Data Structures

Introduction

A data structure
is any way of organising the data in your program

Data structures

Any data structure also comes with certain *operations* that it supports

- Arrays: a[i] (get), a[i] = x (set)
- Haskell lists: cons, head, tail
- Maps: get, set, insert, delete, …
- Many many more

Why do we need data structures?

- You can program anything without using fancy data structures
- ...but it might run *very slowly*
- You need to know about data structures to write efficient programs
- Might be the difference between a program finishing instantly and taking 1000 years!

This course

- *How to design* data structures
  - Lectures and exercises
- *How to reason* about their performance
  - Lectures and exercises
- *How to use them* in programming
  - Labs and exercises

This course

- Lecturer: Nick Smallbone (me)
  - nicsma@chalmers.se, room 5463
- Assistants: Staffan Björnesjö and Bartolomeus Jankowski
- Google group – please sign up!
- All info on the website!

Registration

“This semester you have to register online in LPW at the Student portal. The registration is obligatory in order to attend the course.

Note that you need to register yourself on the course the same day as the first lecture otherwise you will lose your place.”

Organisation

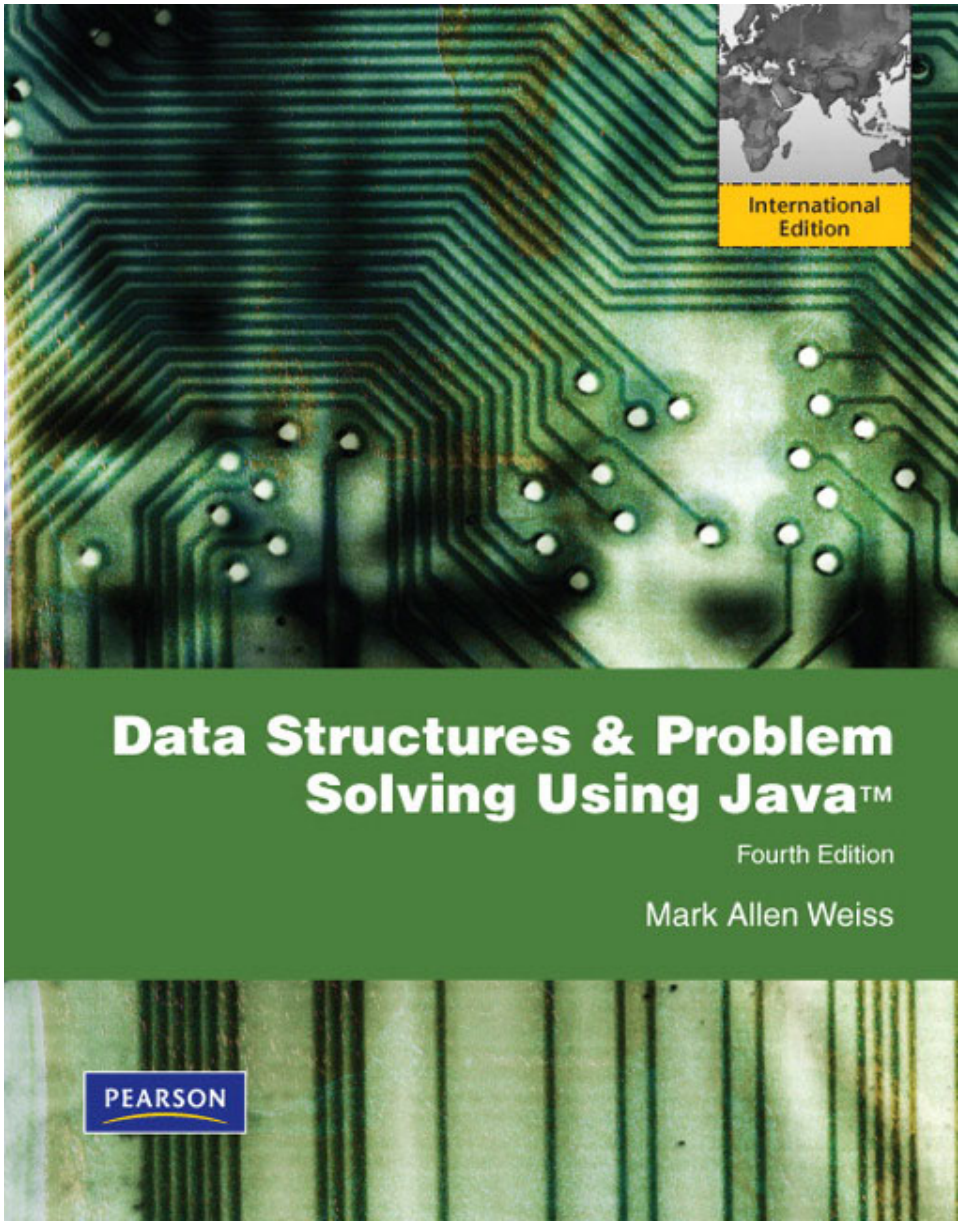
- Lectures twice a week: Wednesday 13-15, Friday 13-15 (all in EL41)
- Labs/exercises three times a week: Tuesday 13-15, Tuesday 15-17, Friday 10-12 (all in 3354+3358)
- Exam at end of course (30th May 14-18)

