DIT181: Data Structures and Algorithms

Introduction and Arrays, Dynamic Arrays

Gül Calikli
Email: calikli@chalmers.se

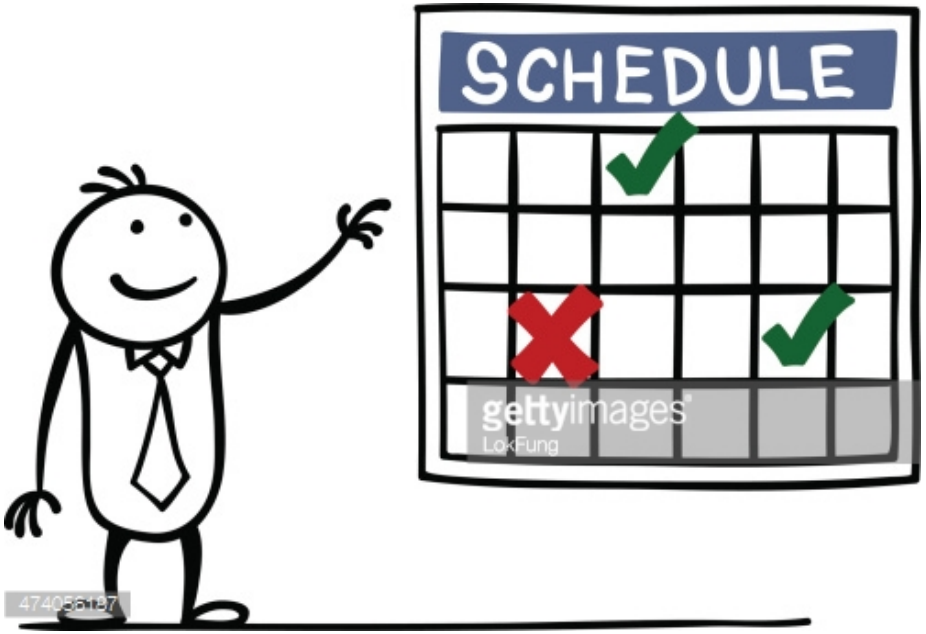
Introduction: About this course …

- **DIT181 TEAM**

- **Lecturers:** Gül Calikli (me) & Michal Palka
  - calikli@chalmers.se; michal.palka@chalmers.se
  - room 476, Jupiter Building, 4th floor
- **Teaching Assistants:**
  - Margit Saal (gussaalma@student.gu.se)
  - Snezhina Racheva (gusracsn@student.gu.se)
  - Yue Kang (yuek@chalmers.se)
  - Yushu Yu (yushu.yu@chalmers.se)

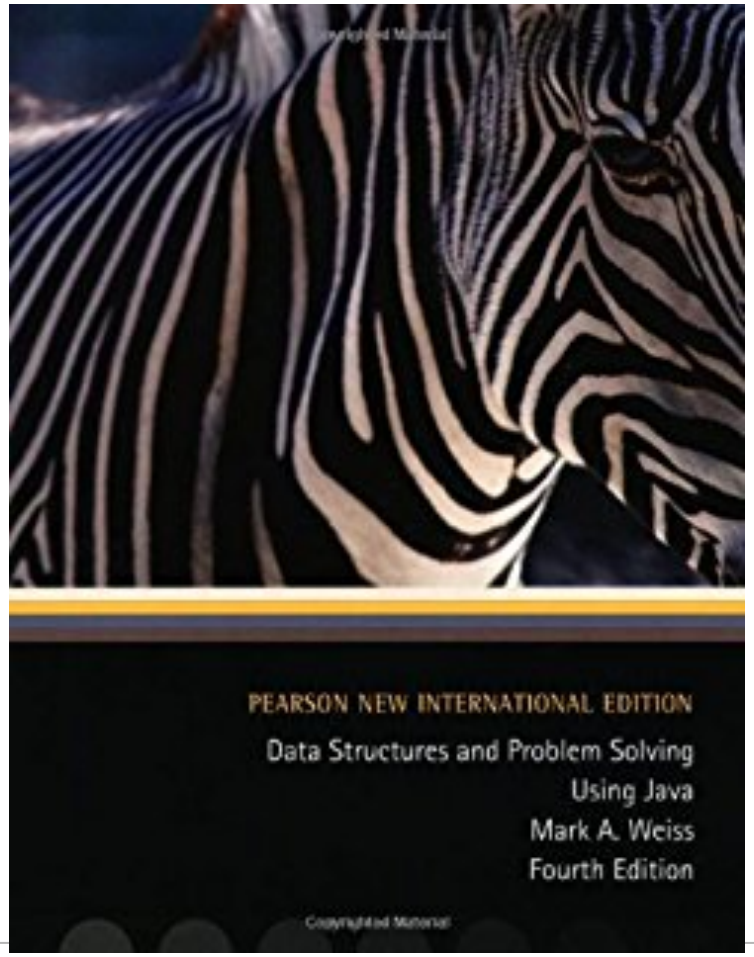
Introduction: About this course …

- **LECTURE TIMES & PLACE**
- Tuesdays, 10:15 am - 12:00 pm
  - Location: Alfons
- Thursdays, 10:15 am - 12:00 pm
  - Location: Alfons



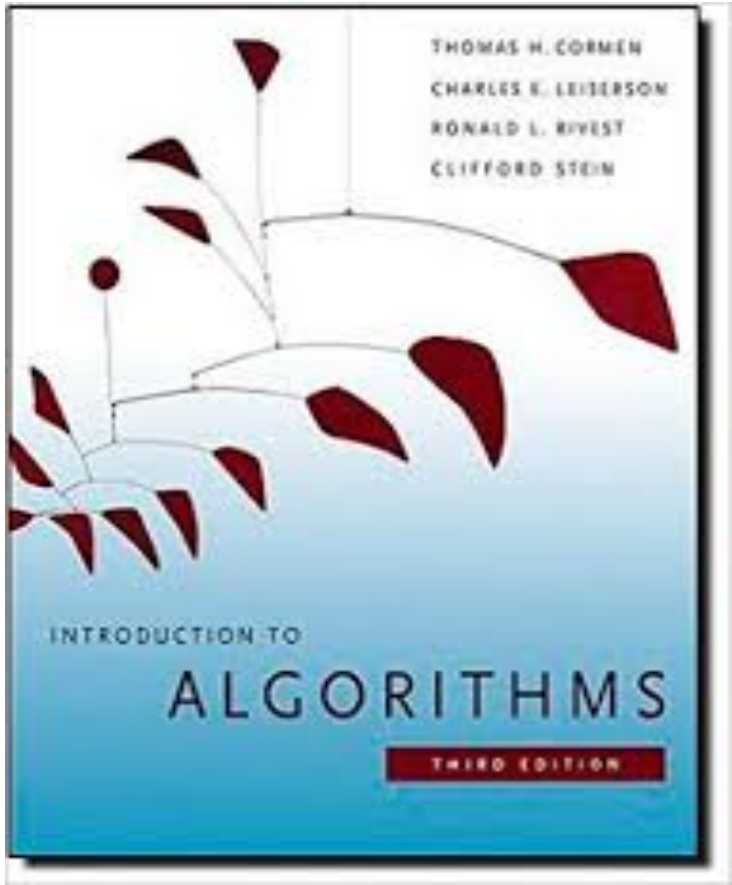
Introduction: About this course …

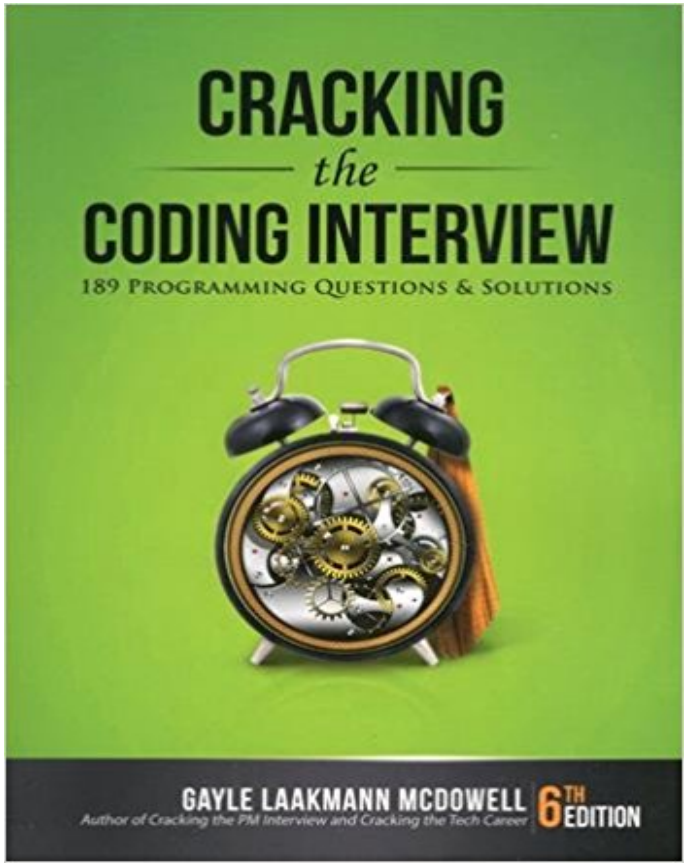
- **COURSE TEXTBOOK**
- **Mark A. Weiss**, "**Data Structures and Problem Solving**", Pearson International Edition, Fourth Edition, ISBN-10: 129202576X; ISBN-13: 978-1292025766



