Welcome

Lectures:
TR 8 - 9:15a White 206

Prof. Marco Alvarez
Gabriel DePace (TA)
Timothy Leonard
Nathan Callanan
Welcome

Lectures:
   TR 8 - 9:15a White 206

Labs:
   W 8 - 9:45a Tyler 55
   W 2 - 3:45p Tyler 55

Prof. Marco Alvarez
Gabriel DePace (TA)
Timothy Leonard
Nathan Callanan
Welcome

Lectures:
  TR 8 - 9:15a White 206

Labs:
  W 8 - 9:45a Tyler 55
  W 2 - 3:45p Tyler 55

My Office Hours:
  F 2 - 3:00p Tyler 257

Prof. Marco Alvarez
Gabriel DePace (TA)
Timothy Leonard
Nathan Callanan
CSC 212?

Review of basic principles of analysis of algorithms
CSC 212?

Review of basic principles of analysis of algorithms

Introduction to fundamental data structures and their algorithms
CSC 212?

Review of basic principles of analysis of algorithms

Introduction to fundamental data structures and their algorithms

arrays, lists, stacks, queues, trees, hash tables, graphs
CSC 212?

Review of basic principles of analysis of algorithms

Introduction to fundamental data structures and their algorithms

arrays, lists, stacks, queues, trees, hash tables, graphs

Survey of classic algorithms for sorting and searching
CSC 212?

Review of basic principles of analysis of algorithms

Introduction to fundamental data structures and their algorithms
  arrays, lists, stacks, queues, trees, hash tables, graphs

Survey of classic algorithms for sorting and searching

Introduction to C/C++ and programming tools
Quick notes about C++

CSC 212 is not about learning a new language however we will offer a C++ boot camp
Quick notes about C++

CSC 212 is not about learning a new language however we will offer a C++ boot camp

Recommended IDEs (check Piazza for more resources)

- Xcode (os x)
- Visual Studio (windows)
- vim + gcc + gdb (superpowers)
Textbook(s)
Grading

Exams (55%)

1 midterm (25%)
1 final (30%)
Grading

Exams (55%)
  1 midterm (25%)
  1 final (30%)

Assignments (45%)
  4 programming assignments (25%)
  6 problem sets (20%)
Grading

Exams (55%)
   1 midterm (25%)
   1 final (30%)

Assignments (45%)
   4 programming assignments (25%)
   6 problem sets (20%)

if your attendance to Lab sessions is < 70%, you get an automatic 0 for all programming assignments!
Resources

Autolab
Piazza
Course Website

Office Hours
Tutoring / Boot Camp
Computer Labs
Your Part

Attendance **required** for lectures and labs
Your Part

Attendance required for lectures and labs participate!
Your Part

Attendance **required** for lectures and labs
participate!

Work Hard
Your Part

Attendance
  required for lectures and labs
  participate!

Work Hard
  read the textbook and recommended readings
Your Part

Attendance

required for lectures and labs
participate!

Work Hard

read the textbook and recommended readings
be honest (your sole goal is to learn)
Your Part

Attendance

required for lectures and labs

participate!

Work Hard

read the textbook and recommended readings

be honest (your sole goal is to learn)

ask for help when needed
Your Part

Attendance
- **required** for lectures and labs
- **participate**!

Work Hard
- read the textbook and recommended readings
- be **honest** (your sole goal is to **learn**)
- ask for help when needed
- submit assignments **on time**!
Stuck? Lost? Hopeless?

try finding answers on your own (textbook, readings, online forums, etc.)
Stuck? Lost? Hopeless?

try finding answers on your own (textbook, readings, online forums, etc.)

post a question/note on Piazza
Stuck? Lost? Hopeless?

try finding answers on your own (textbook, readings, online forums, etc.)

post a question/note on **Piazza**

contact your TA
Stuck? Lost? Hopeless?

try finding answers on your own (textbook, readings, online forums, etc.)

post a question/note on Piazza

contact your TA

come to office hours
ENIAC lost one vacuum tube roughly every day or two. With almost 18,000 tubes, locating and replacing the failed one was challenging ....

