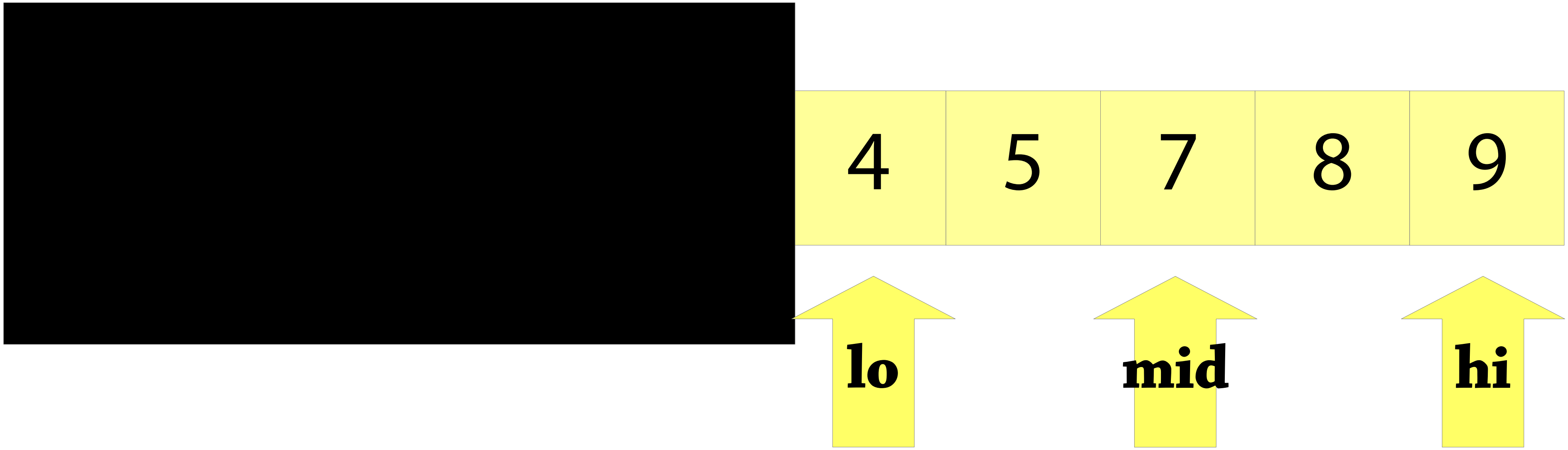
- If we double the size of the array, we need to look at one more array element
- With an array of size 2^n, after n tries, we are down to 1 element
- On an array of size n takes about $\mathbf{\log_2 n}$ tries!

On an array of a billion elements, need to search **30** elements, compared to a billion for linear search!


10000
8000
6000
4000
2000
0
0
200000
400000
600000
800000
1x10^6
Linear search
Binary search

