CSE 250

Data Structures

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Lec 04: Intro to Complexity

Announcements and Feedback

- Normal recitations (w/attendance) begin next week
- Academic Integrity Quiz due 9/8 @ 11:59PM **(MUST GET 100%)**
- PA0 due 9/8 @ 11:59PM **(MUST GET 100%)**
  - See @38 on Piazza if you ran into server errors
- WA1 due 9/8 @ 11:59PM

Thought Experiment

An Abstract Data Type is a specification of **what** a data structure can do

Thought Experiment

Often, many data structures can satisfy a given ADT…how do you choose?

Thought Experiment

Data Structure 1

- Very fast `prepend`, `get first`
- Very slow `get nth`

Data Structure 2

- Very fast `get nth`, `get first`
- Very slow `prepend`

Data Structure 3

- Very fast `get nth`, `get first`
- Occasionally slow `prepend`

Which is better?

Thought Experiment

Data Structure 1 (LinkedList)
- Very fast `prepend`, `get first`
- Very slow `get nth`

Data Structure 2 (Array)
- Very fast `get nth`, `get first`
- Very slow `prepend`

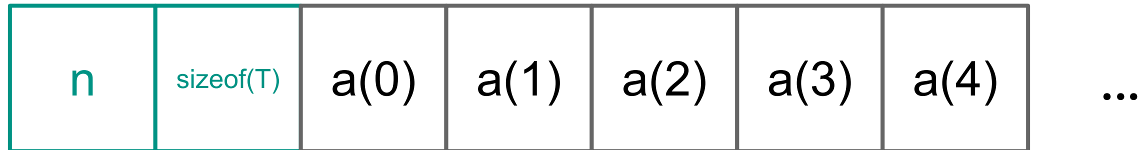
Data Structure 3 (ArrayList…in reverse)
- Very fast `get nth`, `get first`
- Occasionally slow `prepend`

Which is better?

IT DEPENDS!

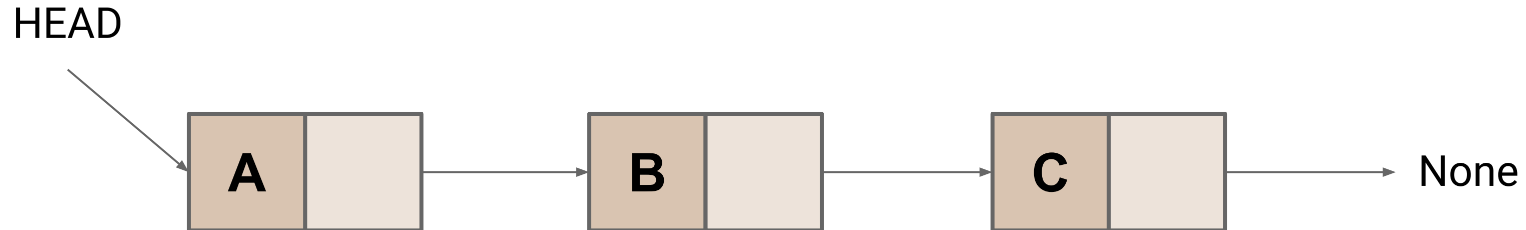
A (very) Brief Refresher: Array

- An array is an ordered container (elements stored one after another)
- Array elements are all stored in a contiguous block of memory