ENIAC lost one vacuum tube roughly every day or two. With almost 18,000 tubes, locating and replacing the failed one was challenging ...

ENIAC lost one vacuum tube roughly every day or two. With almost 18,000 tubes, locating and replacing the failed one was challenging.

ENIAC lost one vacuum tube roughly every day or two. With almost 18,000 tubes, locating and replacing the failed one was challenging...
ENIAC lost one vacuum tube roughly every day or two. With almost 18,000 tubes, locating and replacing the failed one was challenging ....

ENIAC lost one vacuum tube roughly every day or two. With almost 18,000 tubes, locating and replacing the failed one was challenging.

ENIAC lost one vacuum tube roughly every day or two. With almost 18,000 tubes, locating and replacing the failed one was challenging.

ENIAC lost one vacuum tube roughly every day or two. With almost 18,000 tubes, locating and replacing the failed one was challenging.

Common Design

Diagram showing the common design of a computer system with components connected to the CPU, memory, data, and information.
Beyond a Great Programmer

Problem Solving Skills
Beyond a Great Programmer

Problem Solving Skills

Language Skills

Q: What makes a programmer?

- problem solving skillz
- nunchuck skills
- research skills
- caffeine addiction
  ...passion ftw!

www.netneeti.com
Beyond a Great Programmer

Problem Solving Skills
Language Skills
CS Core Knowledge

Q: What makes a programmer?

- problem solving skillzz
- nunchack skills
- research skills
- caffeine addiction
...passion ftw!

www.netneedi.com
Beyond a Great Programmer

Problem Solving Skills
Language Skills
CS Core Knowledge
Discipline
testing, coding style
Let's get started ...
Understand and Model the Problem
Understand and Model the Problem

Design a Solution
Understand and Model the Problem

Design a Solution

Correct? Efficient? Scalable?
Understand and Model the Problem

Design a Solution

Correct? Efficient? Scalable?

why not?

NO
Understand and Model the Problem

Design a Solution

Correct? Efficient? Scalable?

why not?

NO

YES
Union-Find problem

Data structure to represent a collection of disjoint sets
Union-Find problem

Data structure to represent a collection of disjoint sets

makeSet(e) create a singleton set for the element e
Union-Find problem

Data structure to represent a collection of disjoint sets

- **makeSet(e)**: create a singleton set for the element $e$
- **union(a, b)**: return the set $a \cup b$
Union-Find problem

Data structure to represent a collection of disjoint sets

- makeSet(e) create a singleton set for the element e
- union(a, b) return the set a U b
- find(e) return the set containing the element e
16 calls to makeset(…)
union(0, 1)
union(5, 6)
union(6, 10)
union(3, 7)
union(7, 11)
16 calls to makeset(…)
union(0, 1)
union(5, 6)
union(6, 10)
union(3, 7)
union(7, 11)
union(10, 11)
16 calls to makeset(…)
union(0, 1)
union(5, 6)
union(6, 10)
union(3, 7)
union(7, 11)
union(10, 11)
16 calls to makeset(…)
union(0, 1)
union(5, 6)
union(6, 10)
union(3, 7)
union(7, 11)
union(10, 11)
union(14, 13)
16 calls to makeset(…)
union(0, 1)
union(5, 6)
union(6, 10)
union(3, 7)
union(7, 11)
union(10, 11)
union(14, 13)
16 calls to makeset(…)
union(0, 1)
union(5, 6)
union(6, 10)
union(3, 7)
union(7, 11)
union(10, 11)
union(14, 13)
// are 6 and 3 connected
find(6) == find(3)?
16 calls to `makeset(…)`

- `union(0, 1)`
- `union(5, 6)`
- `union(6, 10)`
- `union(3, 7)`
- `union(7, 11)`
- `union(10, 11)`
- `union(14, 13)`

