Runtime Analysis

Hitesh Boinpally
Summer 2023

Agenda

- Motivation
- Tactics
- Practice

Optimized Code

- We now know lots of ways to write code
- We want some way to determine the most “efficient” implementation

- Efficiency can mean different things
  - Developer time
  - Memory
  - Time

Optimized Code

- We now know lots of ways to write code
- We want some way to determine the most “efficient” implementation

- Efficiency can mean different things
  - Developer time
  - Memory
  - **Time**

Initial Idea: Time the Program

- An intuitive solution – just time the program
- Check time when it starts, run the program, then see what the time is afterwards
  - Compare other programs using this
- Called “wall-clock time”

Initial Idea: Time the Program

- An intuitive solution – just time the program
- Check time when it starts, run the program, then see what the time is afterwards
  - Compare other programs using this
- Called “wall-clock time”

- **Problem: Too many variables in the runtime of a program**
  - “Multi-Tasking”
  - Computer specs
  - Different servers

Next Idea: Count # of Steps

- Rather than worry about precise times, instead we'll focus on "counting steps"
- We won't be too particular in what is and isn't a "step"
  - `x += 5`: could be considered 1 or 3 steps
- Gives us a fast, easy way to compare programs
- Care about how programs perform for **massive inputs**

Next Idea: Count # of Steps

- Rather than worry about precise times, instead we'll focus on "counting steps"
- We won't be too particular in what is and isn't a "step"
  - `x += 5`: could be considered 1 or 3 steps
- Gives us a fast, easy way to compare programs
- Care about how programs perform for **massive inputs**

Example

- Care about how programs perform for **massive inputs**
- What's the runtime for the following program?

```
public static void loopAnalysis(int n) {
    for (int i = 0; i < n; i++) {
        System.out.println("some basic action");
    }
}
```