Lecture 12: Binary Search; Complexity

07/20/22

Reminders

- A2 Resubmission due Wednesday 7/20 @ 11:59pm
- Checkpoint 5 due Sunday 7/24 @ 11:59 pm

Midterm on Friday

- We give <u>lots</u> of partial credit! Write down everything you know.
  - No credit for pseudocode or comments – only for real code
  - Your code doesn’t have to be complete to earn credit
    - If you know you need a while loop, but don’t exactly know what the condition is, write down the while loop anyway
    - etc.
- Manage your time well.
  - Move on to the next question if you feel like you’re stuck
- Don’t write before or after time is called – nothing you write is worth -10 points

Midterm on Friday

- Bring your Husky ID

- We will start at 10:50 sharp to give you the full 60 minutes.
    - Arrive early!

- Make sure you sleep!

Complexity / Efficiency

- Finally!
- Best of CS

Sum up numbers 1 to n

- Let's write a method to calculate the sum from 1 to some n:

```
public static int sum1(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum += i;
    }
    return sum;
}
```

- Gauss also has a way of solving this:

```
public static int sum2(int n) {
    return n * (n + 1) / 2;
}
```

Which one is more efficient?

$$1 + 2 + \ldots + n = \frac{n(n+1)}{2}$$

Efficiency

- **Efficiency:** measure of computing resources used by code.
  - Resources:
    - Time
    - Space
    - Energy
    - …
  - Most commonly refers to <u>time</u>

- We want to be able to compare different algorithms to see which is more efficient

Runtime Efficiency Try 1

- Let's time the methods!

```
n = 1                sum1 took    0ms, sum2 took 0ms
n = 5                sum1 took    0ms, sum2 took 0ms
n = 10               sum1 took    0ms, sum2 took 0ms
n = 100              sum1 took    0ms, sum2 took 0ms
n = 1,000            sum1 took    1ms, sum2 took 0ms
n = 10,000,000       sum1 took    8ms, sum2 took 0ms
n = 100,000,000      sum1 took   43ms, sum2 took 0ms
n = 2,147,483,647    sum1 took  804ms, sum2 took 0ms
```

Runtime Efficiency Try 1

- Let's time the methods!

```
n = 1                sum1 took    0ms, sum2 took 0ms
n = 5                sum1 took    0ms, sum2 took 0ms
n = 10               sum1 took    0ms, sum2 took 0ms
n = 100              sum1 took    0ms, sum2 took 0ms
n = 1,000            sum1 took    0ms, sum2 took 0ms
n = 10,000,000       sum1 took   10ms, sum2 took 0ms
n = 100,000,000      sum1 took   47ms, sum2 took 0ms
n = 2,147,483,647    sum1 took  784ms, sum2 took 0ms
```

Runtime Efficiency Try 1

- Different computers give different results
- The same computer gives different results!!! D:<

- Let's time the methods!

```
n = 1              sum1 took    0ms, sum2 took 0ms
n = 5              sum1 took    0ms, sum2 took 0ms
n = 10             sum1 took    0ms, sum2 took 0ms
n = 100            sum1 took    0ms, sum2 took 0ms
n = 1,000          sum1 took    1ms, sum2 took 0ms
n = 10,000,000     sum1 took    3ms, sum2 took 0ms
n = 100,000,000    sum1 took  121ms, sum2 took 0ms
n = 2,147,483,647  sum1 took 1750ms, sum2 took 0ms
```

Runtime Efficiency Try 2

- Count number of “simple steps” our algorithm takes to run
- Assume the following:
  - Statement: any single statement 1 step to run
    - `int x = 5;`
    - `boolean b = (5 + 1 * 2) < 15 + 3;`
    - `System.out.println(“Hello”);`
  - Loop: number of times the loop runs * the steps in the body
  - Method call: total number of steps inside the method's body

Efficiency Example

$3 + N + 3N$

```
public static void method1(int N) {
    statement1;
    statement2;
    statement3;

    for (int i = 1; i <= N; i++) {
        statement4;
    }

    for (int i = 1; i <= N; i++) {
        statement5;
        statement6;
        statement7;
    }
}
```

3

$N \times 1 = N$

1

$3N$

3

How many “steps” are in this method?

```
public static void method2(int N) {
    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= N; j++) {
            statement1;
        }
    }

    for (int i = 1; i <= N; i++) {
        statement2;
        statement3;
        statement4;
        statement5;
    }
}
```

$N^2 + 4N$

N^2

N

1

$4N$

4

Sum: how many steps in each?