Implementing binary search

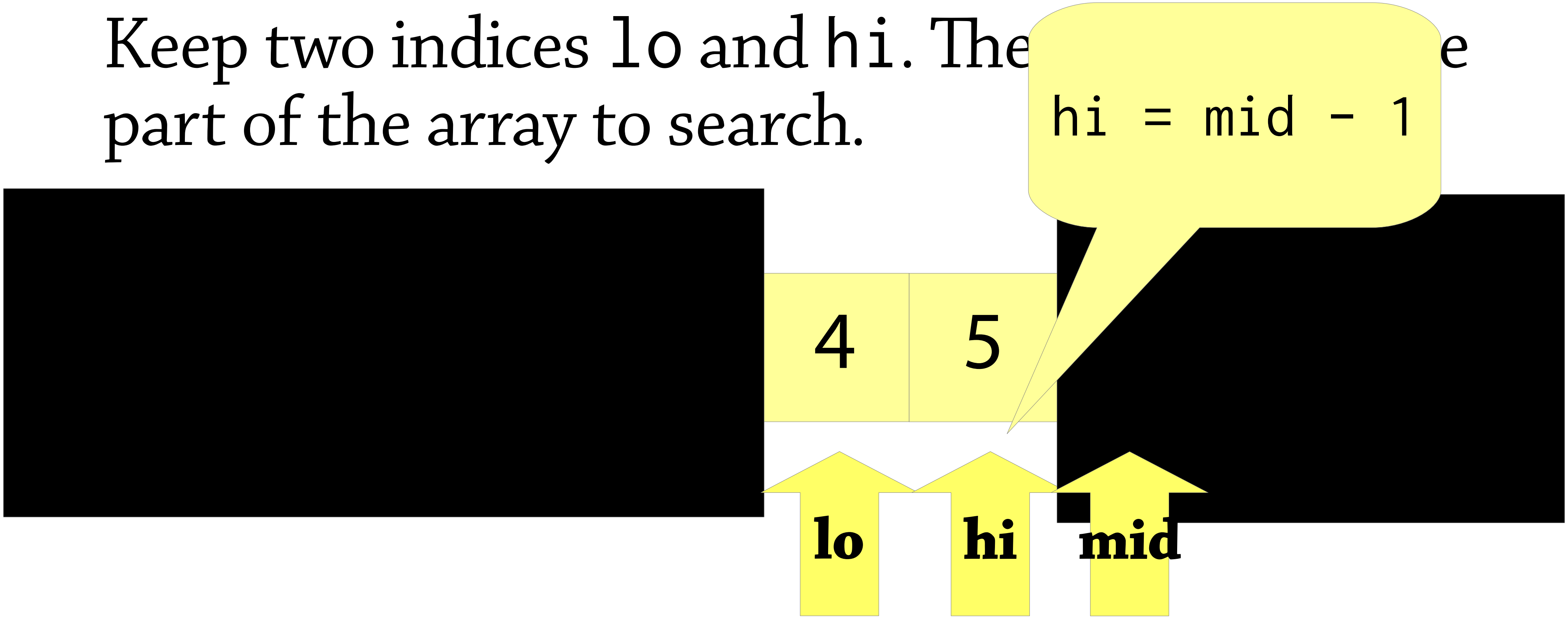
Keep two indices `lo` and `hi`. They represent the part of the array to search.



Let `mid = (lo + hi) / 2` and look at `a[mid]` – then either