// are 6 and 3 connected
`find(6) == find(3)?` 👍
16 calls to makeset(…)
union(0, 1)
union(5, 6)
union(6, 10)
union(3, 7)
union(7, 11)
union(10, 11)
union(14, 13)

// are 6 and 3 connected
find(6) == find(3)? 👍

// are 4 and 8 connected
find(4) == find(8)?
16 calls to `makeset(...)`
union(0, 1)
union(5, 6)
union(6, 10)
union(3, 7)
union(7, 11)
union(10, 11)
union(14, 13)

// are 6 and 3 connected
`find(6) == find(3)?` 👍

// are 4 and 8 connected
`find(4) == find(8)?` 👎
16 calls to makeset(…)
union(0, 1)
union(5, 6)
union(6, 10)
union(3, 7)
union(7, 11)
union(10, 11)
union(14, 13)

// are 6 and 3 connected
find(6) == find(3)? 👍

// are 4 and 8 connected
find(4) == find(8)? 👎
union(8, 9)
16 calls to `makeset(…)`
union(0, 1)
union(5, 6)
union(6, 10)
union(3, 7)
union(7, 11)
union(10, 11)
union(14, 13)

// are 6 and 3 connected
find(6) == find(3)? 👍

// are 4 and 8 connected
find(4) == find(8)? 👎

union(8, 9)
16 calls to makeset(…)
union(0, 1)
union(5, 6)
union(6, 10)
union(3, 7)
union(7, 11)
union(10, 11)
union(14, 13)

// are 6 and 3 connected
find(6) == find(3)? 👍

// are 4 and 8 connected
find(4) == find(8)? 👎
union(8, 9)
union(8, 4)
16 calls to `makeset(...)`
union(0, 1)
union(5, 6)
union(6, 10)
union(3, 7)
union(7, 11)
union(10, 11)
union(14, 13)

// are 6 and 3 connected
`find(6) == find(3)?` 👍

// are 4 and 8 connected
`find(4) == find(8)?` 👎

union(8, 9)
union(8, 4)
16 calls to makeset(…)
union(0, 1)
union(5, 6)
union(6, 10)
union(3, 7)
union(7, 11)
union(10, 11)
union(14, 13)
// are 6 and 3 connected
find(6) == find(3)? 👍
// are 4 and 8 connected
find(4) == find(8)? 👎
union(8, 9)
union(8, 4)
find(4) == find(9)?
16 calls to makeset(…)
union(0, 1)
union(5, 6)
union(6, 10)
union(3, 7)
union(7, 11)
union(10, 11)
union(14, 13)
// are 6 and 3 connected
find(6) == find(3)? 😊
// are 4 and 8 connected
find(4) == find(8)? 😊
union(8, 9)
union(8, 4)
find(4) == find(9)? 😊
Implementing a quick find
Implementing a quick find

Data Structure
Implementing a quick **find**

**Data Structure**

integer array **id** of length **n**
Implementing a quick find

Data Structure

integer array `id` of length `n`

`id[i]` is an identifier for the set containing `i`
Implementing a quick **find**

**Data Structure**

integer array \textbf{id} of length \textbf{n}

\textbf{id}[i] is an identifier for the set containing \textbf{i}
Implementing a quick find

How to implement makeset(i)?
Implementing a quick find

How to implement `makeset(i)`?

trivial, \( id[i] = i \)
Implementing a quick `find`

How to implement `makeset(i)`?

trivial, \( \text{id}[i] = i \)

How to implement `find(i)`?
Implementing a quick **find**

How to implement `makeset(i)`?
trivial, `id[i] = i`

How to implement `find(i)`?
trivial, `return id[i]`
Implementing a quick `find`

How to implement `makeset(i)`?
trivial, \( \text{id}[i] = i \)

How to implement `find(i)`?
trivial, return \( \text{id}[i] \)

How to implement `union(a, b)`?
Implementing a quick \textbf{find}

How to implement \texttt{makeset(i)}?  
trivial, \( \text{id}[i] = i \)

