CS 150: Measuring Program Time

Cynthia Taylor
Oberlin College
April 14th 2013

Recall Timing of 2 Search Functions

- Simple search took several orders of magnitude longer to run than binary search

Measuring Runtime

- We want to understand how the execution time for an algorithm increases when the problem size increases

- One approach: measure the execution time on different-sized problems

Measuring Runtime: The Problem

- Precise runtimes may not be useful as a way to describe algorithm performance
- They depend on the specific machine on which the program is being run
- More useful is an abstract characterization of the rate at which an algorithm's runtime grows

Linear Time

```
def linear(n):
    total = 0
    for i in range(n):
        total += 1
    return total
```

- A linear increase in the problem size leads to a linear increase in the execution time.
- The above function is linear-time. What happens when we call it with 1000000? 2000000? 3000000?

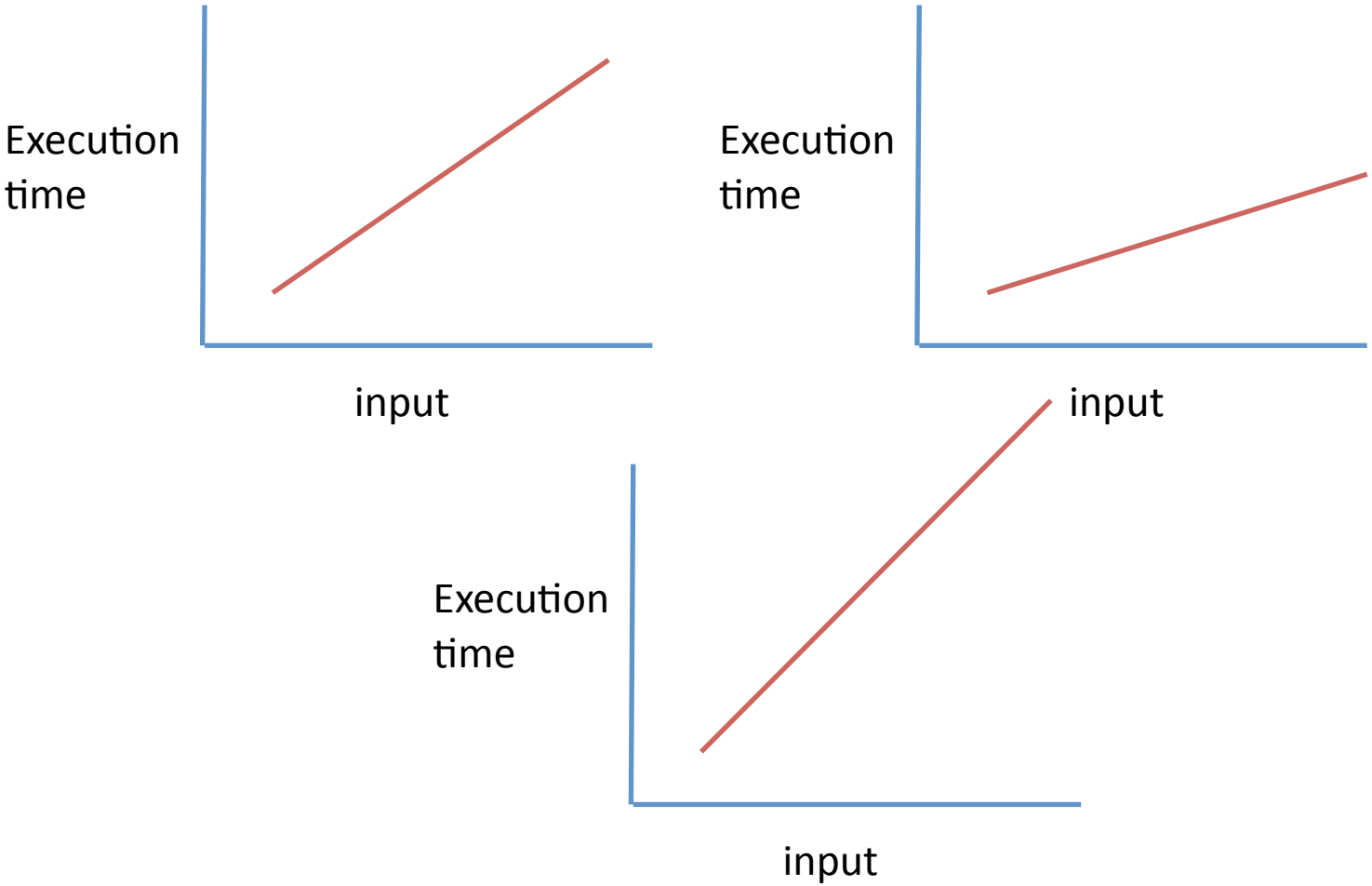
Linear Time

```
def linear(n):
    total = 0
    for i in range(n):
        total += 1
        total += 1
    return total
```