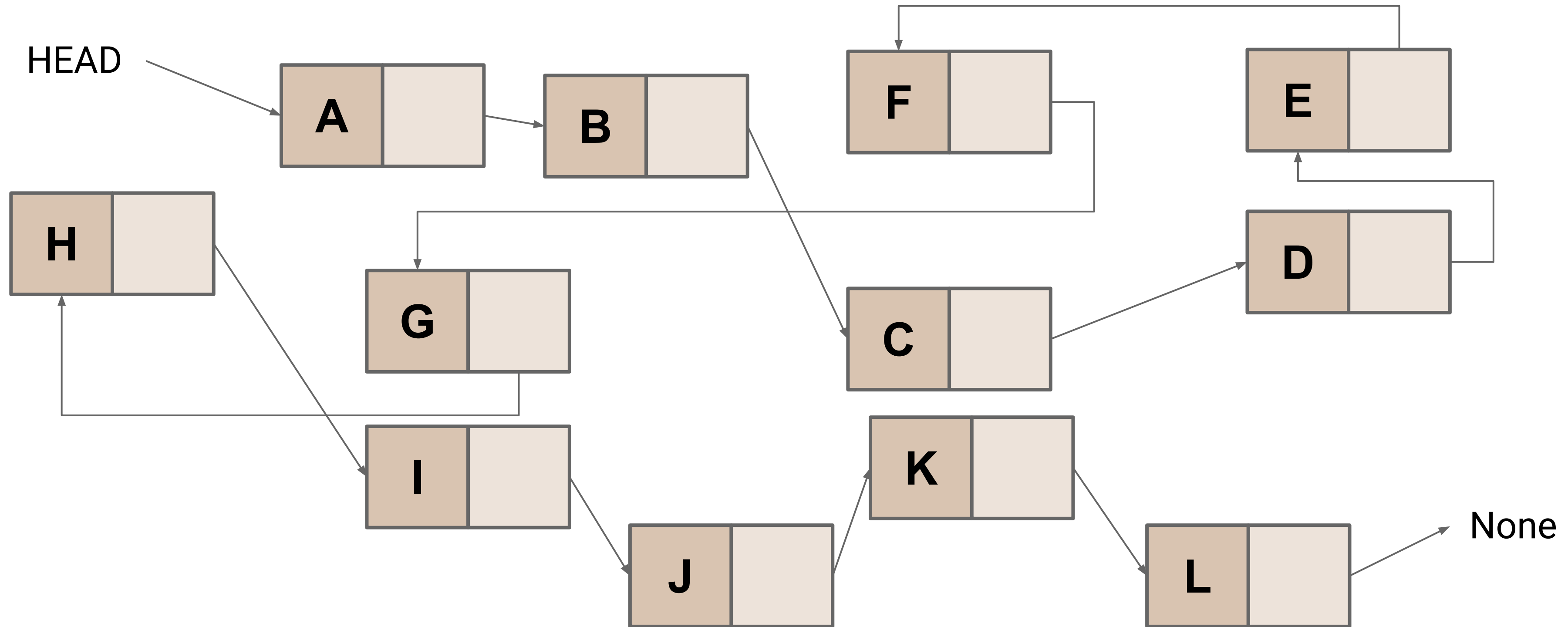


A (very) Brief Refresher: Linked Lists

- Also an ordered container
- Each element stores a pointer to the next element
  - …not necessarily in a contiguous block of memory

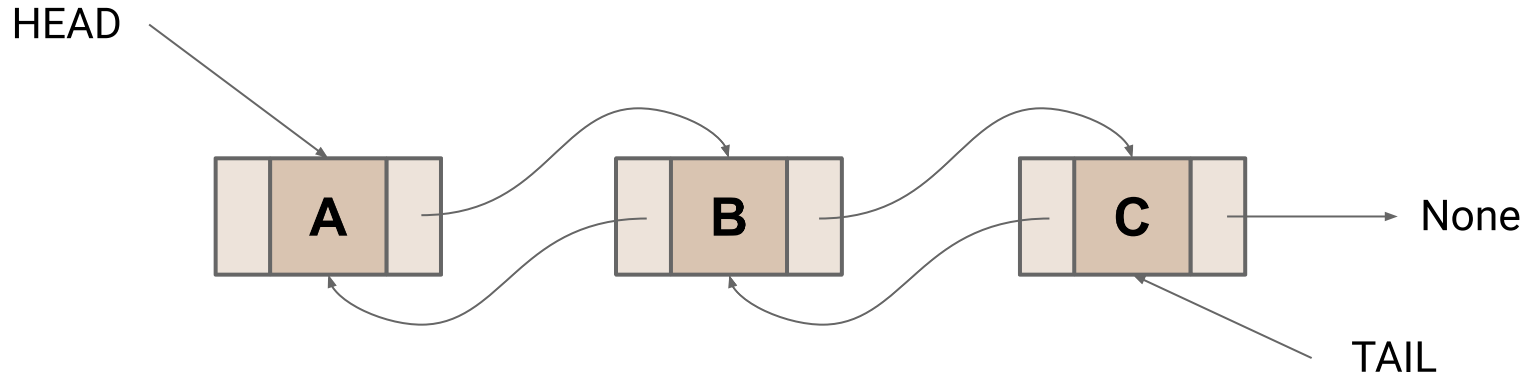


A (very) Brief Refresher: Linked Lists



A (very) Brief Refresher: Linked Lists

- Can also be doubly linked (a next AND a prev pointer per node)
- PA1 will have you implementing a **Sorted Doubly Linked List** with some minor twists



Thought Experiment

Data Structure 1 (LinkedList)
- Very **fast** `prepend`, `get first`
- Very **slow** `get nth`

Data Structure 2 (ArrayList)
- Very **fast** `get nth`, `get first`
- Very **slow** `prepend`

Data Structure 3 (ArrayList...in reverse)
- Very **fast** `get nth`, `get first`
- Occasionally **slow** `prepend`

What is "fast"? "slow"?

Attempt #1: Wall-clock time?

- What is fast?
  - 10s? 100ms? 10ns?
  - …it depends on the task
- Algorithm vs Implementation
  - Compare Grace Hopper's implementation to yours
- What machine are you running on?
  - Your old laptop? A lab machine? The newest, shiniest processor on the market?
- What bottlenecks exist? CPU vs IO vs Memory vs Network…

Attempt #1: Wall-clock time?

- What is fast?
  - 10s? 100ms? 10ns?
  - …it depends on the task
- Algorithm vs Implementation
  - Compare Grace Hopper's implementation to yours
- What machine are you running on?
  - Your old laptop? A lab machine? The newest, shiniest processor on the market?
- What bottlenecks exist? CPU vs IO vs Memory vs Network…