Introduction: About this course …

- **Recommended Reading List:**





Introduction: About this course …

- **GRADING**

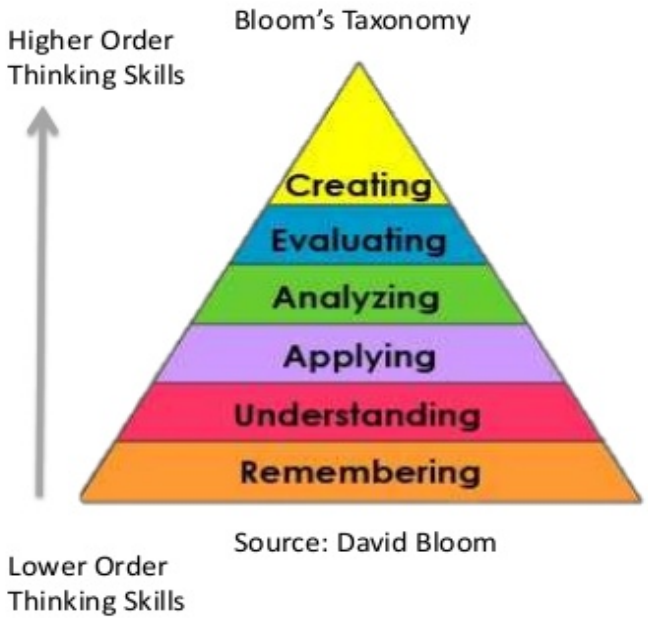
Sub-courses

1. Written exam, 4.5 higher education credits
   Grading scale: Pass with Distinction (VG), Pass (G) and Fail (U)
2. Assignments, 3 higher education credits
   Grading scale: Pass (G) and Fail (U)
   The Assignments will consist of the following:
   - In-Class exercises will make up 10% of the grade for "Assignments".
   - 3 assignments consisting of written (theory and practice related) questions and short programming exercises. (Each assignment makes up 15% of the grade for the sub-course "Assignments", hence making up 45% of the grade for "Assignments" in total.)
   - A programming project together with a written report. (Programming project makes up 45% of the grade for the sub-course "Assignments", 35% being for the source code of the project and 10% being for documentation.)

Introduction: About this course …

- **LECTURE FORMAT**
  - We will try to employ "Active Learning" → Lecturing Accompanies by In-Class Exercises

WHAT IS ACTIVE LEARNING?



"Students that are engaged in more activities than just listening. They are involved in dialog, debate, writing, and problem solving as well as higher order thinking, e.g. analysis, synthesis, evaluation, and creating" - Bonwell and Eison, 1991

- Let’s start our first lecture…

A Simple Coding Problem

- **In-Class Exercise 1.1**:
- Write a program that reads a file, and then outputs exactly what it has read.

A Simple Coding Problem

- **In-Class Exercise 1.1**:
- Write a program that reads a file, and then outputs exactly what it has read.

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

One way of doing it!

A Simple Coding Problem

- **In-Class Exercise 1.2**:
- Change the following program so that it reads a file, then prints out total number of characters in that file and then prints out exactly what it has read from that file?

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

A Simple Coding Problem

- **In-Class Exercise 1.2**:
- Change the following program so that it reads a file, then prints out total number of characters in that file and then prints out exactly what it has read from that file?

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

IDEA: read the file into a string.

A Simple Coding Problem

- **In-Class Exercise 1.2**:
- Change the following program so that it reads a file, then prints out total number of characters in that file and then prints out exactly what it has read from that file?

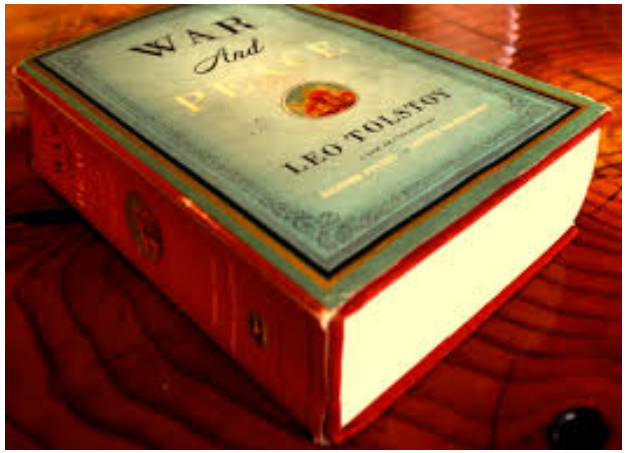
```
String result = "";
int num_chars = 0;
Character c = readChar();
while(c != null) {
        result += c;
        num_characters += 1;
        System.out.print(c);
        c = readChar();
}
System.out.println(num_chars);
System.out.print(result);
```

IDEA: read the file into a string.

A Simple Coding Problem

- If you execute the following code to read ”War and Peace” (novel by Tolstoy) from a text file, you’ll see that it is **very slow.**

```
String result = “”;
int num_chars = 0;
Character c = readChar();
while(c != null) {
        result += c;
        num_characters += 1;
        c = readChar();
}
System.out.println(num_chars);
System.out.print(result);
```



The right way to solve the problem…

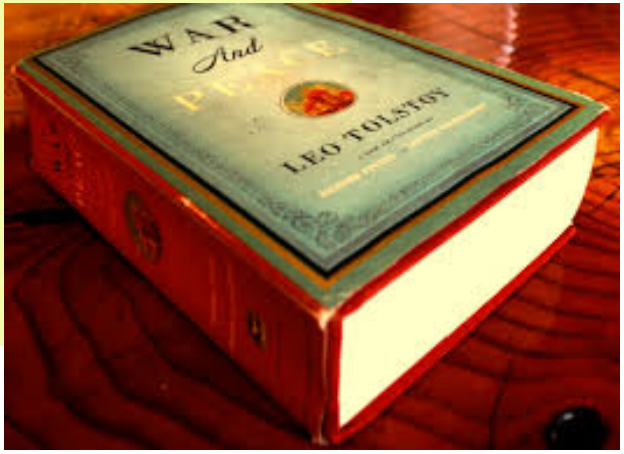
- Use a **`StringBuilder()`** instead of **`String`**

```
StringBuilder result = new StringBuilder();
int num_chars = 0;
Character c = readChar();
while(c != null) {
        result.append(c);
        num_characters += 1;
        c = readChar();
}
System.out.println(num_chars);
System.out.print(result);
```

