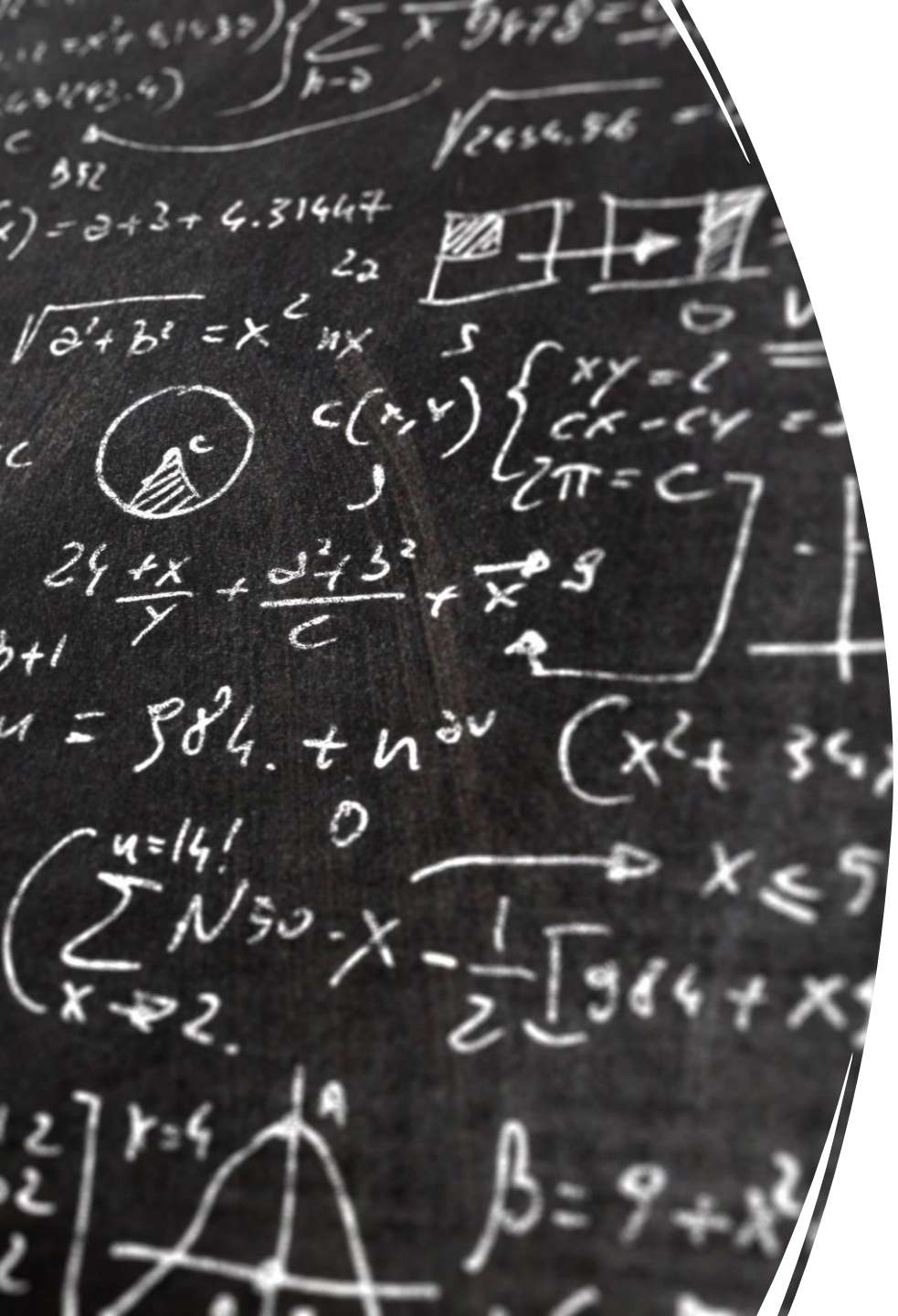


# CS 113 – Computer Science I

# Lecture 18 – OOP & Runtime

Adam Poliak

11/15/2022

# Announcements

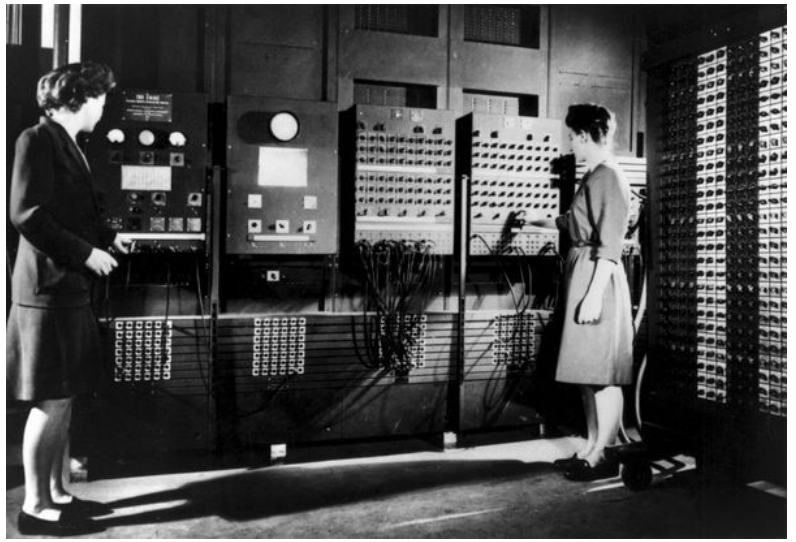
- Assignment 09
  - Due Wednesday 11/23

# Sorting demos - Bank

- Create a Customer class that stores a name (String) and balance (double)
- Create an array of Customers from a CSV (comma separate value) file
  - Each line represents a customer
  - Split each line based on a delimiter character (comma in this case)
- Sort customers based on name
- Sort customers based on balance

# Measuring performance

How do we quantify performance?



# Computing the speed of your programs

Compute the time needed to execute a function

```
import java.lang.System.*;

public static void main(String[] args) {
    …..

    double start = System.currentTimeMillis()/1000.0; // converts to seconds
    bubbleSort(L);
    double end = System.currentTimeMillis()/1000.0;

    System.out.printf("Time: %.10f", (end-start));
}
```

# Runtime analysis

Idea: An algorithm with fewer steps is faster to compute

What is a step?

A single instruction (e.g. assignment, add, etc)

Runtime analysis estimates the number of steps of an algorithm

Runtime analysis helps answer questions such as

How does the performance scale as we increase the input size?
e.g. how long does it take to sort arrays of increasing size?

What is the best case performance? worse case? average case?

# Runtime analysis: Big-O notation

Quantifies worse-case performance

- theoretical measure of how performance changes with input size

Advantages

- Hardware-independent measure