Wall-clock time is not terribly useful…

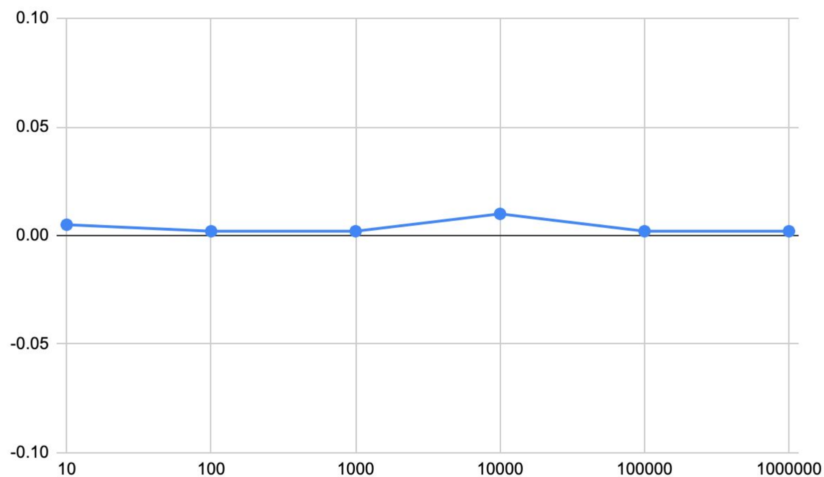
Analysis Checklist

1. Don’t think in terms of wall-time, think in terms of “number of steps”

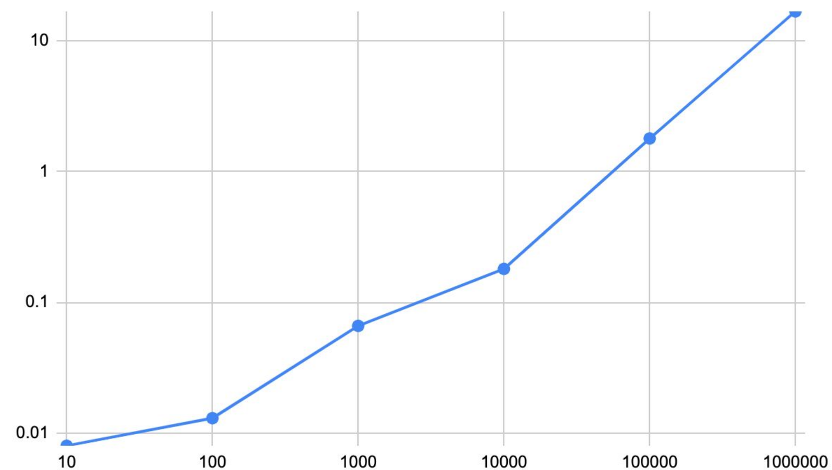
Let’s do a quick demo…

Comparing Random Access for Array vs List

Array

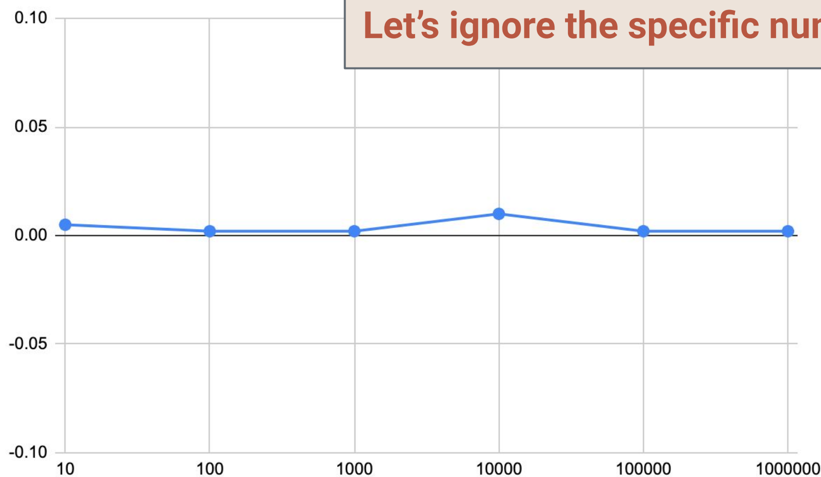


List

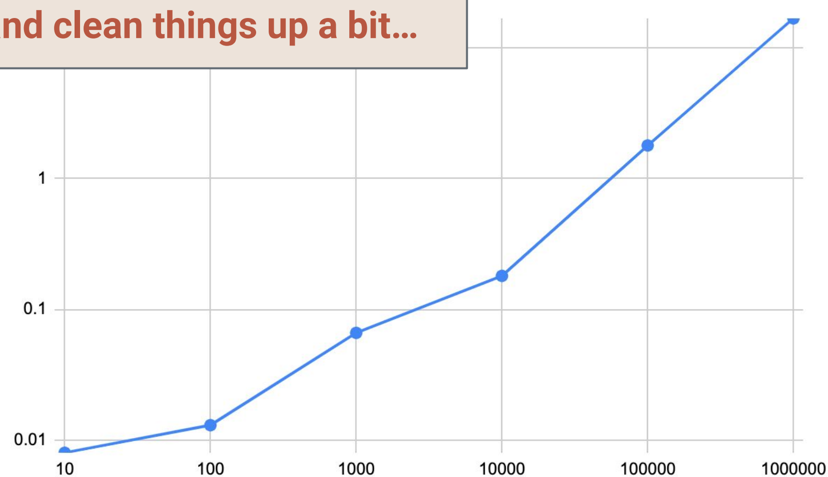


Comparing Random Access for Array vs List

Array



List

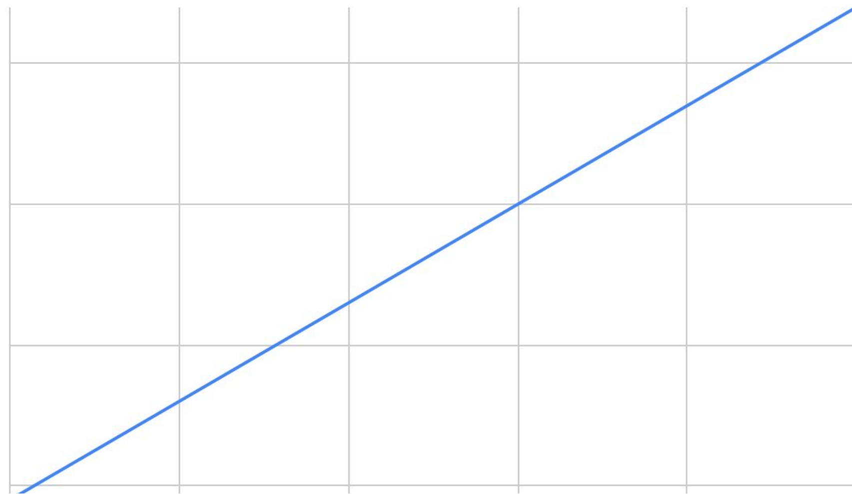


Let's ignore the specific numbers and clean things up a bit…

Comparing Random Access for Array vs List

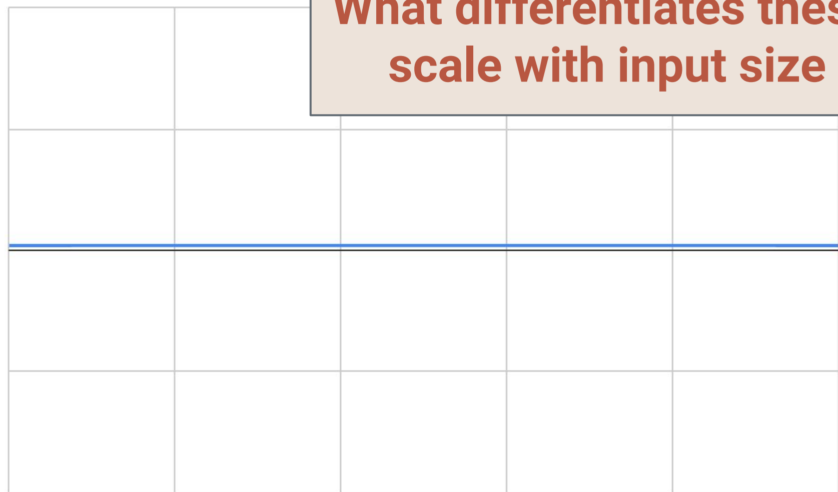
Array

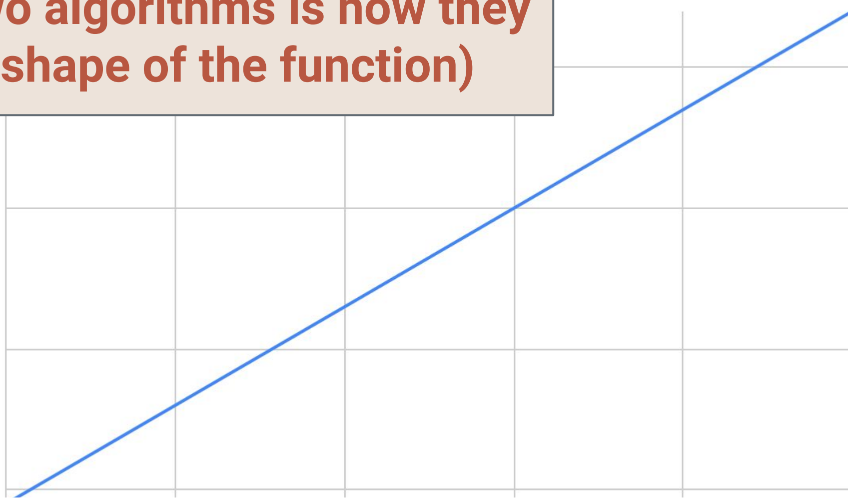
List



Comparing Random Access for Array vs List

Array **List**





What differentiates these two algorithms is how they scale with input size (the shape of the function)

Analysis Checklist

1. Don’t think in terms of wall-time, think in terms of “number of steps”
2. **To give a useful solution, we should take “scale” into account**
    - **How does the runtime change as we change the size of the input?**

Counting Steps

```
1 public void userFullName(User[] users, int id) {
2   User user = users[id];
3   String fullName = user.firstName + user.lastName;
4   return fullName;
5 }
```

Counting Steps

```java
1 public void userFullName(User[] users, int id) {
2   User user = users[id];
3   String fullName = user.firstName + user.lastName;
4   return fullName;
5 }