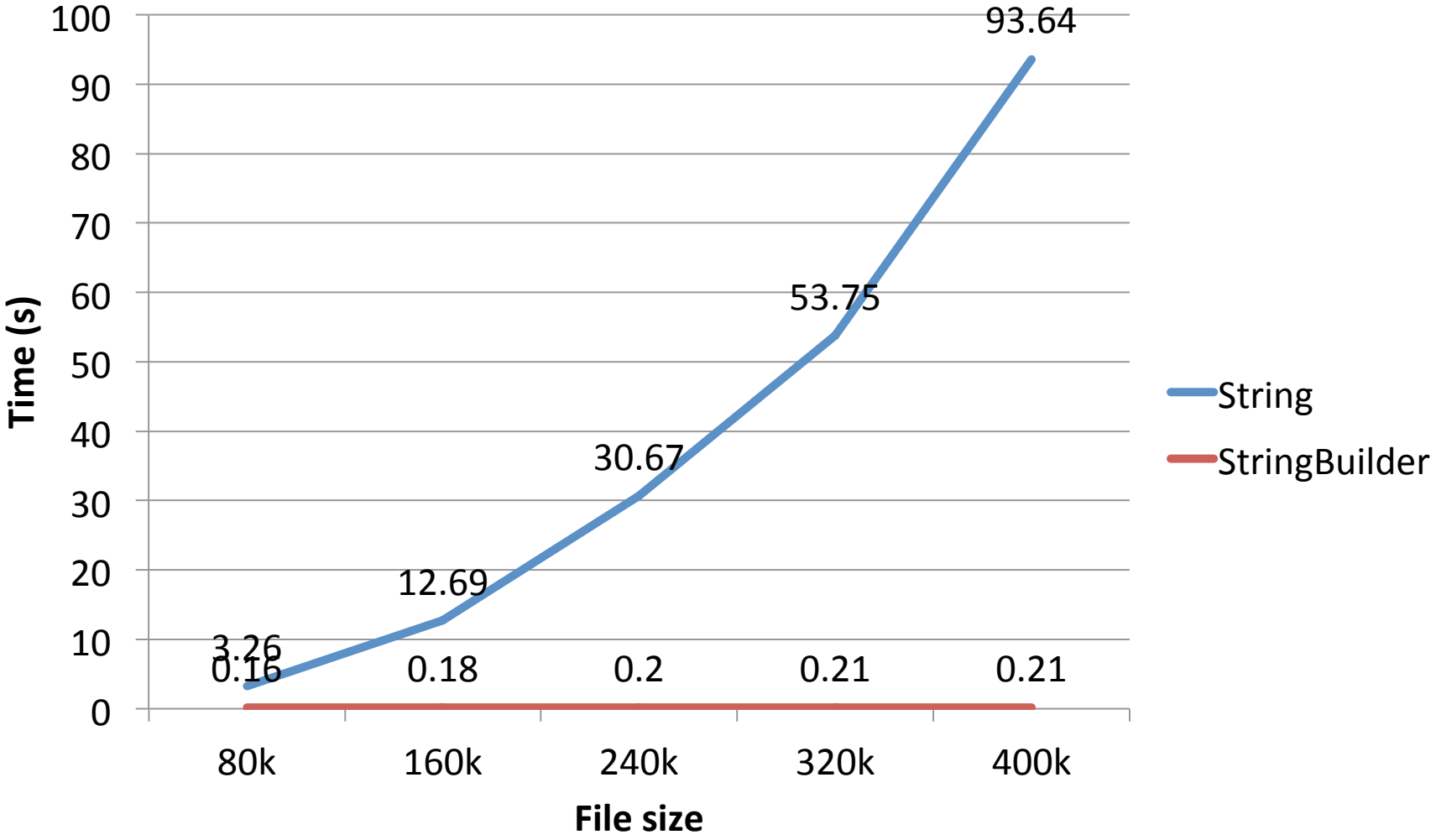
The right way to solve the problem…

- If you execute the following code to read ”War and Peace” (novel by Tolstoy) from a text file, you’ll see that it is **much faster.**

```
StringBuilder result = new StringBuilder();
int num_chars = 0;
Character c = readChar();
while(c != null) {
        result.append(c);
        num_characters += 1;
        c = readChar();
}
System.out.println(num_chars);
System.out.print(result);
```



`String` vs. `StringBuilder` Performance



The right way to solve the problem…

- **Question:** Why is one algorithm very slow and the other one much faster?

VERY SLOW

```
String result = "";
int num_chars = 0;
Character c = readChar();
while(c != null) {
        result += c;
        num_characters += 1;
        c = readChar();
}
System.out.println(num_chars);
System.out.print(result);
```

```
StringBuilder result = new StringBuilder();
int num_chars = 0;
Character c = readChar();
while(c != null) {
        result.append(c);
        num_characters += 1;
        c = readChar();
}
System.out.println(num_chars);
System.out.print(result);
```

MUCH FASTER

Behind the scenes…

- Let's have a closer look at the first algorithm:

```
String result = "";
int num_chars = 0;
Character c = readChar();
while(c != null) {
        result += c;
        num_characters += 1;
        c = readChar();
}
System.out.println(num_chars);
System.out.print(result);
```

A string is basically an array of characters

- `String s = "hello";`
- , or
- `char[] s = {'h','e','l','l','o'};`

Behind the scenes…

- Let's have a closer look at the first algorithm:

```
String result = "";
int num_chars = 0;
Character c = readChar();
while(c != null) {
        result += c;
        num_characters += 1;
        c = readChar();
}
System.out.println(num_chars);
System.out.print(result);
```

Let's focus on what the following line does:

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

Behind the scenes…

- Let's have a closer look at the first algorithm:

```
String result = "";
int num_chars = 0;
Character c = readChar();
while(c != null) {
        result += c;
        num_characters += 1;
        c = readChar(
}
System.out.println(n
System.out.print(res
```

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

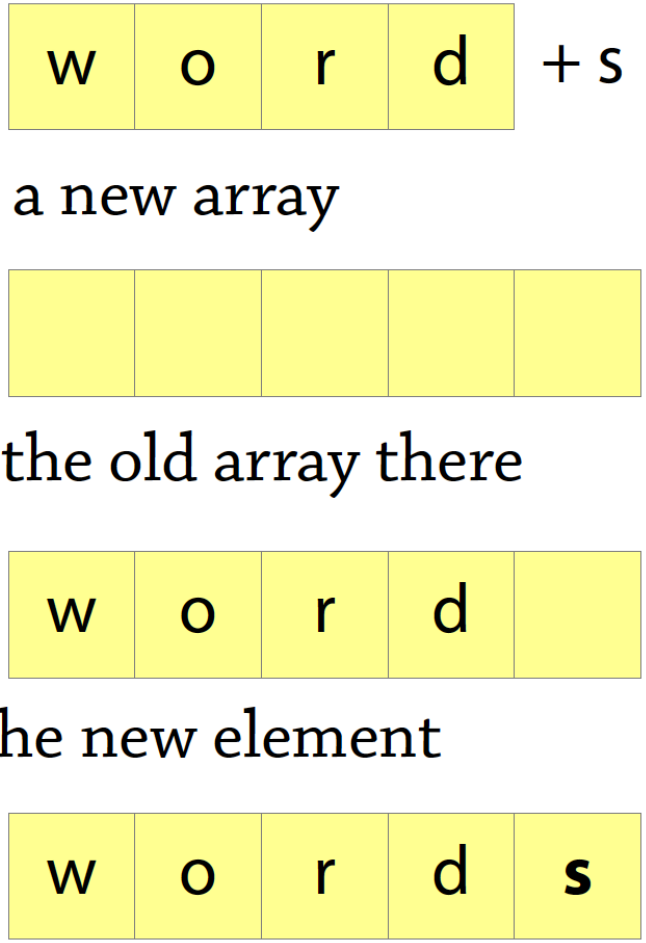
Behind the scenes…

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



Well, is it really so bad?

- Imagine we are reading a file of length n:

Well, is it really so bad?

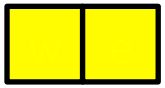
- Imagine we are reading a file of length n:

w + ‘e’

Well, is it really so bad?

- Imagine we are reading a file of length n:

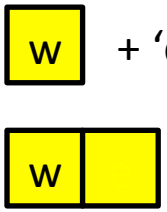
+ ‘e’



- Create an array of size 2

Well, is it really so bad?

- Imagine we are reading a file of length n:

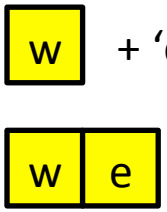


+ ‘e’

- Copy content of previous array to the new array one character at a time

Well, is it really so bad?

- Imagine we are reading a file of length n:

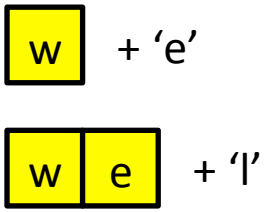


+ ‘e’

- Store the new character at the end

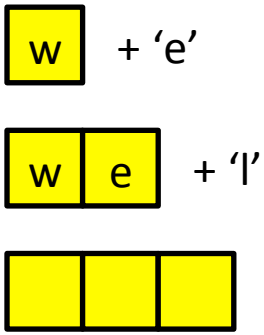
Well, is it really so bad?

- Imagine we are reading a file of length n:



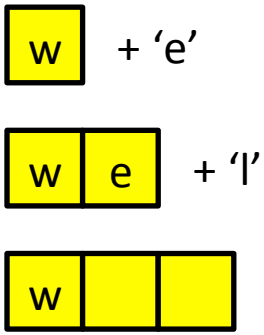
Well, is it really so bad?