set `lo = mid+1`, or `hi = mid-1`, depending on the value of `a[mid]`

Implementing binary search

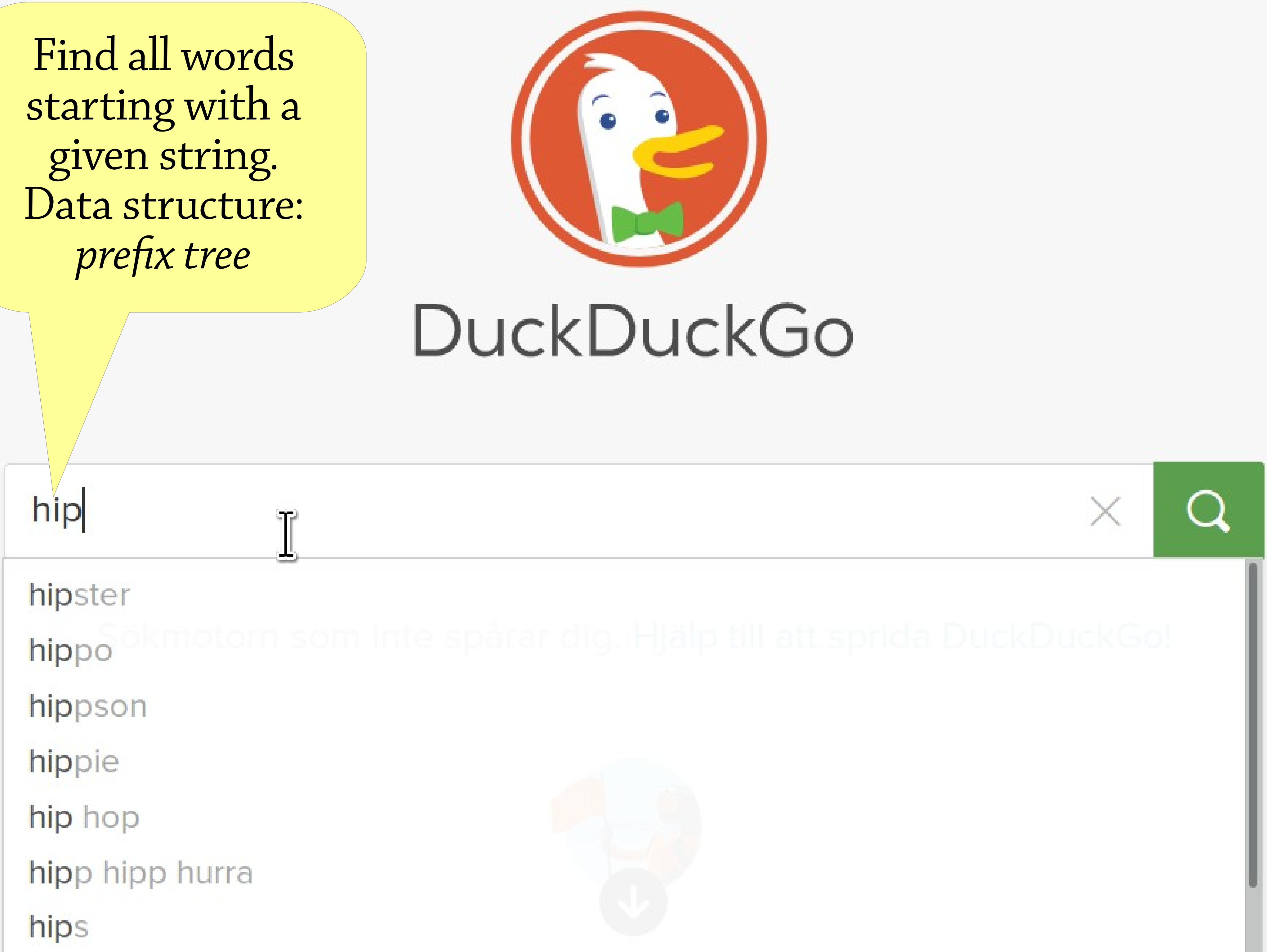
Keep two indices `lo` and `hi`. The e part of the array to search.


