CompSci 201, L7: Runtime Efficiency

(Also more about `String` vs `StringBuilder`)

Logistics, Coming up

- Today 2/6
  - Project 1 Nbody due today
  - Project 2 Markov releasing tomorrow (due in 2 weeks)
- Wednesday 2/8
  - APT 3 due
- Next Monday 2/13
  - Midterm Exam 1
    - Covers everything through THIS week, up to and including asymptotic analysis / Big O
    - Example/Practice exams available this evening

Midterm Exams

See details on [course website assignments and grading page]

- 60 minutes, in-class exam.
- Short-answer problems. Reason about algorithms, data structures, code.
- Can bring 1 double sided reference sheet (8.5x11 in), write/type whatever notes you like.
- No electronic devices out during exam

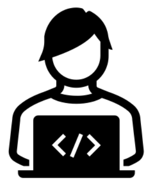
Exam Grades and Missing Exams

- Three midterm exams scheduled (E1, E2, E3).
- Final exam has 3 corresponding parts (F1, F2, F3).
- Overall course exam grade is:

  AVG(max(E1, F1), max(E2, F2), max(E3, F3))

- Meaning the final exam serves:
  - As a makeup, if you need to miss a midterm, and/or
  - As an opportunity to demonstrate more learning, if you're unhappy with your midterm score.

Problem Solving with Maps
Word Pattern Problem

Live Coding 

[Word Pattern Problem]

Runtime Efficiency, an Empirical Look at String Concatenation

Two methods for repeated concatenation

```java
public static String repeatConcatA(int reps, String toConcat) {
    String result = new String();
    for (int i=0; i<reps; i++) {
        result += toConcat;
    }
    return result;
}
```

methodA: Using String object and basic + operator

```java
public static String repeatConcatB(int reps, String toConcat) {
    StringBuilder result = new StringBuilder();
    for (int i=0; i<reps; i++) {
        result.append(toConcat);
    }
    return result.toString();
}
```

methodB: Using StringBuilder object and append method

Empirical timing experiment

Can see the code on gitlab here.

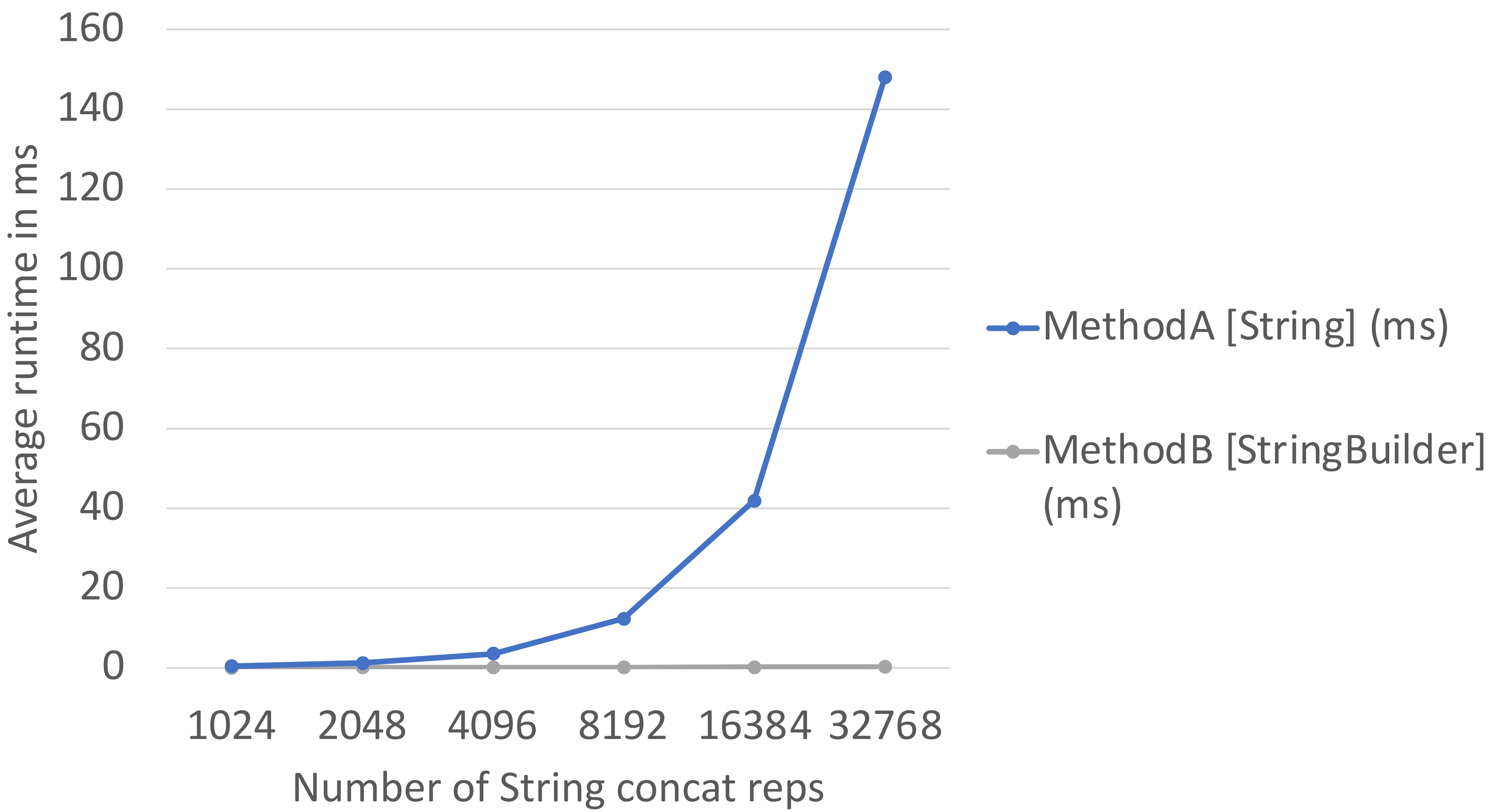
```java
public class StringConcatTiming {
    static final int NUM_TRIALS = 100;
    static final int REPS_PER_TRIAL = 1024;
    static final String TO_CONCAT = "201";

    Run | Debug
    public static void main(String[] args) {
        long totalTime = 0;
        for (int trial=0; trial<NUM_TRIALS; trial++) {
            long startTime = System.nanoTime();
            //repeatConcatA(REPS_PER_TRIAL, TO_CONCAT);
            repeatConcatB(REPS_PER_TRIAL, TO_CONCAT);
            long endTime = System.nanoTime();
            totalTime += (endTime - startTime);
        }
        double avgTime = (double)totalTime / NUM_TRIALS;
        System.out.printf("Avg time per trial is %f ms", avgTime*1E-6);
    }
```