- Imagine we are reading a file of length n:



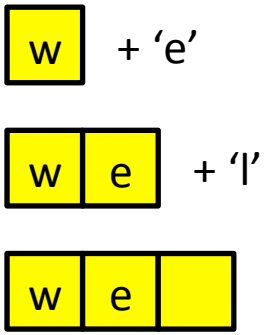
Well, is it really so bad?

- Imagine we are reading a file of length n:



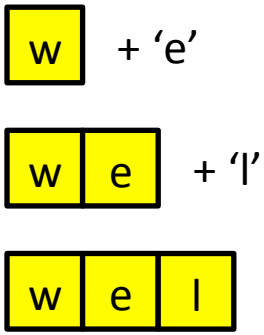
Well, is it really so bad?

- Imagine we are reading a file of length n:



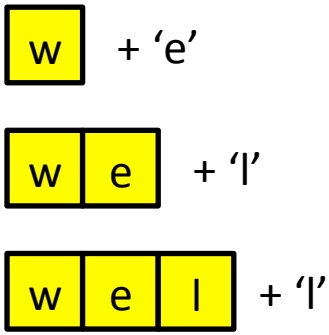
Well, is it really so bad?

- Imagine we are reading a file of length n:



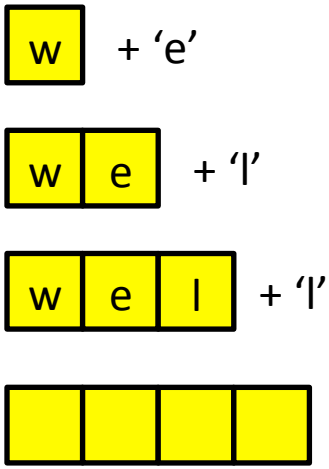
Well, is it really so bad?

- Imagine we are reading a file of length n:



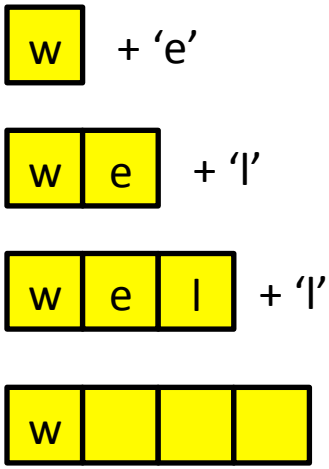
Well, is it really so bad?

- Imagine we are reading a file of length n:



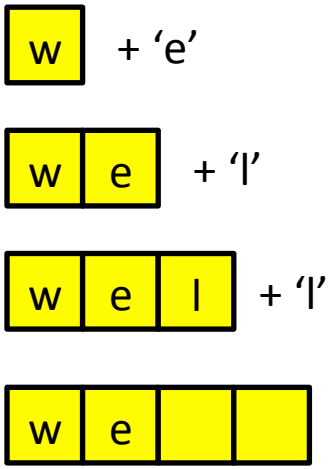
Well, is it really so bad?

- Imagine we are reading a file of length n:



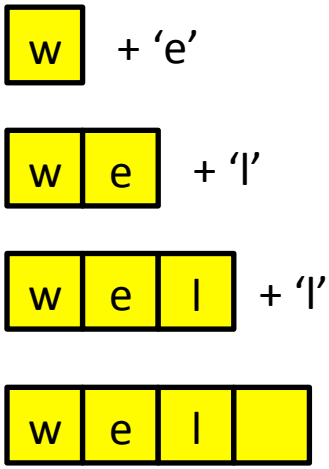
Well, is it really so bad?

- Imagine we are reading a file of length n:



Well, is it really so bad?

- Imagine we are reading a file of length n:



Well, is it really so bad?

- Imagine we are reading a file of length n:

w	+ ‘e’	**1** character copied
w e	+ ‘l’	**2** characters copied
w e l	+ ‘l’	**3** characters copied
w e l l	…….	**4** characters copied
…		
w e l l , P r i n	……. b u r s	**n-1** characters copied
w e l l , P r i n	……. b u r s t	**n** characters copied

Well, is it really so bad?

- Imagine we are reading a file of length n:

w	+ ‘e’	**1** character copied
w e	+ ‘l’	**2** characters copied
w e l	+ ‘l’	**3** characters copied
w e l l	…….	**4** characters copied
…		

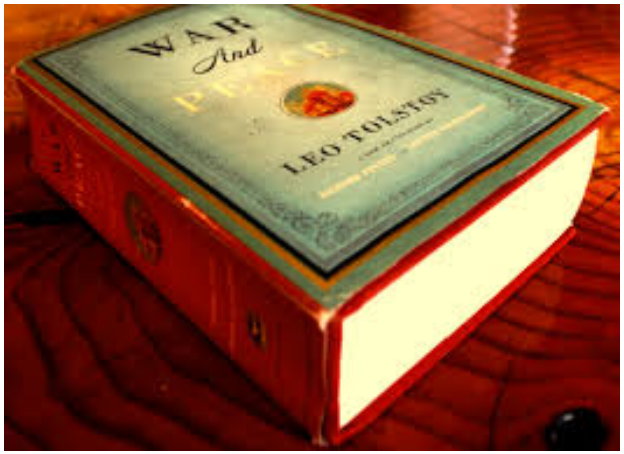
Total $n*n+1/2 \approx n\uparrow 2 /2$ characters are copied!

w	e	l	l	,	P	r	i	n	…….	b	u	r	s						**n-1** characters copied
w	e	l	l	,	P	r	i	n	…….	b	u	r	s	t					**n** characters copied

Well, is it really so bad?

```
String result = "";
int num_chars = 0;
Character c = readChar();
while(c != null) {
        result += c;
        num_characters += 1;
        c = readChar();
}
System.out.println(num_chars);
System.out.print(result);
```

Total $n*n+1/2 \approx n\uparrow 2/2$ characters are copied!



For the novel "War and Peace":
n = 3,600,000

Hence 1800000 x 3600000 = **6,480,000,000,000**
Characters are copied!!!!!

Let’s make some improvements!

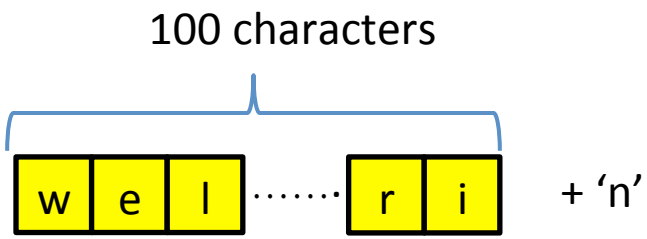
- It's not efficient to copy the whole array every time we append a character.

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 copy anything!

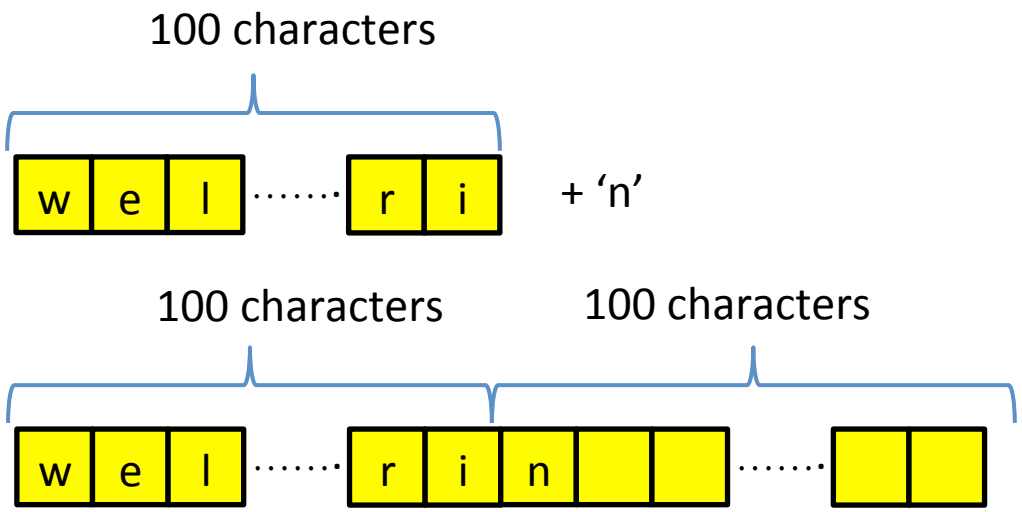
Improvement #1

- Imagine we are reading a file of length n:



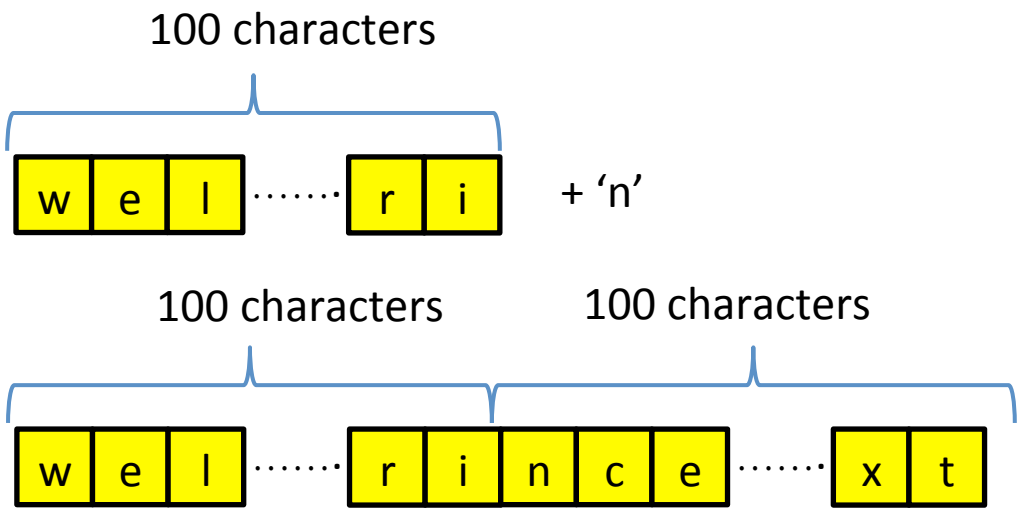
Improvement #1

- Imagine we are reading a file of length n:



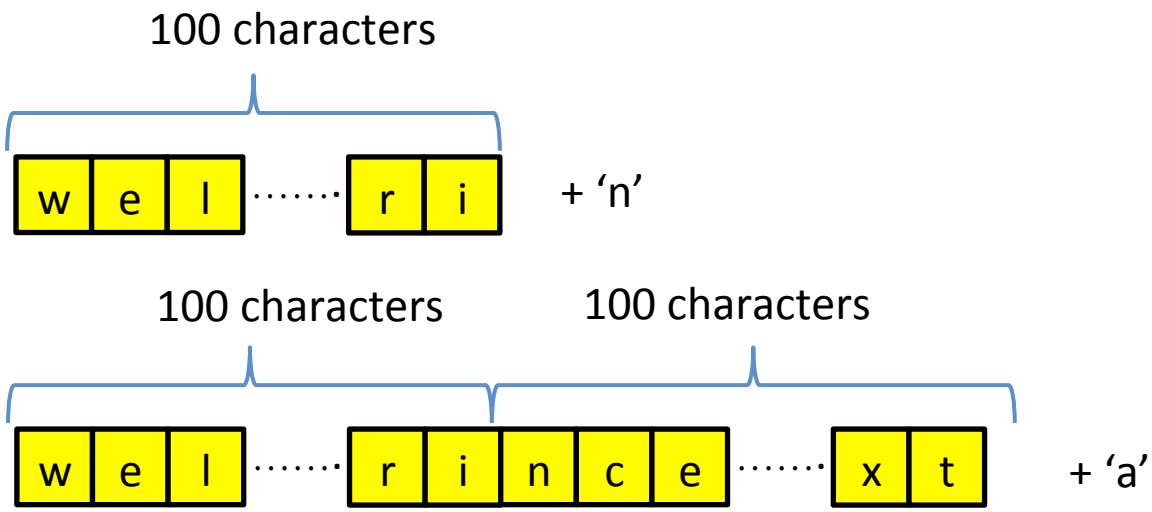
Improvement #1

- Imagine we are reading a file of length n:



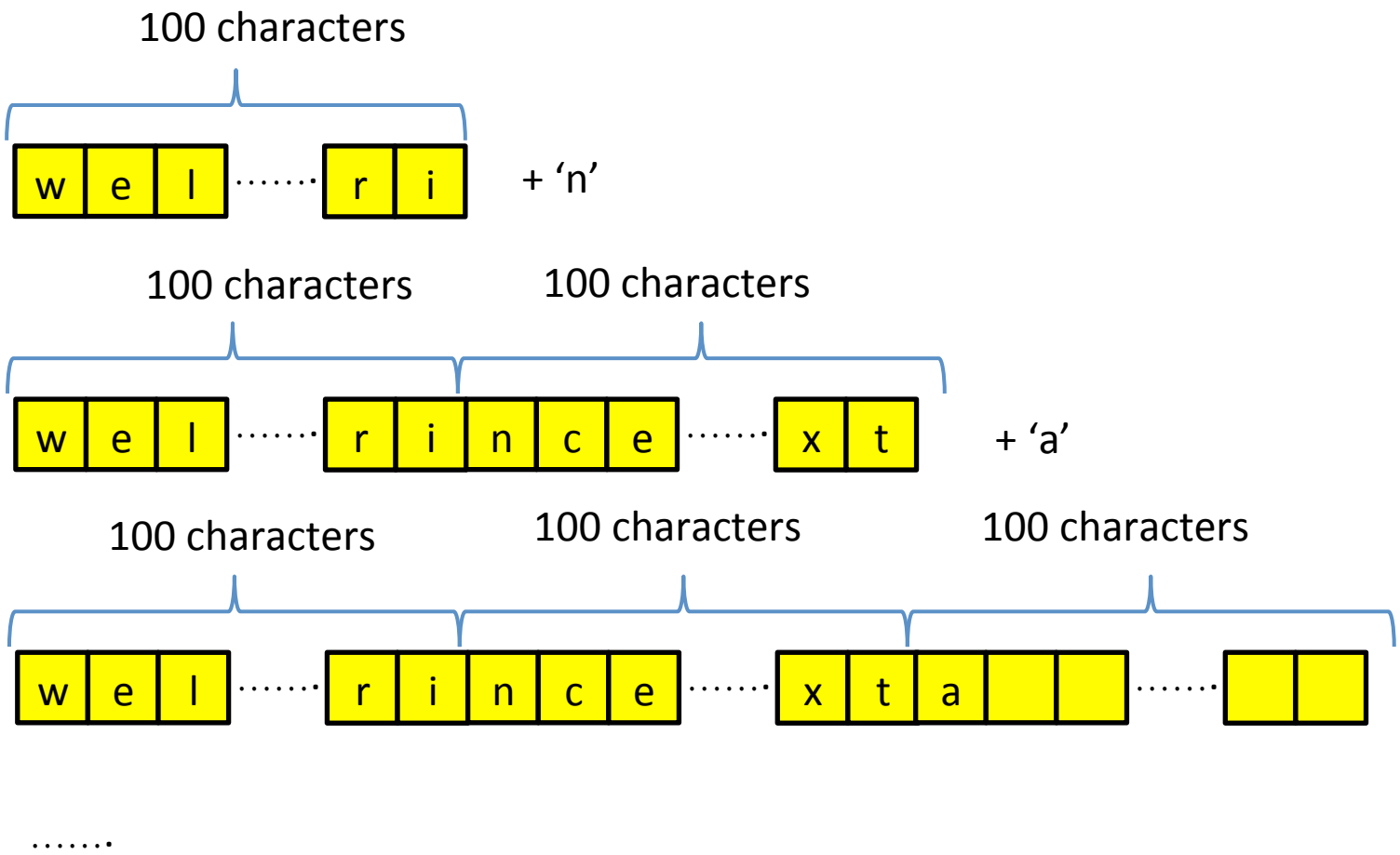
Improvement #1

- Imagine we are reading a file of length n:



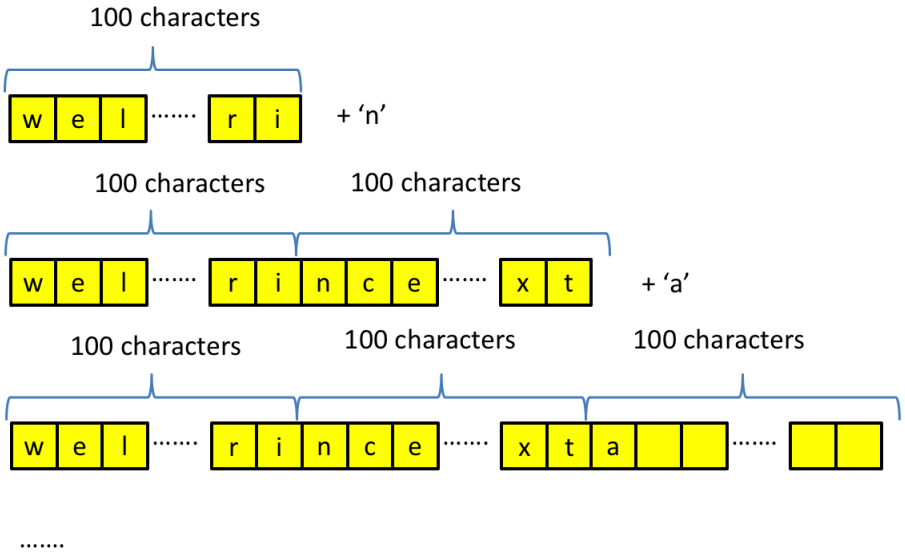
Improvement #1

- Imagine we are reading a file of length n:



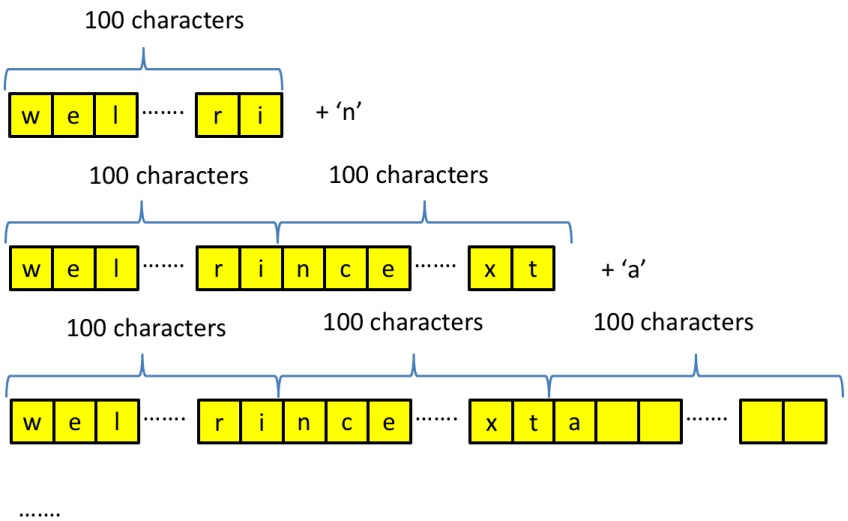
Improvement #1

- **In-Class Exercise 1.3**:
- How many characters of the novel “War and Peace” will be copied now? (n = 3,600,000 characters in total in the book)



Improvement #1

- **In-Class Exercise 1.3**:
- How many characters of the novel "War and Peace" will be copied now? (n = 3,600,000 characters in total in the book)

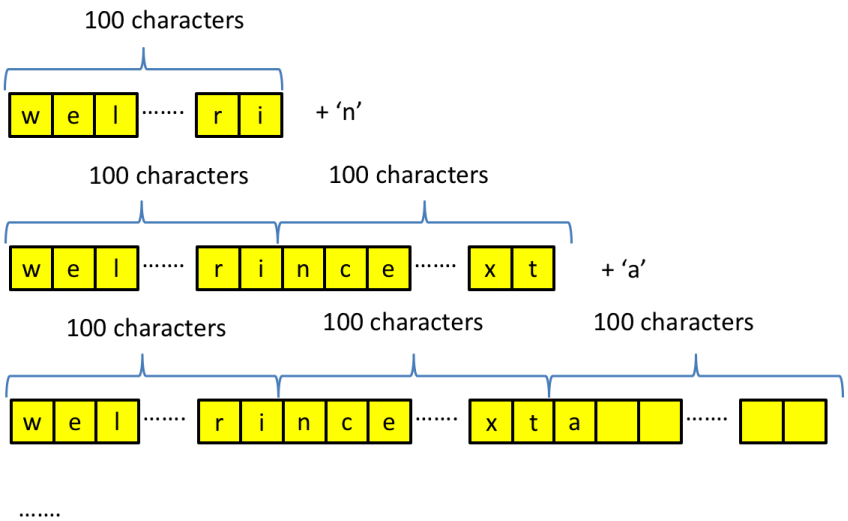


ANSWER:

- We will avoid copying the array for 99 appends out of 100.
- In other words, we will copy the array $(1/100)^{th}$ as often...
- Hence, instead of copying $n^2/2$ = 6,480,000,000,000 characters, we will copy only $n^2/2*100$ = 64,800,000,000

Improvement #1

- **In-Class Exercise 1.3**:
- How many characters of the novel “War and Peace” will be copied now? (n = 3,600,000 characters in total in the book)



ANSWER:

- We will avoid copying the array for 99 appends out of 100.
- In other words, we will copy the array $(1/100)^{th}$ as often...
- Hence, instead of copying $n^2/2$ = 6,480,000,000,000 characters, we will copy only $n^2/2*100$ = 64,800,000,000!

Improvement #2

- Double the size of the array, whenever the array gets full.
- **In-Class Exercise 1.4**: How many character copies will be made in total for the novel “War and Peace” now?

Improvement #2

array size =1
# of copies = 0

Improvement #2

array size =1
# of copies = 0

+ ‘e’

Improvement #2

array size =1
# of copies = 0

+ ‘e’

array size =2
# of copies = 1

Improvement #2

array size =1
# of copies = 0

+ ‘e’

array size =2
# of copies = 1

+ ‘l’

Improvement #2

array size =1
# of copies = 0

+ ‘e’

array size =2
# of copies = 1

+ ‘l’

array size =4
# of copies = 2

Improvement #2

array size =1
# of copies = 0

+ ‘e’

array size =2
# of copies = 1

+ ‘l’

array size =4
# of copies = 2

Improvement #2

array size =1
# of copies = 0

+ ‘e’

array size =2
# of copies = 1

+ ‘l’

array size =4
# of copies = 2

Improvement #2

array size =1
# of copies = 0

+ ‘e’

array size =2
# of copies = 1

+ ‘l’

array size =4
# of copies = 2

+ ‘,’

Improvement #2

array size =1
# of copies = 0

+ ‘e’

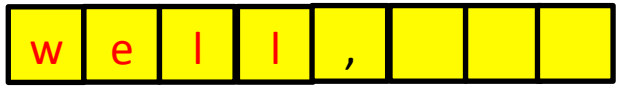
array size =2
# of copies = 1

+ ‘l’

array size =4
# of copies = 2

+ ‘,’

array size =8
# of copies = 4



Improvement #2

array size =1
# of copies = 0

w

+ ‘e’

array size =2
# of copies = 1

w	e

+ ‘l’

array size =4
# of copies = 2

w	e	l	l

+ ‘,’

array size =8
# of copies = 4

w	e	l	l	,	n	o	w

Improvement #2

array size =1
# of copies = 0

w

+ ‘e’

array size =2
# of copies = 1

w	e

+ ‘l’

array size =4
# of copies = 2

w	e	l	l

+ ‘,’

array size =8
# of copies = 4

w	e	l	l	,	n	o	w

+ ‘!’

Improvement #2

array size =1
# of copies = 0

+ ‘e’

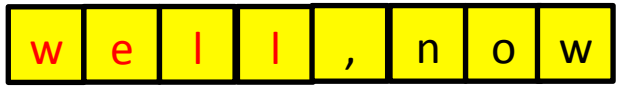
array size =2
# of copies = 1

+ ‘l’

array size =4
# of copies = 2

+ ‘,’

array size =8
# of copies = 4



+ ‘!’

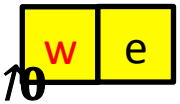
array size =16
# of copies = 8

Improvement #2

array size =1
# of copies = 0

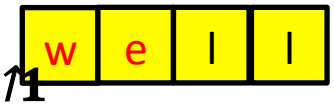
+ ‘e’

array size =2
# of copies = 1=**2/0**



+ ‘l’

array size =4
# of copies = 2=**2/1**



+ ‘,’

array size =8
# of copies = 4=**2/2**

+ ‘!’

array size =16
# of copies = 8=**2/3**

Improvement #2

array size =1
# of copies = 0

w

+ ‘e’

array size =2
# of copies = 1=$\mathbf{2\uparrow 0}$

w	e

+ ‘l’

array size =4
# of copies = 2=$\mathbf{2\uparrow 1}$

w	e	l	l

+ ‘,’

array size =8
# of copies = 4=$\mathbf{2\uparrow 2}$

w	e	l	l	,	n	o	w

+ ‘!’

array size =16
# of copies = 8=$\mathbf{2\uparrow 3}$

w	e	l	l	,	n	o	w	!							

…….

array size = $\mathbf{2\uparrow m}$
# of copies = …

w	e	l	l	,	n	o	w	!