Labs

- Four labs and one hand-in
  - First lab: phone book. Deadline end of week 2
- Do them in pairs (ask me if you want to do them alone)
- Must pass labs and exam to pass the course

Book

- Mark Weiss: Data Structures and Problem Solving Using Java, 4th ed.
- Order from e.g. Adlibris (650 kr)



Reading a file

```
String result = “”;
String line = in.readLine();
while(line != null) {
    result += line + "\n";
    Line = in.readLine();
}
```

- On a big file, takes an extremely long time
- Appending many strings together is slow

Reading a file, take 2

```
StringBuilder result = new StringBuilder();
String line = in.readLine();
while(line != null) {
    result.append(line + “\n”);
    Line = in.readLine();
}
```

- Runs quickly even on big files
- StringBuilder is a *data structure* that supports efficiently appending strings

Arrays

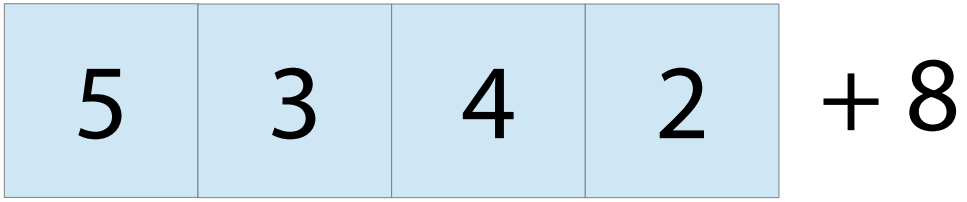
- The most basic data structure
- a[i]: read index i of array a
- a[i] = x: write index i of array a
- new int[10]: create a new array
- The size of an array is fixed once it's created

Dynamic arrays

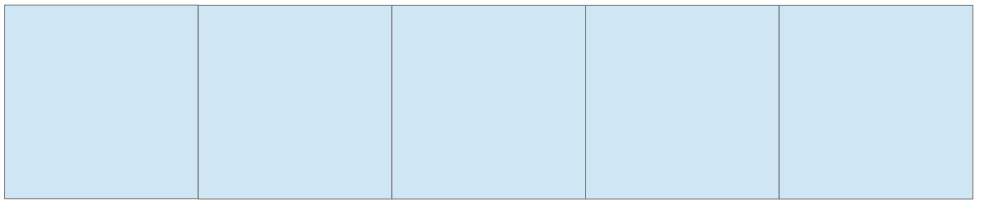
- Sometimes you don't know how big an array should be in advance
- A *dynamic array* provides an operation to *add* an element to the end of an array (changing its size)

One attempt

To add an element to `array`, create a new array with one more element and copy everything there

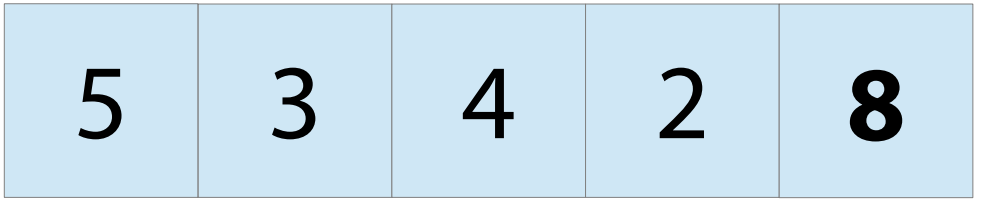


1. Make a new array



2. Copy the old array there

3. Add the new element



Implementation

```
Object[] array = {};

void add(Object x) {
  Object[] newArray = new Object[array.length + 1];
  for (int i = 0; i < array.length; i++)
    newArray[i] = array[i];
  newArray[array.length] = x;
  array = newArray;
}
```

Performance

- Each time you add an element to `array`, you copy `array.length` elements
- Start with an empty array, and add n elements:
  - First add copies nothing, second add copies one element, third add copies two elements, and so on
  - $1 + 2 + \ldots + (n-1) = \mathbf{n(n-1) / 2}$

n(n-1) / 2

- Suppose copying one element takes one microsecond.
- n = 10000: 50 million elements copied! 50 seconds.
- n = one million: 500 **billion** elements copied!
  Nearly a week!