- This function is STILL linear time.

- If we increase the problem size by d (a linear increase), the number of steps always increases by 2d

Linear Time


Execution time
input
Execution time
input
Execution time
input

Is This Function Linear Time?

```
def mystery(n):
    total = 0
    for i in range(n):
        total += 1
    for i in range(n):
        total += 1
    return total
```

A. Yes

B. No

C. I don't know

Quadratic Time

```
def quadratic(n):
    total = 0
    for i in range(n):
        for j in range(n):
            total += 1
    return total
```

- Increasing the problem size by fixed increments causes nonlinear increases in execution time
- For size n, this function performs a number of steps proportional to n^2

Cubic Time

```
def quadratic(n):
    total = 0
    for i in range(n):
        for j in range(n):
            for k in range(n):
            total += 1
    return total
```

- This function is even worse!
- For size n, this function performs a number of steps proportional to n^3

This algorithm is:

```
def mystery(n):
    total = 0
    for i in range(n):
        for j in range(10000):
            for k in range(50):
                total += 1
    return total
```

A. Linear (n)

B. Quadratic (n^2)

C. Cubic (n^3)

D. Something else

E. I don't know

This algorithm is:

```
def mystery(n):
    total = 0
    for i in range(n):
        for j in range(n):
            for k in range(50):
                total += 1
    return total
```

A. Linear (n)

B. Quadratic (n^2)

C. Cubic (n^3)

D. Something else

E. I don't know

This algorithm is:

N=4: 4,2,1,0
N =8: 8,4,2,1,0
N=16: 16, 8,4,2,1,0

log n

```
def f(n):
    sum = 0
    while n > 0:
        sum = sum + n ** 2
        n = n // 2
    return sum
```

A. Better than linear
B. Linear (n)
C. Quadratic (n^2)
D. Worse than quadratic
E. I don't know

Order Notation

- "Big O" notation
- "on the order of" . . .
- Some constant times $\log_2 n$, n, n^2, n^3, 2^n, etc
- Linear O(n), Quadratic O(n^2), Cubic O(n^3)

Your coworker tells you he's found a way to change your algorithm so it does HALF as much work. Knowing that the algorithm is currently $O(n^2)$, you say:

A. That's great. This reduces our runtime to O(n) so it will take less time to run.
B. That's great. Our runtime stays the same, but it will finish in around half the time.
C. That doesn't do anything; our runtime stays the same, so it takes the same amount of time to run.
D. That doesn't do anything; it does reduce our runtime to O(n) but that doesn't mean it runs faster.

The problem size is len(s). This algorithm is:

```
def f(s):
    num = 0
    for c1 in s:
        for c2 in s:
            if c1 == c2:
                num = num + 1
    return num
```

A. Better than linear
B. Linear (n)
C. Quadratic (n^2)
D. Worse than quadratic
E. I don't know

The problem size is len(s). This algorithm is:

```
def f(s):
    num = 0

    for c1 in s:
        s2 = s
        while len(s2) > 0:
            s2 = s2[:len(s2)//2]
```

Out for loop – linear O(n)
While loop – O(log n)

O(n log n)

Next Time

- Sorting!

- Prelab 9 – Wednesday in class