static final used for constants here

Going to time both methods separately.

Empirical results



Empirical results in more detail

Reps	MethodA (ms)	MethodB (ms)
1024	0.384	0.050
2048	1.136	0.061
4096	3.443	0.077
8192	12.244	0.099
16384	41.754	0.143
32768	147.719	0.207

Multiply reps by 2 multiplies runtime by 4. Quadratic complexity.

Multiply reps by 2 multiplies runtime by ~2. Linear complexity.

Empirical results in more detail

Reps	MethodA ns/rep	MethodB ns/rep
1024	0.375	0.048
2048	0.555	0.030
4096	0.841	0.019
8192	1.495	0.012
16384	2.548	0.009
32768	4.508	0.006

Runtime / rep increasing, *greater than linear* complexity.

Runtime / rep not increasing, at most linear complexity.

What’s going on? Documentation?

docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String

Class String

java.lang.Object
    java.lang.String

All Implemented Interfaces:

`Serializable, CharSequence, Comparable<String>, Constable, ConstantDesc`

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence, Constable, ConstantDesc
```

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented a

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings.

methodA revisited

```
19    public static String repeatConcatA(int reps, String toConcat) {
20        String result = new String();
21        for (int i=0; i<reps; i++) {
22            result += toConcat;
23        }
24        return result;
25    }
```

String is immutable, line 22 creates a new string and copies result then toConcat.

How many characters will be copied per iteration if `toConcat == “201”`?

- i=0: 3
- i=1: 6
- i=2: 9
- …
- On iteration i, need to copy 3*(i+1) characters!

How many total characters are copied? Algebra!

methodA: i goes from 0 to reps-1, copy 3*(i+1) characters per iteration.

$$\sum_{i=0}^{\mathrm{reps}-1} 3(\mathrm{i}+1) = 3(\mathrm{reps}) + 3\left(\sum_{i=0}^{\mathrm{reps}-1} \mathrm{i}\right)$$

$$= 3(\mathrm{reps}) + 3\left(\frac{\mathrm{reps}}{2}\right)(0 + \mathrm{reps} - 1)$$

$$= \frac{3}{2}(\mathrm{reps}^2 + \mathrm{reps})$$

Arithmetic series formula:

$$\sum_{i=1}^{n} a_i = \left(\frac{n}{2}\right)(a_1 + a_n)$$

Abstracting, Intro to Big O Notation (Preview for next time)

- The 3/2 in $\frac{3}{2}\text{reps}^2$ doesn't tell us much about how the performance *scales with the size of reps*.

- Often, we use *asymptotic notation*, especially *Big O notation* to abstract away constants.

- For example: let N = reps, then we say that the asymptotic runtime complexity is $O(N^2)$.
  - If you ~double N, you ~quadruple the runtime

Two general Big O rules

1. Can drop constants
   - $2N+3 \rightarrow O(N)$
   - $0.001N + 1{,}000{,}000 \rightarrow O(N)$

2. Can drop lower order terms
   - $2N^2+3N \rightarrow O(N^2)$
   - $N+\log(N) \rightarrow O(N)$
   - $2^N + N^2 \rightarrow O(2^N)$

Hiearchy of some common complexity classes

Big O	Name	Example
$O(2^N)$	Exponential	Calculate all subsets of a set
$O(N^3)$	Cubic	Multiply NxN matrices
$O(N^2)$	Quadratic	Loop over all *pairs* from N things
O(N log(N))	Nearly-linear	Sorting algorithms
O(N)	Linear	Loop over N things
O(log(N))	Logarithmic	Binary search a sorted list
O(1)	Constant	Addition, array access, etc.

How does StringBuilder work?