Attempt two

When copying the array, increase its size by 10 elements instead of one, and only copy everything when the array gets full

5	3	4	2	8	7	3	2	1	9

Add an element:

5	3	4	2	8	7	3	2	1	9
3									

Add an element:

5	3	4	2	8	7	3	2	1	9
3	**2**								

Implementation

```
Object[] array = {};
int size = 0;

void add(Object x) {
  if (array.length == size) {
    Object[] newArray = new Object[array.length + 10];
    for (int i = 0; i < array.length; i++)
      newArray[i] = array[i];
    array = newArray;
  }
  array[size] = x;
  size++;
}
```

Performance

- Each time you add **10 elements** to `array`, you copy `array.length` elements
- Start with an empty array, and add **10n** elements:
  - First ten adds copy no elements, second ten adds copy ten elements, third ten adds copy twenty elements, and so on
  - $10 + 20 + \ldots + 10(n\text{-}1) = 10(1 + 2 + \ldots + n\text{-}1) = 10n(n\text{-}1) / 2 =$ **$5n(n\text{-}1)$**

Performance

- With **10n** adds, we copy **5n(n-1)** elements.
- To get the number of copies for **n** adds, substitute n/10 for n, to get 5(n/10)(n/10-1), or about **n(n-1)/20**.
- Can handle 10 times as large inputs – but that's all!

Attempt three

Whenever the array gets full, instead of adding 10 elements, **double its size**

```
Object[] array = {null};
int size = 0;

void add(Object x) {
  if (array.length == size) {
    Object[] newArray = new Object[array.length * 2];
    for (int i = 0; i < array.length; i++)
      newArray[i] = array[i];
    array = newArray;
  }
  array[size] = x;
  size++;
}
```

Performance

- Each time you **double the size** of `array`, you copy `array.length` elements
- Start with an empty array, and add $\mathbf{2^n+1}$ elements:
  - After 1 add array is full; second add copies 1 element
  - After 2 adds array is full; third add copies 2 elements
  - After 4 adds array is full; fifth add copies 4 elements
  - …
  - After 2^n adds array is full; 2^n+1th add copies 2^n elements
  - $1 + 2 + 4 + 8 + \ldots + 2^n = 2^{n+1} - 1$ copies

Performance

- For $\mathbf{2^n+1}$ adds, we copy $2^{n+1} - 1$ elements. This is the **worst case**.
- $2^{n+1} - 1 = 2 \times (2^n+1) - 3$
- For **n** adds, we will copy at most 2n – 3 elements

$1 + 2 + 4 + 8 + \ldots + 2^{n-1} = 2^n - 1$? Why?