- Allows us to analyze different approaches without implementing the algorithm first

# Big-O Example – Compute a sum

```
int sum = 0;
int i = 0;
while (i < n) {
   sum = sum + i;
   i++;
}
System.out.println(sum);
```

# Big-O counting

- defining, assigning variables (1 step)
- printing, reading input (built-in function calls: “k” steps)
- mul, divide, sub, add, mod, etc (1 step)
- testing conditions (1 step)

# Runtime practice

a)

```
int n = getInputSize();
for (int i = 0; i < n; i++) {
    System.out.println(i);
}
```

# Runtime practice

b)

```
int n = getInputSize();
for (int i = 0; i < 100; i++) {
    System.out.println(i*n);
}
```

# Runtime practice

c)

```
int n = getInputSize();
for (int i = 0; i < n; i++) {
    System.out.println(i);
}

for (int j = 0; j < n; j++) {
   System.out. println(j);
}
```

# Runtime practice

d)

```
int n = getInputSize();
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        System.out.println(i, j);
    }
}
```

# Runtime practice

e)

```
int n = getInputSize();
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        System.out.println(i, j);
    }
}
```

# Runtime practice

h)

```
int[] lst = {1,2,3,5,7,12,19,34,55,67,99,101};
int n = lst.length;
int mid = floor(n/2);
System.out.println(lst[mid]);
```

# Runtime practice

i)

```
int n = getInputSize();
for (int i = 0; i < n; i++) {
  k = n;
  while (k > 1) {
     System.out.println(i, k);
     k = k/2;
  }
}
```

# Runtime practice

f)

```
int n = getInputSize();
for (int i = 0; i < n; i++) {
    for (int j = 0; j < 10; j++) {
        println(i, j);
    }
}
```

# Runtime practice

g)

```
int n = getInputSize();
while (n > 1) {
    println(n);
    n = n/2;
}
```