…..

t	,	e	n	d

Improvement #2

array size =1
# of copies = 0

| w |

+ ‘e’

array size =2
# of copies = 1=$2\uparrow 0$

| w | e |

+ ‘l’

array size =4
# of copies = 2=$2\uparrow 1$

| w | e | l | l |

+ ‘,’

array size =8
# of copies = 4=$2\uparrow 2$

| w | e | l | l | , | n | o | w |

+ ‘!’

array size =16
# of copies = 8=$2\uparrow 3$

| w | e | l | l | , | n | o | w | ! | | | | | | | |

…….

array size = $2\uparrow m$
# of copies = …

| w | e | l | l | , | n | o | w | ! | ….. | t | , | e | n | d |

Assume that this is the step just before the last step.

Improvement #2

array size =1
# of copies = 0

w

+ ‘e’

array size =2
# of copies = 1=$\mathbf{2\uparrow 0}$

w	e

+ ‘l’

array size =4
# of copies = 2=$\mathbf{2\uparrow 1}$

w	e	l	l

+ ‘,’

array size =8
# of copies = 4=$\mathbf{2\uparrow 2}$

w	e	l	l	,	n	o	w

+ ‘!’

array size =16
# of copies = 8=$\mathbf{2\uparrow 3}$

w	e	l	l	,	n	o	w	!							

…….

array size = $\mathbf{2\uparrow m}$
# of copies = …

w	e	l	l	,	n	o	w	!

…..

t	,	e	n	d

<u>Question</u>: Assume that array size is $2\uparrow m$ at the step just before the last step. What would $2\uparrow m$ be equal to in the **worst case** so that **maximum** possible number of **character copies** would take place?

Improvement #2

array size =1
# of copies = 0

w

+ ‘e’

array size =2
# of copies = 1=$\mathbf{2\uparrow 0}$

w	e

+ ‘l’

array size =4
# of copies = 2=$\mathbf{2\uparrow 1}$

w	e	l	l

+ ‘,’

array size =8
# of copies = 4=$\mathbf{2\uparrow 2}$

w	e	l	l	,	n	o	w

+ ‘!’

array size =16
# of copies = 8=$\mathbf{2\uparrow 3}$

w	e	l	l	,	n	o	w	!							

…….

array size = $\mathbf{2\uparrow m}$
# of copies = …

w	e	l	l	,	n	o	w	!

…..

t	,	e	n	d

Improvement #2

array size =1
# of copies = 0

w

+ 'e'

array size =2
# of copies = 1=2^0

w	e

+ 'l'

array size =4
# of copies = 2=2^1

w	e	l	l

+ ','

array size =8
# of copies = 4=2^2

w	e	l	l	,	n	o	w

+ '!'

array size =16
# of copies = 8=2^3

w	e	l	l	,	n	o	w	!							

.....

array size = 2^m
# of copies = ...

w	e	l	l	,	n	o	w	!	.....	t	,	e	n	d

+ '.'

You double the size of the array only for the last character in the book "War and Peace"

$\mathbf{2\uparrow m = N-1 \approx N}$, since N=3,600,000 is a large number

Improvement #2

array size =1
# of copies = 0

| w |

+ 'e'

array size =2
# of copies = 1=2^0

| w | e |

+ 'l'

array size =4
# of copies = 2=2^1

| w | e | l | l |

+ ','

array size =8
# of copies = 4=2^2

| w | e | l | l | , | n | o | w |

+ '!'

array size =16
# of copies = 8=2^3

| w | e | l | l | , | n | o | w | ! | | | | | | | |

.....

array size = $2\uparrow m$
# of copies = ...

| w | e | l | l | , | n | o | w | ! | ..... | t | , | e | n | d |

+ '.'

array size = $2\uparrow m+1$
# of copies = $\mathbf{2\uparrow m}$

| w | e | l | l | , | n | o | w | ! | ..... | t | , | e | n | d | . |

Improvement #2

array size =1
# of copies = 0

| w |

+ 'e'

array size =2
# of copies = 1=2^0

| w | e |

+ 'l'

array size =4
# of copies = 2=2^1

| w | e | l | l |

+ ','

array size =8
# of copies = 4=2^2

| w | e | l | l | , | n | o | w |

+ '!'

array size =16
# of copies = 8=2^3

| w | e | l | l | , | n | o | w | ! | | | | | | | |

.....

array size = $2\uparrow m$
# of copies = ...

| w | e | l | l | , | n | o | w | ! | ..... | t | , | e | n | d |

+ '.'

array size = $2\uparrow m+1$
# of copies = $\mathbf{2\uparrow m}$

| w | e | l | l | , | n | o | w | ! | ..... | t | , | e | n | d | . |

Total number of character copies
$= 2\uparrow 0 + 2\uparrow 1 + 2\uparrow 2 + 2\uparrow 3 + \ldots + 2\uparrow m$
$= \sum i=0\uparrow m \; 2\uparrow i$

Improvement #2

array size =1
# of copies = 0

| w |

+ 'e'

array size =2
# of copies = 1=2^0

| w | e |

+ 'l'

array size =4
# of copies = 2=2^1

| w | e | l | l |

+ ','

array size =8
# of copies = 4=2^2

| w | e | l | l | , | n | o | w |

+ '!'

array size =16
# of copies = 8=2^3

| w | e | l | l | , | n | o | w | ! | | | | | | | |

.....

array size = $2\uparrow m$
# of copies = ...

| w | e | l | l | , | n | o | w | ! | ..... | t | , | e | n | d |

+ '.'

array size = $2\uparrow m+1$
# of copies = $\mathbf{2\uparrow m}$

| w | e | l | l | , | n | o | w | ! | ..... | t | , | e | n | d | . |

Total number of character copies
$= 2\uparrow 0 + 2\uparrow 1 + 2\uparrow 2 + 2\uparrow 3 + \ldots + 2\uparrow m$
$= \sum i=0\uparrow m \; 2\uparrow i$ → This is **finite geometric series**

Improvement #2

array size =1
# of copies = 0

w

+ 'e'

array size =2
# of copies = 1=2^0

w	e

+ 'l'

array size =4
# of copies = 2=2^1

w	e	l	l

+ ','

array size =8
# of copies = 4=2^2

w	e	l	l	,	n	o	w

+ '!'

array size =16
# of copies = 8=2^3

w	e	l	l	,	n	o	w	!							

.....

array size = $2\uparrow m$
# of copies = ...

w	e	l	l	,	n	o	w	!

..... | t | , | e | n | d |

+ '.'

array size = $2\uparrow m+1$
# of copies = $\mathbf{2\uparrow m}$

w	e	l	l	,	n	o	w	!

..... | t | , | e | n | d | . |

Total number of character copies
$= 2\uparrow 0 + 2\uparrow 1 + 2\uparrow 2 + 2\uparrow 3 + \ldots + 2\uparrow m$
$= \sum i=0\uparrow m\, 2\uparrow i = 1-2\uparrow m+1 / 1-2$
$= 2\uparrow m+1$

Improvement #2

array size =1
# of copies = 0

w

+ 'e'

array size =2
# of copies = 1=2^0

w	e

+ 'l'

array size =4
# of copies = 2=2^1

w	e	l	l

+ ','

array size =8
# of copies = 4=2^2

w	e	l	l	,	n	o	w

+ '!'

array size =16
# of copies = 8=2^3

w	e	l	l	,	n	o	w	!							

.....

array size = $2\uparrow m$
# of copies = ...

w	e	l	l	,	n	o	w	!

.....

t	,	e	n	d

+ '.'

array size = $2\uparrow m+1$
# of copies = $\mathbf{2\uparrow m}$

w	e	l	l	,	n	o	w	!

.....

t	,	e	n	d	.

- Total number of character copies = $2\uparrow m+1$
- $2\uparrow m+1$ = ? (in terms of N)

Improvement #2

array size =1
# of copies = 0

w

+ 'e'

array size =2
# of copies = 1=2^0

w	e

+ 'l'

array size =4
# of copies = 2=2^1

w	e	l	l

+ ','

array size =8
# of copies = 4=2^2

w	e	l	l	,	n	o	w

+ '!'

array size =16
# of copies = 8=2^3

w	e	l	l	,	n	o	w	!							

.....

array size = $2\uparrow m$
# of copies = ...

w	e	l	l	,	n	o	w	!

..... | t | , | e | n | d |

+ '.'