```

How many steps does this function take?

Counting Steps

```
1 public void userFullName(User[] users, int id) {
2   User user = users[id];
3   String fullName = user.firstName + user.lastName;
4   return fullName;
5 }
```

7 steps…ish? Maybe? What the heck is a step?

Counting Steps

```
1 public void updateUsers(User[] users) {
2   x = 1;
3   for(user : users) {
4     user.id = x;
5     x = x + 1;
6   }
7 }
```

Counting Steps

```
1 public void updateUsers(User[] users) {
2   x = 1;
3   for(user : users) {
4     user.id = x;
5     x = x + 1;
6   }
7 }
```

1

Counting Steps

```
1 public void updateUsers(User[] users) {
2   x = 1;
3   for(User user : users) {
4     user.id = x;
5     x = x + 1;
6   }
7 }
```

$$1 + \sum_{user \in users}$$

Counting Steps

```
1 public void updateUsers(User[] users) {
2   x = 1;
3   for(User user : users) {
4     user.id = x;
5     x = x + 1;
6   }
7 }
```

$$1 + \sum_{user \in users} 4$$

Counting Steps

```
1 public void updateUsers(User[] users) {
2   x = 1;
3   for(User user : users) {
4     user.id = x;
5     x = x + 1;
6   }
7 }
```

$$1 + \sum_{user \in users} 4 = 1 + 4 \cdot |\texttt{users}|$$

Counting Steps

```
1  public void totalReads(User[] users, Post[] posts) {
2    int totalReads = 0;
3    for(Post post : posts) {
4      int userReads = 0;
5      for(User user : users) {
6        if(user.readPost(post)){ userReads += 1; }
7      }
8      totalReads += userReads;
9    }
10 }
```