# Linear Search - revisited

# Binary Search – what is the runtime?

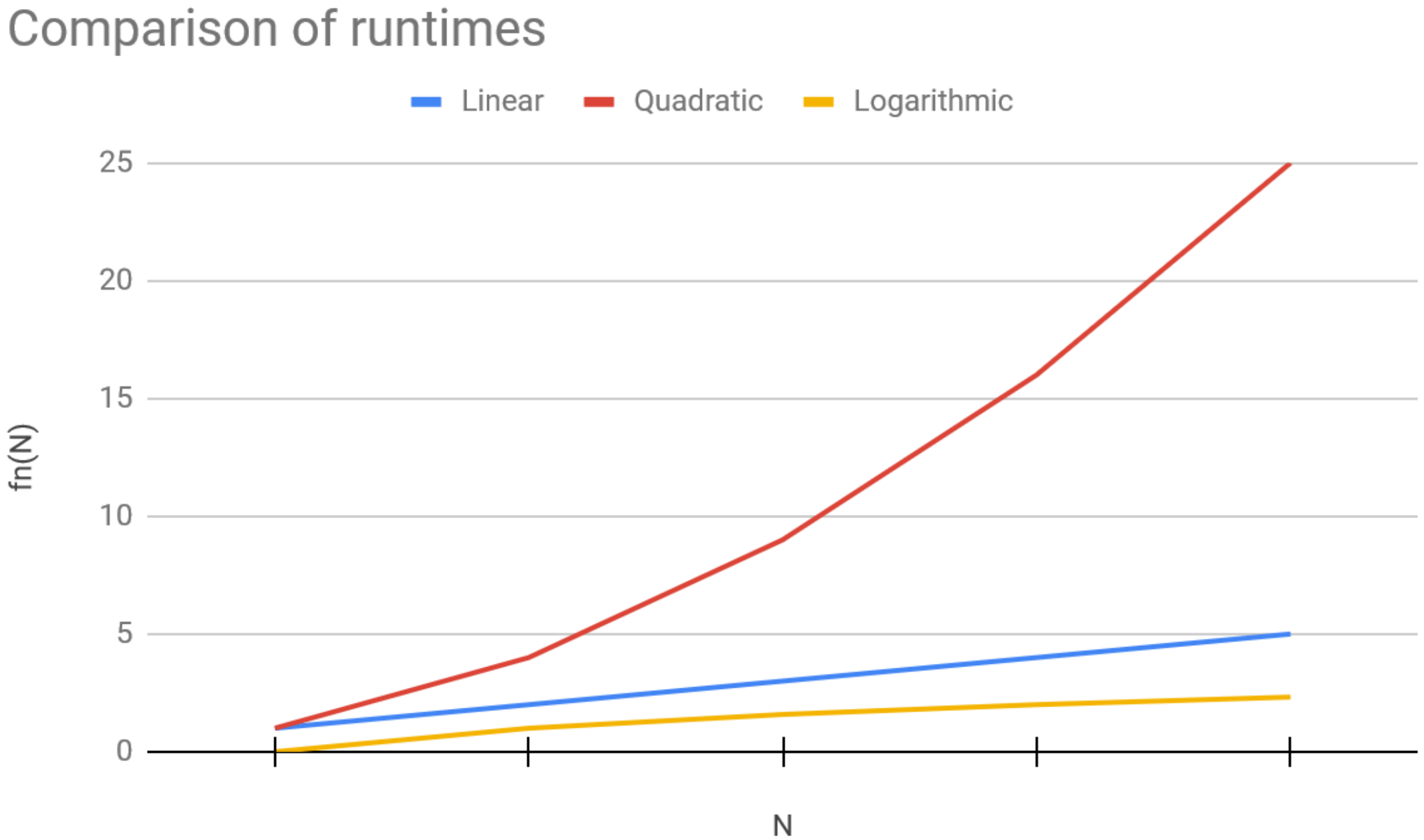
```
public static int search(int x, int[] L) {
    int low = 0;
    int high = L.length-1;
    while (low <= high) {
        int mid = (low + high)/2;
        if (x > L[mid]) {
            low = mid+1;
        }
        else if (x < L[mid]) {
            high = mid-1;
        }
        else {
            return mid;
        }
    }
    return -1;
}
```