"Every string builder has a capacity. As long as the length of the character sequence contained in the string builder does not exceed the capacity, it is not necessary to allocate a new internal buffer. If the internal buffer overflows, it is automatically made larger." - StringBuilder JDK 17 documentation.

- But how does it grow?
- Geometrically! Like ArrayList, HashMap, …
  - Still linear amortized complexity, for same reasons

StringBuilder is like an ArrayList of characters

- Suppose we run the code:

```
StringBuilder() sb = new StringBuilder(3);
sb.append(“hi”);
sb.append(“ya”);
```

Initial buffer/array capacity

Array representing StringBuilder

~~h~~	~~i~~	~~y~~

h	i	y	a		

How many total characters are copied with a StringBuilder?

Suppose we start with capacity 3 and double when out of capacity, appending a length 3 string reps times…

$$3 \cdot \text{reps} + \sum_{i=0}^{\approx \log_2(3 \cdot \text{reps})} 2^i = 3 \cdot \text{reps} + 6 \cdot \text{reps}$$

The “good case” copies

From doubling and copying the array

Geometric series formula:
$$\sum_{i=0}^{n} a\, r^i = a\left(\frac{1 - r^{n+1}}{1 - r}\right)$$

$$= 9 \cdot \text{reps}$$

Memory/Runtime Tradeoff

```java
public static String repeatConcatB(int reps, String toConcat) {
    StringBuilder result = new StringBuilder();
    for (int i=0; i<reps; i++) {
        result.append(toConcat);
    }
    System.out.printf("String builder capacity is %d characters%n", result.capacity());
    System.out.printf("Result length is %d characters%n", result.length());
    return result.toString();
}
```

PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL

```
String builder capacity is 147454 characters
Result length is 98304 characters
```

Final StringBuilder is using about 146k / 98k ~= 1.5 times as much memory as necessary. Very common tradeoff in data structures!

What's the real difference between methodA and methodB?

- methodA: Copies roughly $\frac{3}{2}(\text{reps}^2 - \text{reps})$
- methodB: copies roughly $9 \cdot \text{reps characters.}$

Reps	~MethodA char copies (millions)	MethodB char copies (millions)
1000	1.5	0.009
2000	6	0.018
4000	24	0.036
8000	95	0.072
16000	383	0.144
32000	1535	0.288

WOTO

Go to duke.is/8qfjk

Not graded for correctness, just participation.

Try to answer *without* looking back at slides and notes.

But do talk to your neighbors!

2

How many **total characters must be copied** by the code **on lines 8 and 9**? Remember that Strings are immutable in Java. *

```
7    String s = "hi";
8    s += "hey";
9    s += s;
```

- 5
- 7
- 9
- 10
- 15
- 30

3

Suppose method A has linear complexity and takes 10 ms to run on an input of size N. About what would you expect the runtime to be for an input of size 2*N? *

- 10 ms
- 20 ms
- 40 ms
- 100 ms

4

Suppose method B has quadratic complexity and takes 10 ms to run on an input of size N. About what would you expect the runtime to be for an input of size 2*N? *

- 10 ms
- 20 ms
- 40 ms
- 100 ms

5

Here is another String concatenation method. Suppose the input string s has a small number of characters, say 3. As a function of the parameter reps, how would you characterize the runtime complexity of the method? Hint: As a function of reps, how many total characters will be copied across all iterations of the loop? *

```
public static String concatAlot(int reps, String s) {
    for (int i=0; i<reps; i++) {
        s += s;
    }
    return s;
}
```

- ○ Constant
- ○ Linear
- ○ Quadratic
- ○ Exponential