Counting Steps

```
1  public void totalReads(User[] users, Post[] posts) {
2    int totalReads = 0;
3    for(Post post : posts) {
4      int userReads = 0;
5      for(User user : users) {
6        if(user.readPost(post)){ userReads += 1; }
7      }
8      totalReads += userReads;
9    }
10 }
```

$1 \cdot$

Counting Steps

```
1  public void totalReads(User[] users, Post[] posts) {
2    int totalReads = 0;
3    for(Post post : posts) {
4      int userReads = 0;
5      for(User user : users) {
6        if(user.readPost(post)){ userReads += 1; }
7      }
8      totalReads += userReads;
9    }
10 }
```

$$1 + \sum_{post \in posts}$$

Counting Steps

```
1  public void totalReads(User[] users, Post[] posts) {
2    int totalReads = 0;
3    for(Post post : posts) {
4      int userReads = 0;
5      for(User user : users) {
6        if(user.readPost(post)){ userReads += 1; }
7      }
8      totalReads += userReads;
9    }
10 }
```

$$1 + \sum_{post \in posts} \Bigg(3$$

Counting Steps

```
public void totalReads(User[] users, Post[] posts) {
  int totalReads = 0;
  for(Post post : posts) {
    int userReads = 0;
    for(User user : users) {
      if(user.readPost(post)){ userReads += 1; }
    }
    totalReads += userReads;
  }
}
```

$$1 + \sum_{post \in posts} \left(3 + \sum_{user \in users} \right.$$

Counting Steps

```
1  public void totalReads(User[] users, Post[] posts) {
2    int totalReads = 0;
3    for(Post post : posts) {
4      int userReads = 0;
5      for(User user : users) {
6        if(user.readPost(post)){ userReads += 1; }
7      }
8      totalReads += userReads;
9    }
10 }
```

$$1 + \sum_{post \in posts} \left(3 + \sum_{user \in users} 2 \right)$$

Steps to "Functions"

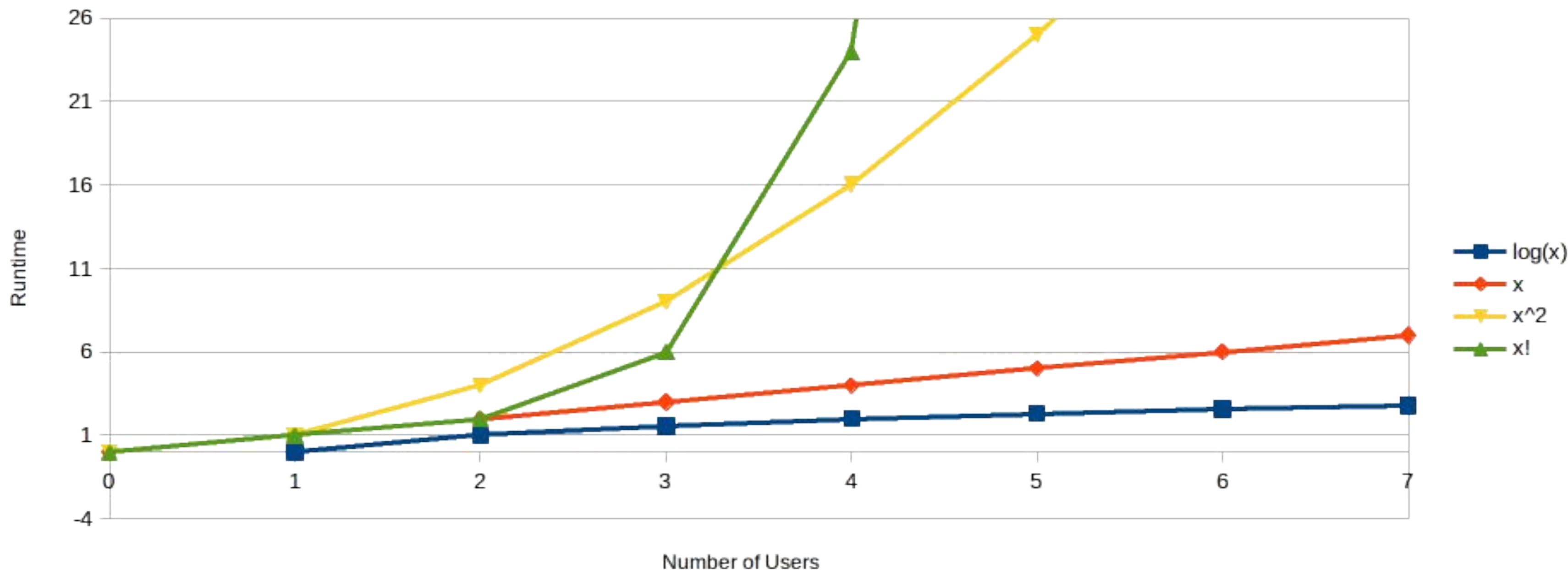
Now that we have number of steps* in terms of summations…