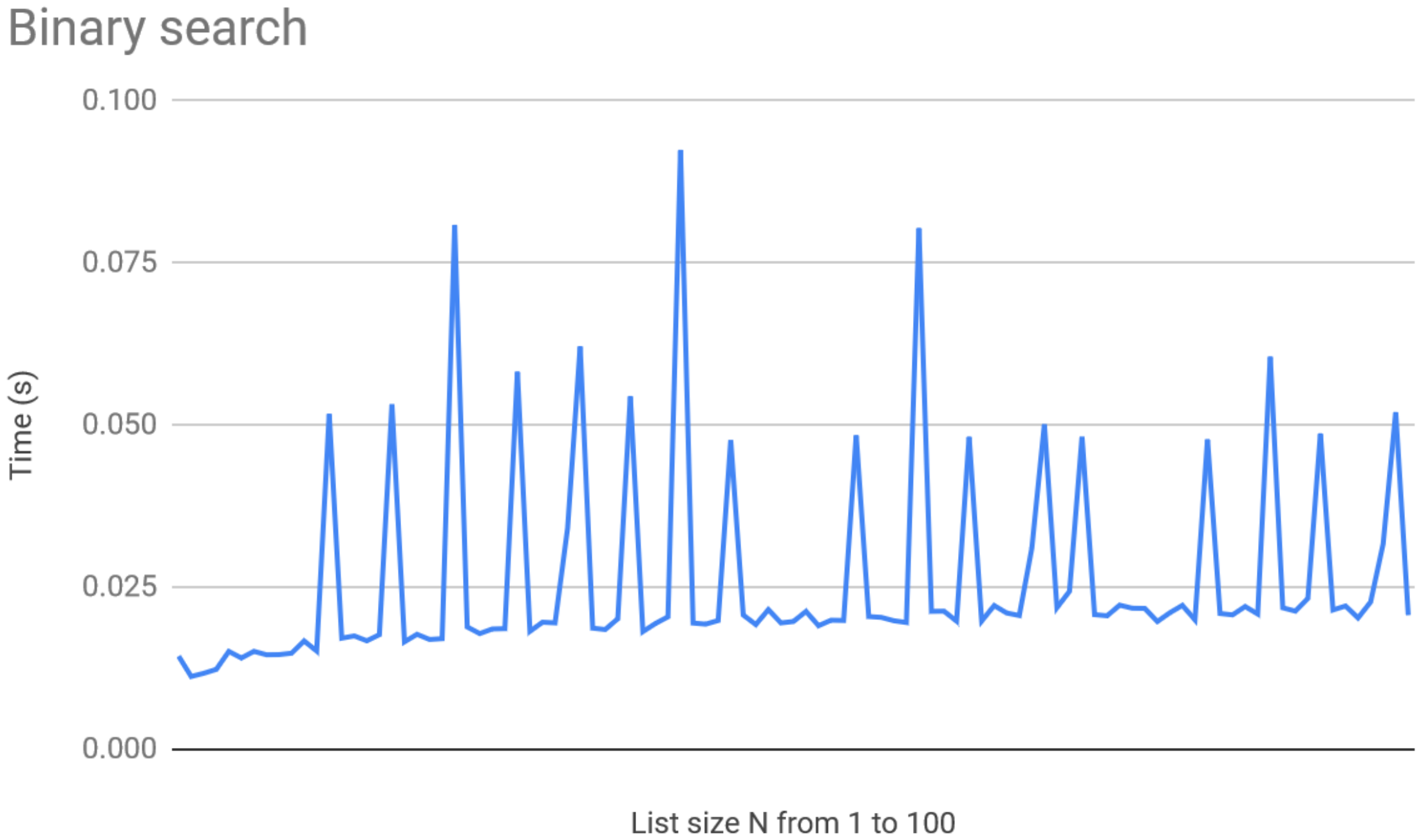
# BubbleSort Revisited – What is the runtime?

```
public static void bubbleSort(int[] L)  {

    for (int n = 0; n < L.length; n++) {
      for (int j = 1; j < L.length-n; j++) {
        if (L[j-1] > L[j]) {
          swap(j-1, j, L);
        }
      }
   }
}
```

# SelectionSort Revisited – What is the runtime?

```
public static void selectionSort(int[] L)   {

  for (int i = 0; i < L.length; i++) {
    int minIdx = i;
    for (int j = i+1; j < L.length; j++) {
      if (L[j] < L[minIdx]) {
        minIdx = j;
      }
    }
    swap(i, minIdx, L);
  }
}
```


Comparison of runtimes
Linear
Quadratic
Logarithmic
25
20
15
10
5
0
fn(N)
N


Binary search
0.100
0.075
0.050
0.025
0.000
Time (s)
List size N from 1 to 100

# Linear Search