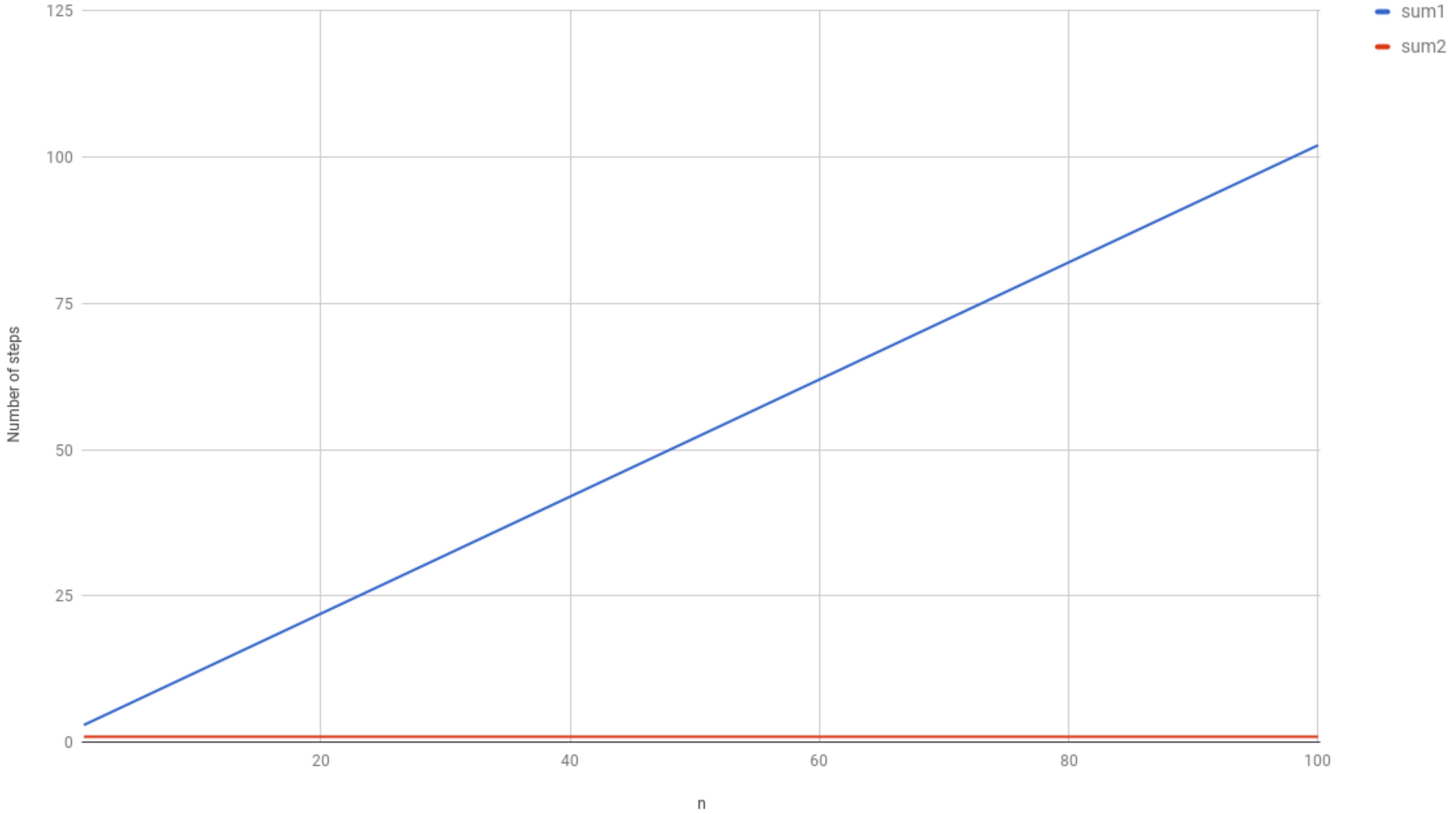
$f(N) = N + 2$

```
public static int sum1(int n) {
    int sum = 0; 1
  [ for (int i = 1; i <= n; i++) {
N |       sum += i; 1
  [ }
    return sum; 1
}
```

$g(N) = 1$

```
public static int sum2(int n) {
    return n * (n + 1) / 2; 1
}
```

sum1 vs. sum2



Big-O

We report runtime efficiency in terms of the general growth rate of an algorithm.

- **N:** size of the input data
- **Growth rate**: change in runtime as N changes.

Big-O is our notation for reporting this growth rate!

- We only care about the general growth rate
- We do not care about specific scaling factors

Big-O

Say an algorithm runs **$0.4N^3 + 25N^2 + 8N + 17$** statements.