…which we can simplify (like in WA1) into mathematical functions…

We can start analyzing runtime as a function

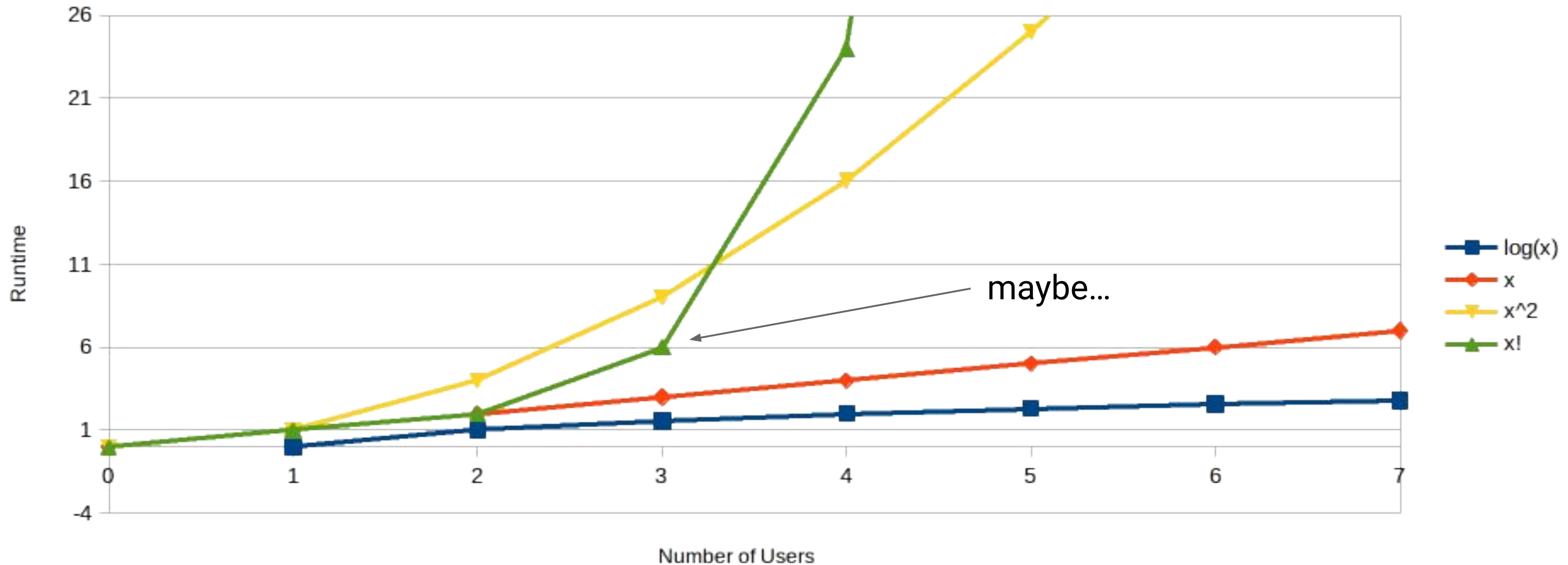
** we'll give a better definition of what a "step" is later*

Runtime as a Function



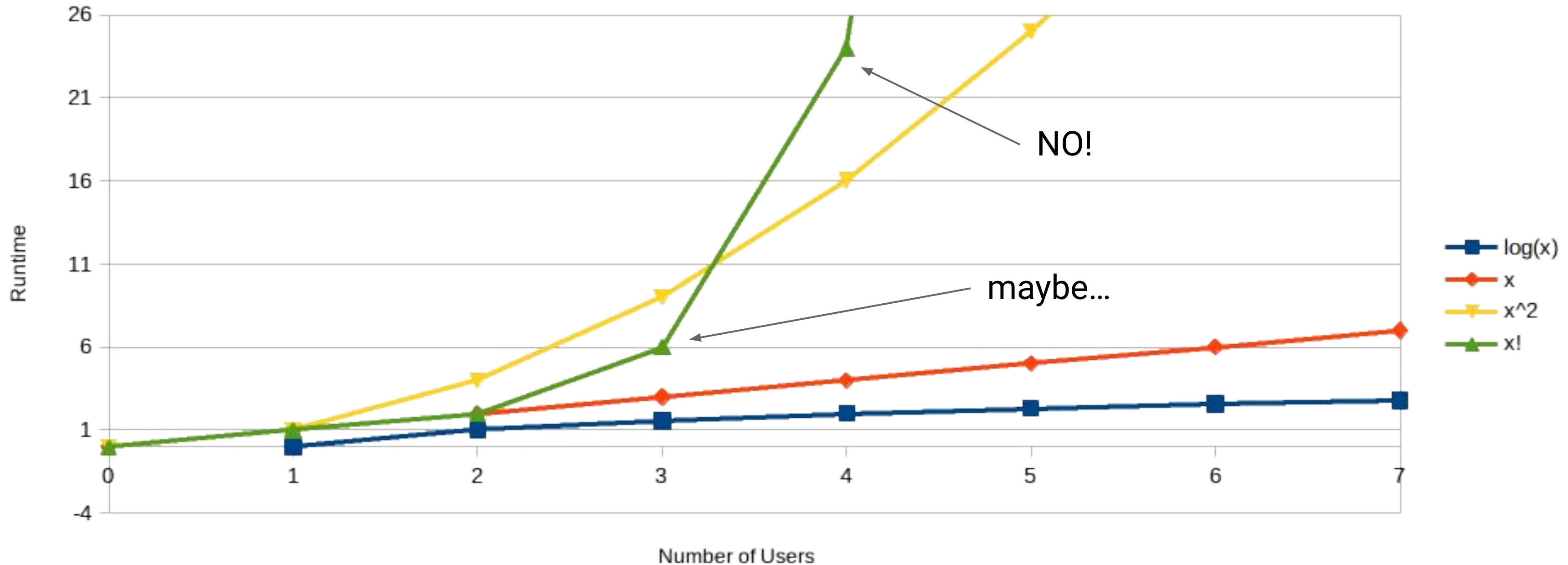
Would you consider an algorithm that takes |Users|! number of steps?