[redacted]
[redacted]

Let `mid = (lo + hi) / 2` and look at `a[mid]` – then either set `lo = mid+1`, or `hi = mid-1`, depending on the value of `a[mid]`

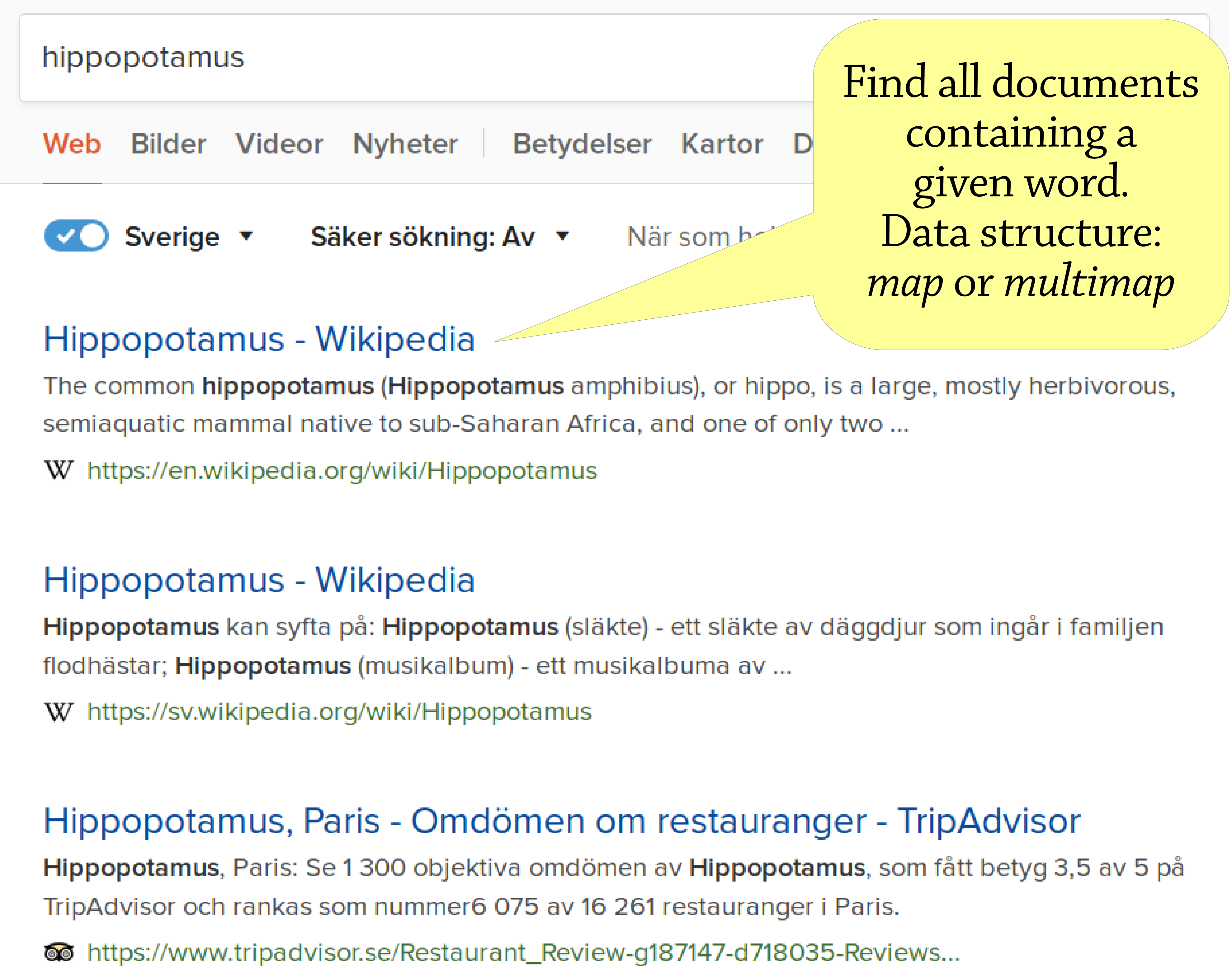
ON BRUTE-FORCE PROGRAMMING

Find all words starting with a given string.
Data structure:
prefix tree


DuckDuckGo
Sökmotorn som inte spårar dig. Hjälp till att sprida DuckDuckGo!

hip

- hipster
- hippo
- hippson
- hippie
- hip hop
- hipp hipp hurra
- hips



hippopotamus

Web Bilder Videor Nyheter | Betydelser Kartor D

✓ Sverige ▾ Säker sökning: Av ▾ När som h

Hippopotamus - Wikipedia

The common **hippopotamus** (**Hippopotamus** amphibius), or hippo, is a large, mostly herbivorous, semiaquatic mammal native to sub-Saharan Africa, and one of only two ...

W https://en.wikipedia.org/wiki/Hippopotamus

Hippopotamus - Wikipedia

Hippopotamus kan syfta på: **Hippopotamus** (släkte) - ett släkte av däggdjur som ingår i familjen flodhästar; **Hippopotamus** (musikalbum) - ett musikalbuma av ...

W https://sv.wikipedia.org/wiki/Hippopotamus

Hippopotamus, Paris - Omdömen om restauranger - TripAdvisor

Hippopotamus, Paris: Se 1 300 objektiva omdömen av **Hippopotamus**, som fått betyg 3,5 av 5 på TripAdvisor och rankas som nummer6 075 av 16 261 restauranger i Paris.

https://www.tripadvisor.se/Restaurant_Review-g187147-d718035-Reviews...

Find all documents containing a given word.
Data structure:
map or *multimap*

Plan best route from A to B.
Algorithm: *Dijkstra's algorithm*
Data structure: *graph* (and others)

Say no to brute-force programming!

Small instances of algorithmic problems can be solved by brute force

- To find all words starting with a given string, check all words one-by-one
- To find all documents containing a given word, check all documents one-by-one
- To find the best route from A to B, check all possible routes

But brute force doesn't scale to real problems!

Say no to brute-f g!

Instead of brute-force programming, use good data structures and good algorithms!

Small instances of a an be solved by brute force

- To find all words starting with a given string, ~~check all words one-by-one~~ **use a prefix tree**
- To find all documents containing a given word, ~~check all documents one-by-one~~ **use a map**
- To find the best route from A to B, ~~check all possible routes~~ **use Dijkstra's algorithm**

But brute force doesn't scale to real problems!

Big points

Data structures: say no to brute-force programming!

- Instead of relying on brute force, look for better ways to organise your data
- Using the right data structure can turn a program from impractically slow to lightning fast – speeding up a program by a constant factor is often insignificant compared to using a better algorithm
- Using the right data structure also makes your program simpler!

Most data structures are based on some simple idea

- e.g., let's increase the amount of slack space as the array gets bigger
- e.g., let's sort the data so that it's quicker to search

We can analyse a program's performance mathematically

Data structure (class) vs abstract data type (interface)

Lab 1: dynamic arrays and binary search!

Reading: Weiss chapter 1 (skip 1.4), 3.1-3.4 skipping the sections about linked lists

If you are going to the exercises now, they are in the EDIT building, rooms EA and EC (you will not be able to do the complexity ones yet)