- Example:
- N = 1 trillion = 1,000,000,000,000
- N^3 = 1 undecillion = 1,000,000,000,000,000,000,000,000,000,000,000,000

- The constants and lower-order terms don't matter! The highest-order term (N^3) dominates the overall runtime.
- We say that this algorithm runs "on the order of" N^3.
- or **$O(N^3)$** for short ("**Big-Oh** of N cubed")

Big-O

Say an algorithm runs N^3 statements.

- Example:
- N = 1 trillion = 1,000,000,000,000
- N^3 = 1 undecillion = 1,000,000,000,000,000,000,000,000,000,000,000,000

- The constants and lower-order terms don't matter! The highest-order term (N^3) dominates the overall runtime.
- We say that this algorithm runs "on the order of" N^3.
- or $O(N^3)$ for short ("**Big-Oh** of N cubed")

Sum: Big-O

N+2

```
public static int sum1(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum += i;
    }
    return sum;
}
```

O(N)

1

```
public static int sum2(int n) {
    return n * (n + 1) / 2;
}
```

O(1)

Sum: Big-O

```java
// Original sum2 implementation
public static int sum2(int n) {
    return n * (n + 1) / 2;    1
}
```

$O(1)$

```java
// Another sum2 implementation
public static int sum2(int n) {
    int temp1 = n + 1;
    int temp2 = n * temp1;
    int temp3 = temp2 / 2;
    return temp3;
}
```

4

$O(1)$

Poll Everywhere

pollev.com/cse143

What is the Big-O efficiency of this method?

$\cancel{7}N + \cancel{2}$

```
public void method(int n) {
    int value = 0;  1
    for (int i = 0; i < 7; i++) {
        for (int j = 0; j < n; j++) {
            value += j;  1
        }
    }
    return value + n / 2;  1
}
```

7N

N

- O(1)
- $\boxed{O(n)}$
- O(7n)
- O(7n + 4)
- O(n^2)
- O(n^3)