Runtime as a Function



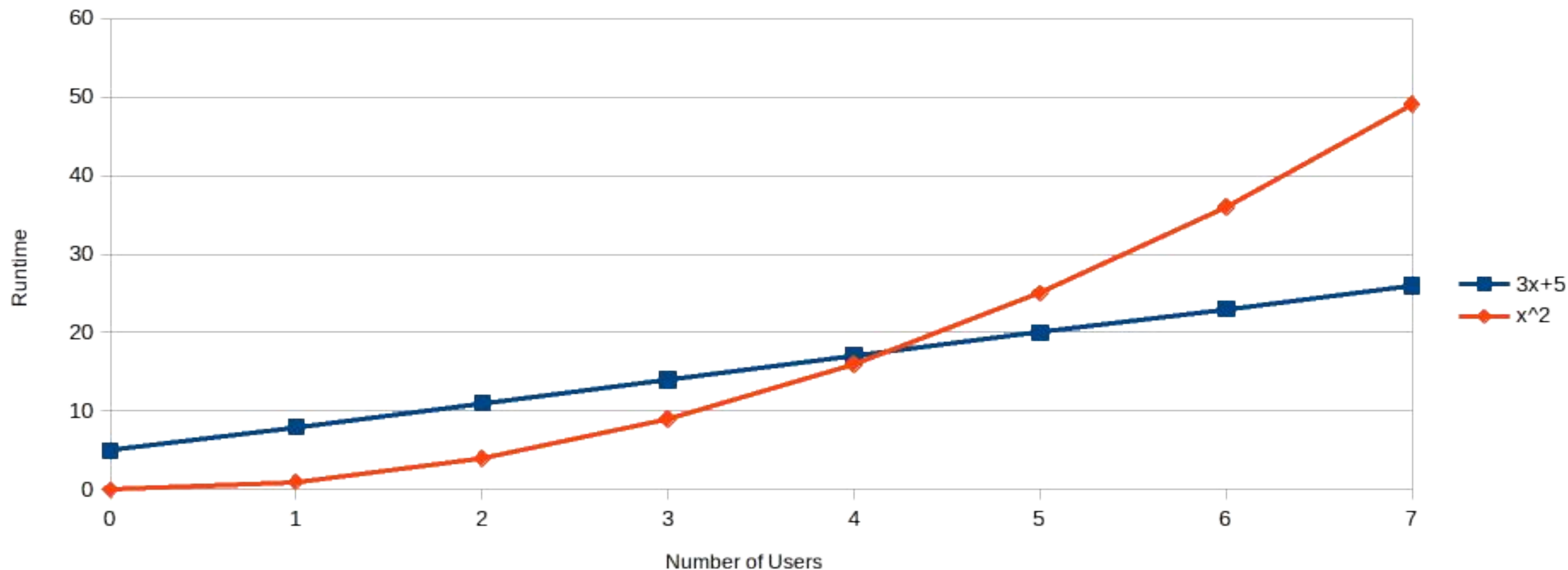
Would you consider an algorithm that takes |Users|! number of steps?

Runtime as a Function



Would you consider an algorithm that takes |Users|! number of steps?

Runtime as a Function



Which is better? 3x|Users|+5 or $|Users|^2$

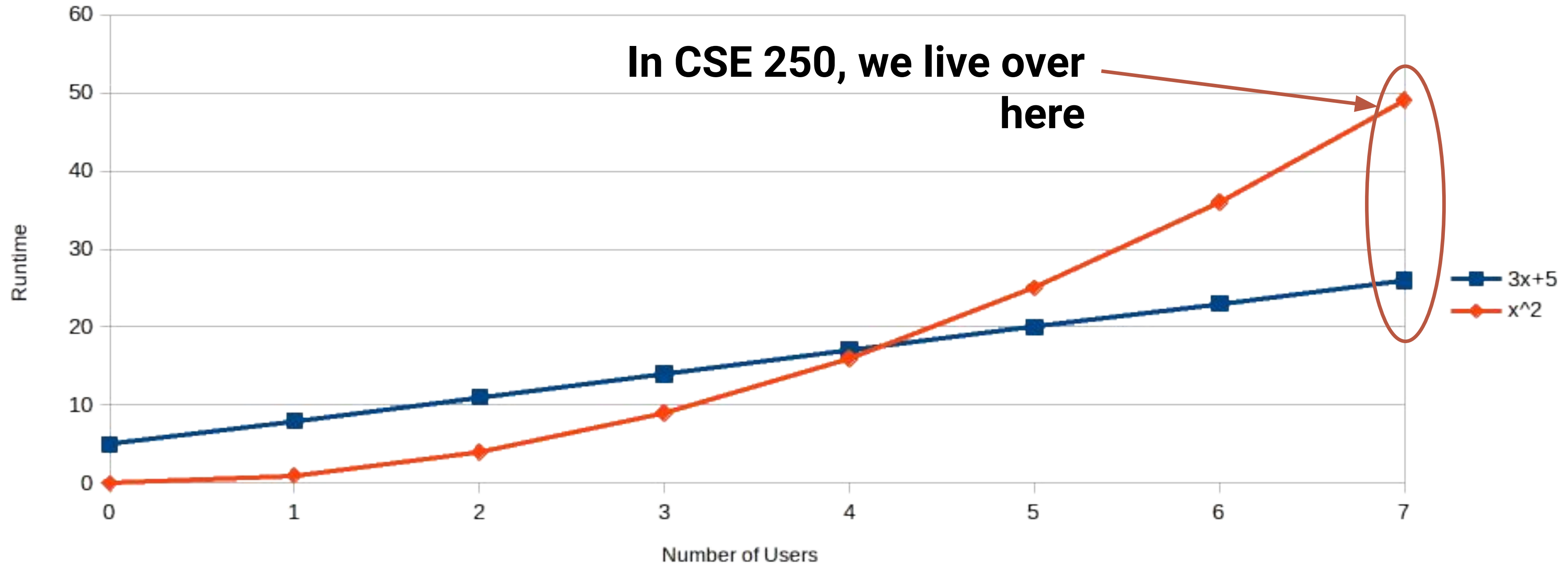
Analysis Checklist

1. Don't think in terms of wall-time, think in terms of "number of steps"
2. To give a useful solution, we should take "scale" into account
    - How does the runtime change as we change the size of the input?