How to implement \texttt{find(i)}?  
trivial, \( \text{return id}[i] \)

How to implement \texttt{union(a, b)}?  
change all values \( \text{id}[a] \) to \( \text{id}[b] \), or vice-versa
void QuickFind::makeset(i) {
    id[i] = i;
}

int QuickFind::find(i) {
    return id[i];
}

void QuickFind::union(int a, int b) {
    for (int i = 0 ; i < n ; i++) {
        if (id[i] == id[a]) {
            id[i] = id[b];
        }
    }
}
void QuickFind::makeset(i) {
    id[i] = i; // 1 write access (constant time)
}

int QuickFind::find(i) {
    return id[i];
}

void QuickFind::union(int a, int b) {
    for (int i = 0 ; i < n ; i++) {
        if (id[i] == id[a]) {
            id[i] = id[b];
        }
    }
}
```cpp
void QuickFind::makeSet(i) {
    id[i] = i;  // 1 write access (constant time)
}

int QuickFind::find(i) {
    return id[i];  // 1 read access (constant time)
}

void QuickFind::union(int a, int b) {
    for (int i = 0; i < n; i++) {
        if (id[i] == id[a]) {
            id[i] = id[b];
        }
    }
}
```
```c
void QuickFind::makeset(i) {
    id[i] = i;
}

int QuickFind::find(i) {
    return id[i];
}

void QuickFind::union(int a, int b) {
    for (int i = 0; i < n; i++) {
        if (id[i] == id[a]) {
            id[i] = id[b];
        }
    }
}
```
void QuickFind::makeset(i) {
    id[i] = i;  // 1 write access (constant time)
}

int QuickFind::find(i) {
    return id[i];  // 1 read access (constant time)
}

void QuickFind::union(int a, int b) {
    for (int i = 0; i < n; i++) {
        if (id[i] == id[a]) {
            id[i] = id[b];  // 2n read accesses (linear time)
        }
    }
    // <=n read accesses (linear time)
}
void QuickFind::makeset(i) {
    id[i] = i;
}

int QuickFind::find(i) {
    return id[i];
}

void QuickFind::union(int a, int b) {
    for (int i = 0 ; i < n ; i++) {
        if (id[i] == id[a]) {
            id[i] = id[b];
        }
    }
}

1 write access (constant time)
1 read access (constant time)
2n read accesses (linear time)
<=n read accesses (linear time)
<=n write accesses (linear time)
## Computational Cost

<table>
<thead>
<tr>
<th></th>
<th>makeset</th>
<th>find</th>
<th>union</th>
</tr>
</thead>
<tbody>
<tr>
<td>quick-find</td>
<td>1</td>
<td>1</td>
<td>n</td>
</tr>
</tbody>
</table>

ignore leading constants
Computational Cost

<table>
<thead>
<tr>
<th></th>
<th>makeset</th>
<th>find</th>
<th>union</th>
</tr>
</thead>
<tbody>
<tr>
<td>quick-find</td>
<td>1</td>
<td>1</td>
<td>n</td>
</tr>
</tbody>
</table>

Creating a problem with $n$ elements takes $n$ accesses (\(n\) calls to \texttt{makeset})
Computational Cost

<table>
<thead>
<tr>
<th></th>
<th>makeset</th>
<th>find</th>
<th>union</th>
</tr>
</thead>
<tbody>
<tr>
<td>quick-find</td>
<td>1</td>
<td>1</td>
<td>$n$</td>
</tr>
</tbody>
</table>

Creating a problem with $n$ elements takes $n$ accesses ($n$ calls to makeset)

Processing $m$ union operations takes $mn$ accesses ($m$ calls to union)

ignore leading constants
Computational Cost

<table>
<thead>
<tr>
<th></th>
<th>makeset</th>
<th>find</th>
<th>union</th>
</tr>
</thead>
<tbody>
<tr>
<td>quick-find</td>
<td>1</td>
<td>1</td>
<td>n</td>
</tr>
</tbody>
</table>

Creating a problem with \( n \) elements takes \( n \) accesses (\( n \) calls to makeset).