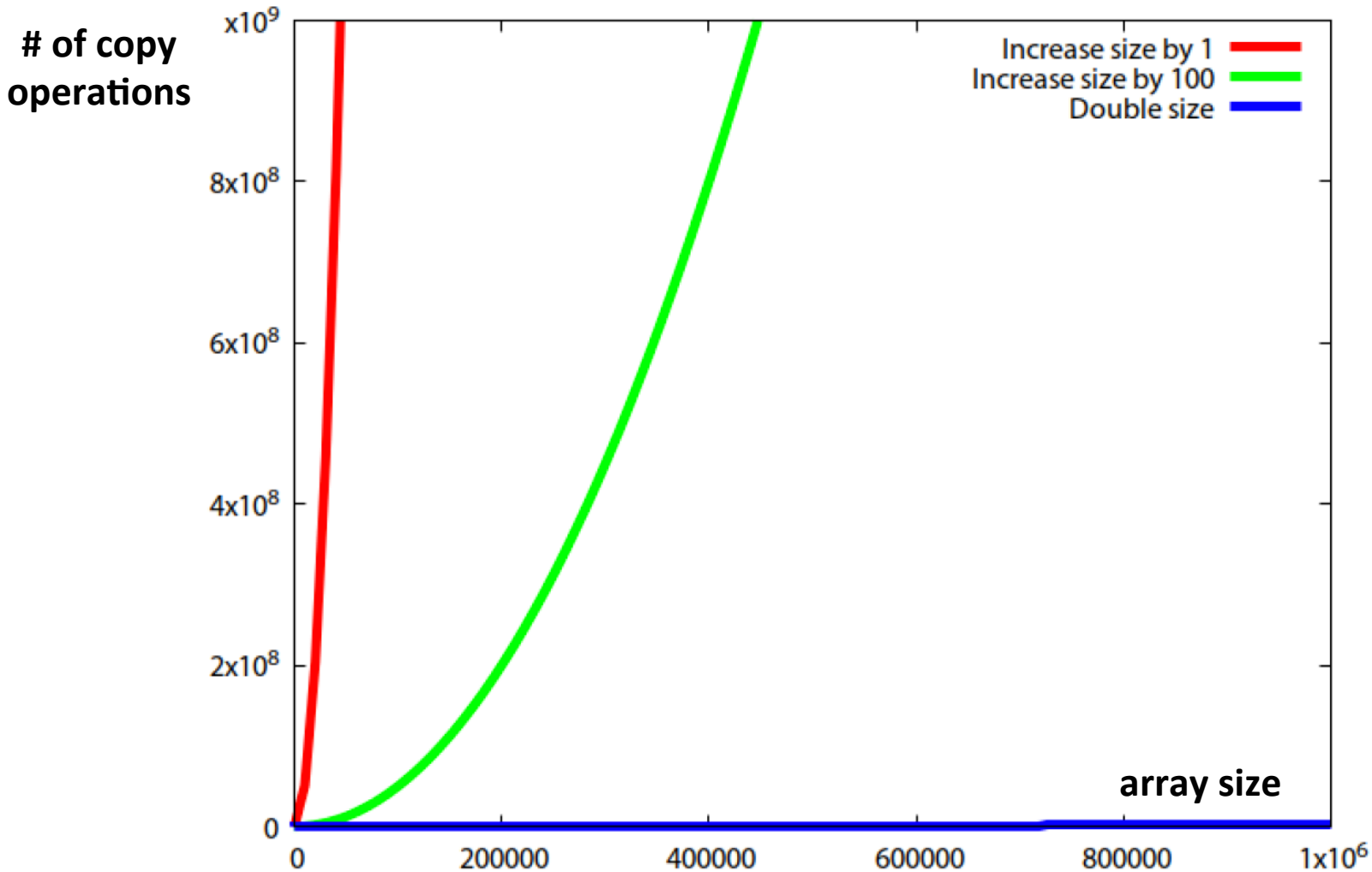
array size = $2\uparrow m+1$
# of copies = $\mathbf{2\uparrow m}$

w	e	l	l	,	n	o	w	!

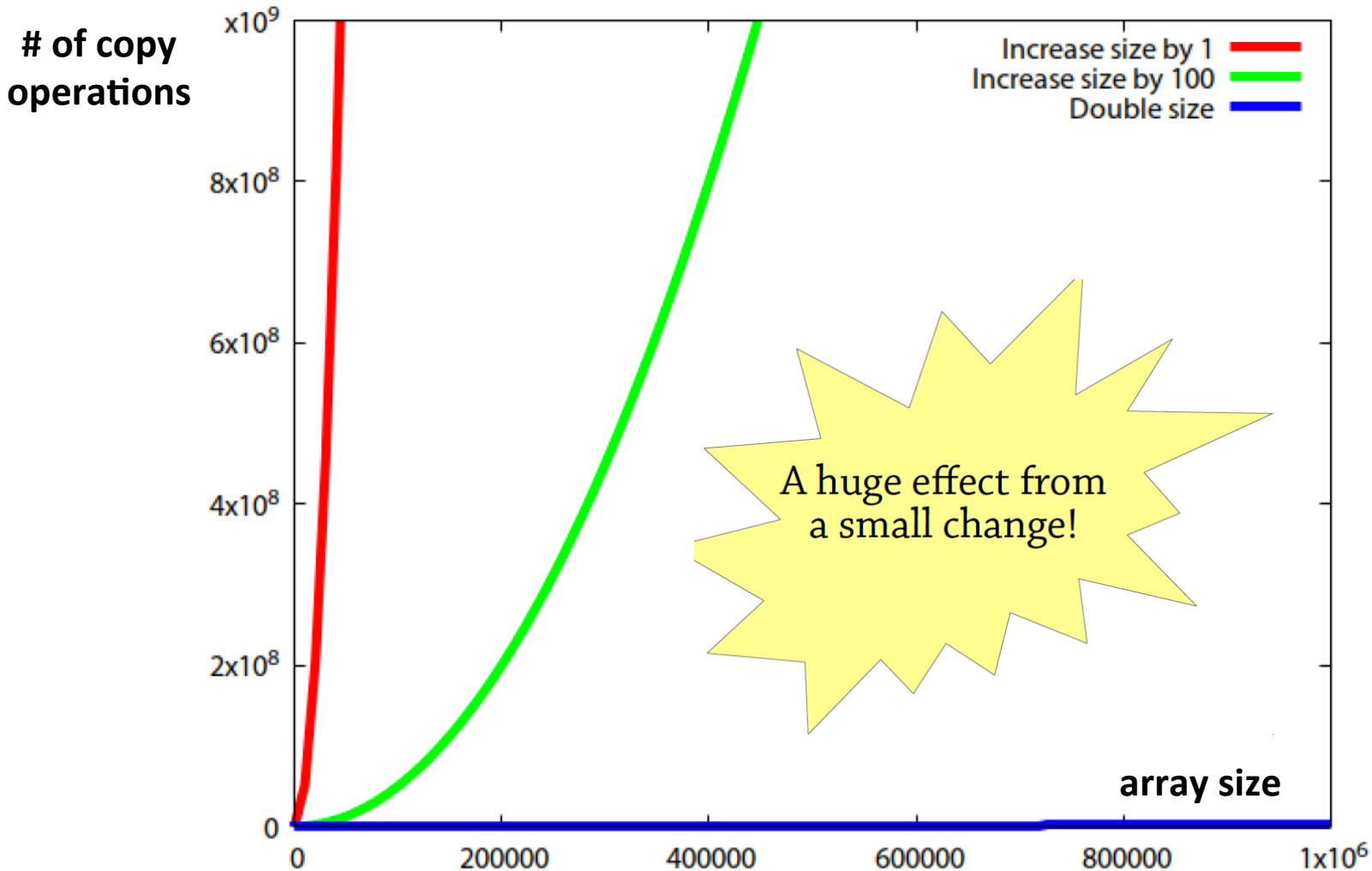
..... | t | , | e | n | d | . |

- Total number of character copies = $2\uparrow m+1$
- Also, previously we found that $2\uparrow m \approx N$
- Hence, total number of character copies is $\approx 2N = 7{,}200{,}0000$ ($where$ $N = 3{,}600{,}000$)

Performance



Performance



Why does it work really?

- The important property:
  - After resizing the array, the new array is exactly half full.
  - For every “expensive” step of copying 2n characters, there are n “cheap” steps with no copying => constant cost of 2 characters copied per step
- **Recommended Exercise:** Also works if we e.g. increase array size by 50% instead of doubling!

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-readin 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 siz of the array each time
- Called `ArrayList` in Java

About `String` 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
- Question: Why can't the `String` class use a dynamic array?