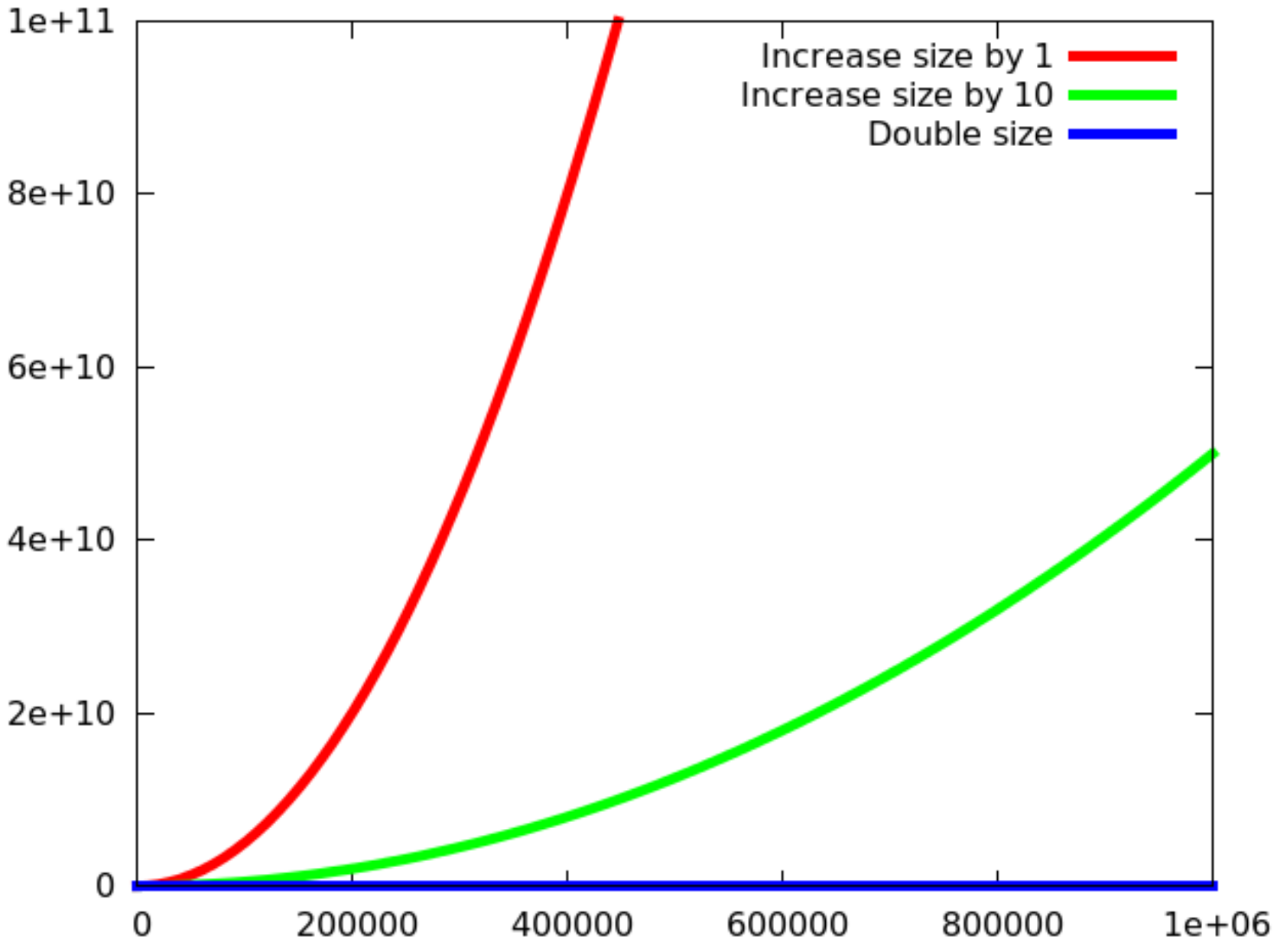
Well, let $S = 1 + 2 + 4 + 8 + \ldots + 2^{n-1}$.

So $2S = 2 + 4 + 8 + \ldots + 2^n$.