Analysis Checklist

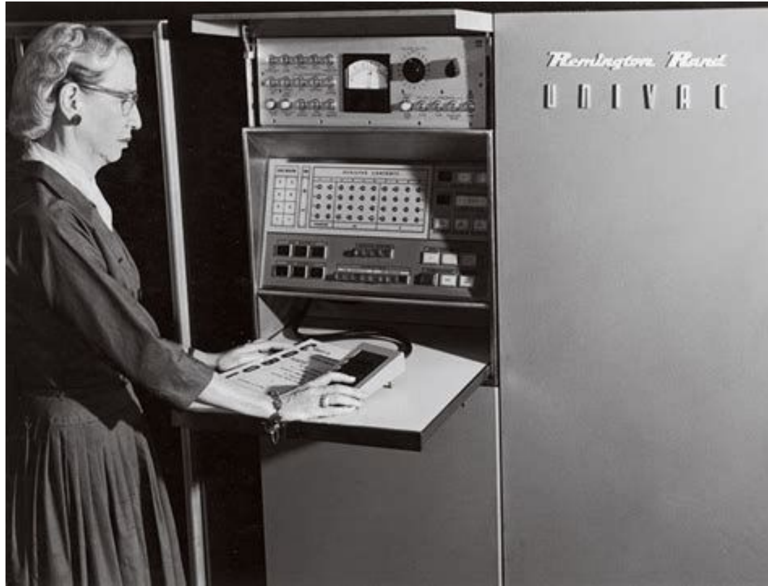
1. Don’t think in terms of wall-time, think in terms of “number of steps”
2. To give a useful solution, we should take “scale” into account
    - How does the runtime change as we change the size of the input?
3. **Focus on “large” inputs**
    - **Rank functions based on how they behave at large scales**

Runtime as a Function



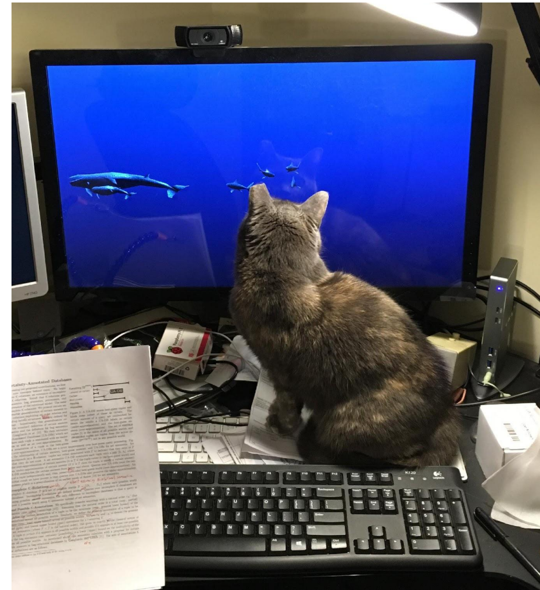
Which is better? 3x|Users|+5 or $|Users|^2$

Goal: Ignore implementation details



Seasoned Pro Implementation

vs

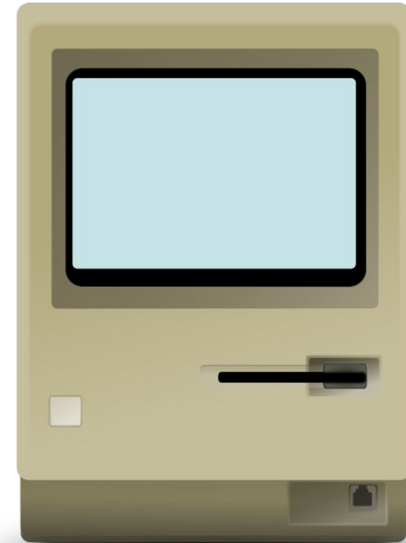


Error 23: Cat on Keyboard

Goal: Ignore execution environment

Intel i9

vs



Motorola 68000

Images from openclipart.org, used with permission

Goal: Judge the Algorithm Itself

- How fast is a step? Don’t care
  - Only count number of steps
- Can this be done in two steps instead of one?
  - “3 steps per user” vs “some number of steps per user”
  - Sometimes we don’t care…sometimes we do

Analysis Checklist

1. Don't think in terms of wall-time, think in terms of "number of steps"
2. To give a useful solution, we should take "scale" into account
    - How does the runtime change as we change the size of the input?
3. Focus on "large" inputs
    - Rank functions based on how they behave at large scales

Analysis Checklist

1. Don't think in terms of wall-time, think in terms of "number of steps"
2. To give a useful solution, we should take "scale" into account
   - How does the runtime change as we change the size of the input?
3. Focus on "large" inputs
   - Rank functions based on how they behave at large scales
4. **Decouple algorithm from infrastructure/implementation**
   - **Asymptotic notation…?**