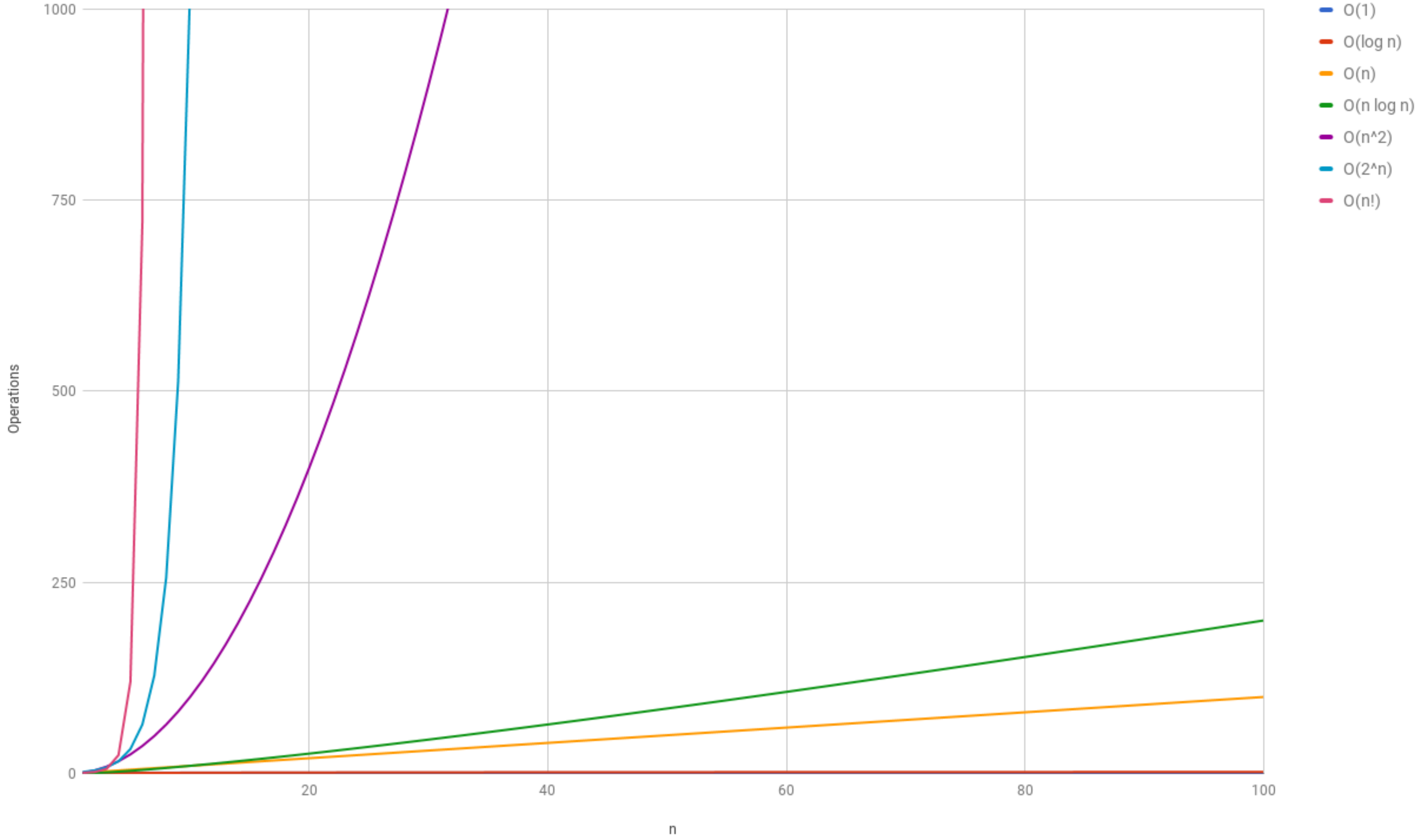
Complexity Classes

Class	**Big-O**	**If you double N…**
constant	O(1)	unchanged
logarithmic	O(log N)	increases slightly
linear	O(N)	doubles
log linear	O(N log N)	slightly more than doubles
quadratic	$O(N^2)$	quadruples
cubic	$O(N^3)$	multiplies by 8
exponential	$O(2^N)$	multiplies <u>drastically</u>

Complexity Comparison

N (input size)	**O(1)**	**O(log N)**	**O(N)**	**O(N log N)**	**$O(N^2)$**	**$O(N^3)$**	**$O(2^N)$**
100	100 ms	100 ms	100 ms	100 ms	100 ms	100 ms	100 ms
200	100 ms	115 ms	200 ms	240 ms	400 ms	800 ms	32.7 sec
400	100 ms	130 ms	400 ms	550 ms	1.6 sec	6.4 sec	12.4 days
800	100 ms	145 ms	800 ms	1.2 sec	6.4 sec	51.2 sec	36.5 million years
1600	100 ms	160 ms	1.6 sec	2.7 sec	25.6 sec	6 min 49.6 sec	$42.1 * 10^{24}$ years
3200	100 ms	175 ms	3.2 sec	6 sec	1 min 42.4 sec	54 min 36 sec	$5.6 * 10^{61}$ years

Visual Complexity Comparison



Sequential Search

- Remember writing `indexOf` in ArrayIntList?
- Sequential search: start at the beginning, examine each element until you find what you want (or reach the end)

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	-4	2	7	10	15	20	22	25	30	36	42	50	56	68	85	92	103

i

Sequential Search

N = size

- What is its complexity class (Big-O)?

O(N)

```
public int indexOf(int value) {
    for (int i = 0; i < size; i++) {
        if (elementData[i] == value) {
            return i;
        }
    }
    return -1;    // not found
}
```

N

What if the array was sorted?

- How could we perform a faster search?

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	-4	2	7	10	15	20	22	25	30	36	42	50	56	68	85	92	103

Binary Search

- *Algorithm:* Examine the middle element of the array.
  - If it is too big, eliminate the right half of the array and repeat.
  - If it is too small, eliminate the left half of the array and repeat.
  - Else it is the value we're searching for, so stop.

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
value	-4	2	7	10	15	20	22	25	30	36	42	50	56	68	85	92	103

Binary Search Time Complexity

- For an array of size N, it eliminates ½ until 1 element remains.
  - N, N/2, N/4, N/8, …, 4, 2, 1
  - How many divisions does it take?

- Think of it from the other direction: How many times do I have to multiply by 2 to reach N?
  - 1, 2, 4, 8, …, N/4, N/2, N
  - Call this number x

$2^x = N$

$x = \log_2 N$

$O(\log N)$

Time Complexity of Collections

`contains(value)`

- ArrayList: O(N)
- LinkedList: O(N)
- TreeSet: O(log N)
- HashSet: O(1)

countUnique – Using a List

N = # words

```
public static int countUnique(Scanner input) {
    List<String> list = new ArrayList<>();
    while (input.hasNext()) {
        String word = input.next();
        if (!list.contains(word)) {
            list.add(word);
        }
    }
    return list.size();
}
```

$1+2+3+\ldots+N$

$= \frac{N(N+1)}{2} = \frac{N^2+\cancel{N}}{\cancel{2}}$

What is the time complexity of this code?

$O(N^2)$

countUnique – Using a Set

N = # words

```
public static int countUnique(Scanner input) {
    Set<String> set = new HashSet<>();
    while (input.hasNext()) {
        String word = input.next();

        set.add(word);
    }
    return set.size();
}
```

N

O(1)

What is the time complexity of this code?

O(N)