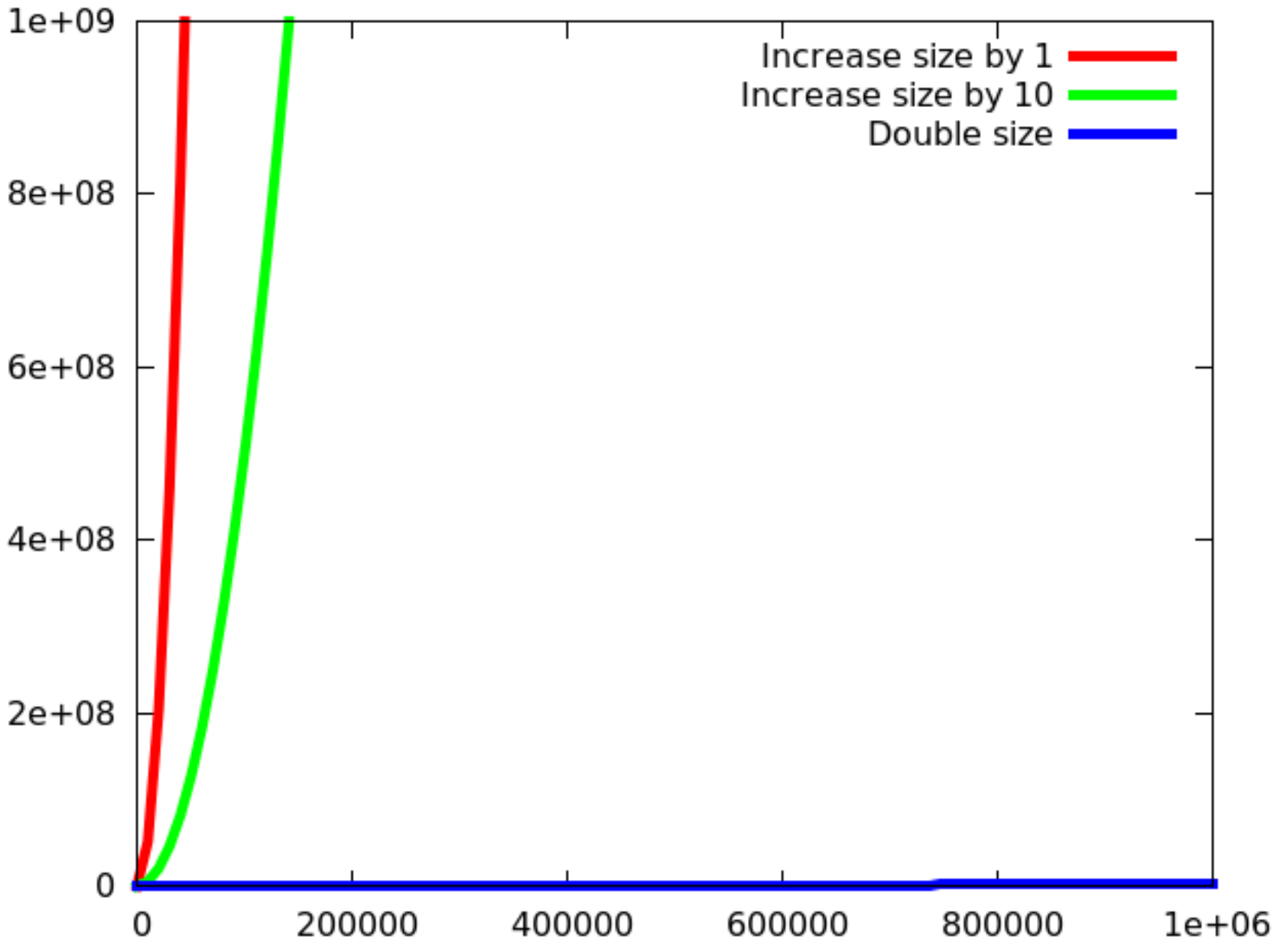
Now, calculate $2S - S$. The terms 2, 4, 8, …, 2^{n-1} cancel out, leaving $2^n - 1$.

But $2S - S = S$. So $S = 2^n - 1$.