Processing \( m \) union operations takes \( m \times n \) accesses (\( m \) calls to union).

If \( m \approx n \), quadratic time (\( n^2 \) accesses).!!
What is wrong with quadratic?

<table>
<thead>
<tr>
<th>Size of Input</th>
<th>N</th>
<th>N^2</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = 10</td>
<td>10</td>
<td>100</td>
</tr>
</tbody>
</table>
What is wrong with quadratic?

<table>
<thead>
<tr>
<th>Size of Input</th>
<th>N</th>
<th>$N^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 10$</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>$n = 100$</td>
<td>100</td>
<td>10.000</td>
</tr>
</tbody>
</table>
What is wrong with quadratic?

<table>
<thead>
<tr>
<th>Size of Input</th>
<th>N</th>
<th>$N^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = 10</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>n = 100</td>
<td>100</td>
<td>10.000</td>
</tr>
<tr>
<td>n = 1000</td>
<td>1000</td>
<td>1.000.000</td>
</tr>
</tbody>
</table>
What is wrong with quadratic?

<table>
<thead>
<tr>
<th>Size of Input</th>
<th>N</th>
<th>N^2</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = 10</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>n = 100</td>
<td>100</td>
<td>10.000</td>
</tr>
<tr>
<td>n = 1000</td>
<td>1.000</td>
<td>1.000.000</td>
</tr>
<tr>
<td>n = 10000</td>
<td>10.000</td>
<td>100.000.000</td>
</tr>
</tbody>
</table>
What is wrong with quadratic?

<table>
<thead>
<tr>
<th>Size of Input</th>
<th>N</th>
<th>N²</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = 10</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>n = 100</td>
<td>100</td>
<td>10.000</td>
</tr>
<tr>
<td>n = 1000</td>
<td>1.000</td>
<td>1.000.000</td>
</tr>
<tr>
<td>n = 10000</td>
<td>10.000</td>
<td>100.000.000</td>
</tr>
<tr>
<td>n = 100000</td>
<td>100.000</td>
<td>10.000.000.000</td>
</tr>
</tbody>
</table>
What is wrong with quadratic?

<table>
<thead>
<tr>
<th>Size of Input</th>
<th>N</th>
<th>N²</th>
</tr>
</thead>
<tbody>
<tr>
<td>n = 10</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>n = 100</td>
<td>100</td>
<td>10.000</td>
</tr>
<tr>
<td>n = 1000</td>
<td>1.000</td>
<td>1.000.000</td>
</tr>
<tr>
<td>n = 10000</td>
<td>10.000</td>
<td>100.000.000</td>
</tr>
<tr>
<td>n = 100000</td>
<td>100.000</td>
<td>10.000.000.000</td>
</tr>
<tr>
<td>n = 1000000</td>
<td>1.000.000</td>
<td>1.000.000.000.000</td>
</tr>
</tbody>
</table>
What is wrong with quadratic?

<table>
<thead>
<tr>
<th>Size of Input</th>
<th>N</th>
<th>$N^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 10$</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>$n = 100$</td>
<td>100</td>
<td>10.000</td>
</tr>
<tr>
<td>$n = 1000$</td>
<td>1.000</td>
<td>1.000.000</td>
</tr>
<tr>
<td>$n = 10000$</td>
<td>10.000</td>
<td>100.000.000</td>
</tr>
<tr>
<td>$n = 100000$</td>
<td>100.000</td>
<td>10.000.000.000</td>
</tr>
<tr>
<td>$n = 1000000$</td>
<td>1.000.000</td>
<td>1.000.000.000.000</td>
</tr>
<tr>
<td>$n = 10000000$</td>
<td>10.000.000</td>
<td>100.000.000.000.000</td>
</tr>
</tbody>
</table>
Graph is big and changing

1 billion people
240 billion photos
1 trillion connections