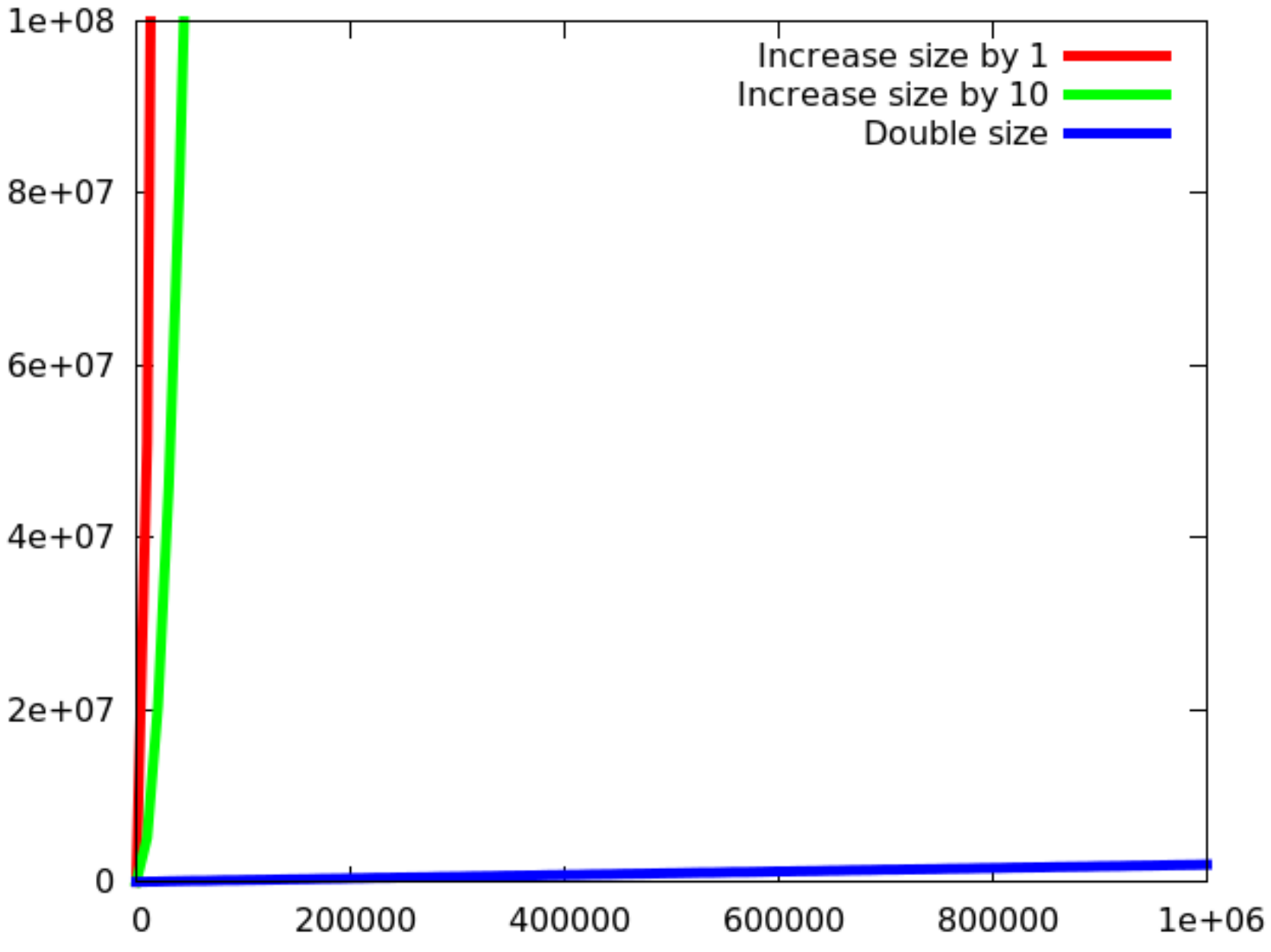
Performance – a graph



Zoom in!


1e+09
8e+08
6e+08
4e+08
2e+08
0
0
200000
400000
600000
800000
1e+06
Increase size by 1
Increase size by 10
Double size

Zoom in!


1e+08
8e+07
6e+07
4e+07
2e+07
0
0
200000
400000
600000
800000
1e+06
Increase size by 1
Increase size by 10
Double size

The right use of data structures can get you such speedups in *your* programs!

ArrayList<E>

```
class ArrayList<E> {
    public ArrayList();
    public E get(int i);
    public void set(int i, E e);
    public boolean add(E e);
    public int size();
    // plus much more
}
```

Back to reading a file

Remember our slow program:

```
String result = “”;
String line = in.readLine();
while(line != null) {
    result = result + line + "\n";
    Line = in.readLine();
}
```

The culprit is the line

```
result = result + line + "\n";
```

It's just like our very slow dynamic array!

Back to reading a file

Remember our *fast* program:

```
StringBuffer result = new StringBuffer();
String line = in.readLine();
while(line != null) {
    result.append(line + “\n”);
    Line = in.readLine();
}
```

`StringBuffer` uses a dynamic array!

(A toy implementation of) StringBuffer

```
class StringBuffer {
  ArrayList<Character> list =
    new ArrayList<Character>;

  void append(String s) {
    for (int i = 0; i < s.length; i++)
      list.add(s.charAt(i));
  }

  String toString() {
    char[] string = new char[list.size()];
    for (int i = 0; i < list.size(); i++)
      string[i] = list.get(i);
    return new String(string);
  }
}
```

Why can't String use StringBuffer?

Couldn't String also use a dynamic array when you append to it? Why does it need to be slow?

Answer: String is **immutable** – you can't change a string after you've created it. s1 + s2 returns a **new string**, so must allocate a **new array**.

Tradeoffs, tradeoffs

- String and StringBuffer have different strengths and weaknesses:
  - StringBuffer has efficient append, String does not
  - But String is immutable. We can't use StringBuffer instead of String everywhere because when someone passes you a string you want to know that it won't change
- All data structures make different tradeoffs

Next lecture

- How to reason about the performance of data structures and algorithms
- How to argue that an algorithm is correct
- Lab session on Friday! (You can already start.)