Public-Service Announcement I

tion for Introduction to Mathematical Thinking, a 2-
hat aims to develop mathematical maturity and pre-
s for CS 70, is now live!
/apply.imt-decal.org. Applications are due on Friday,

help make CS 70 more accessible, we're starting In-
o Mathematical Thinking, a 2-unit DeCal meant to
ents to some ideas and concepts in discrete mathe-
e they're tested on them in CS 70. The course will
such as proof techniques, set theory, number the-
binatorics. We've worked with professors that have
) to pick these specific topics.

he full list of topics, along with the FAQs, at the
ite: http://imt-decal.org."

Public-Service Announcement II

enator Chow is looking for applicants to intern in her
ear. Interns could choose from a variety of commit-
ations/Media, UC Berkeley Club Recruitment Portal
, Christian Committee, etc.), adding resume expe-
ecoming ingratiated in a positive community driving
mpus.

can be found here: tinyurl.com/SenatorChowApp

s interested in CS opportunities specifically, Sen-
office is working to develop a platform similar to
n campus for club recruitment and extracurricular
on. As such, we need front-end coders and design-
s back end coders."

CS61B Lecture #3

e forgiving during the first week or so, but try to get
mitted by Thursday night. *DBC: Let us know if you can't
g to work!*

, there are almost 60 people who have accounts but do
repositories. You cannot hand anything in without the
get this part of the lab done!

re are about 950 students with accounts, which is sub-
s than the enrollment + waitlist. You must have an ac-
e a repo, which you need to turn things in!

crowded, so I may very well start dropping people who
o be doing the labs and homework.

courage signing up for classes with conflicting lectures,
re is a way to seek an exception. You will have a final
u have a lecture conflict; we do not consider such con-
d grounds to take an alternative final.

More Iteration: Sort an Array

out the command-line arguments in lexicographic or-

```
the quick brown fox jumped over the lazy dog
ox jumped lazy over quick the the
```

```
t {
rint WORDS lexicographically. */
void main(String[] words) {
0, words.length-1);
;

A[L..U], with all others unchanged. */
rt(String[] A, int L, int U) { /* "TOMORROW" */ }

one line, separated by blanks. */
int(String[] A) { /* "TOMORROW" */ }
```


How do We Know If It Works?

refers to the testing of individual units (methods, classes)
ram, rather than the whole program.

, we mainly use the JUnit tool for unit testing.

`TestYear.java` in lab #1.

testing refers to the testing of entire (integrated) set
the whole program.

e, we'll look at various ways to run the program against
uts and checking the output.

esting refers to testing with the specific goal of check-
s, enhancements, or other changes have not introduced
ssions).

Test-Driven Development

tests first.

nit at a time, run tests, fix and refactor until it works.

ally going to push it in this course, but it is useful and
ollowing.

Simple JUnit

ıckage provides some handy tools for unit testing.

notation `@Test` on a method tells the JUnit machinery nethod.

on in Java provides information about a method, class, n be examined within Java itself.)

of methods with names beginning with `assert` then allow ses to check conditions and report failures.

e.]

Testing sort

ty easy: just give a bunch of arrays to `sort` and then ney each get sorted properly.

e sure we cover the necessary cases:

ses. E.g., empty array, one-element, all elements the

tative "middle" cases. E.g., elements reversed, elements one pair of elements reversed, ...

Selection Sort

```
s A[L..U], with all others unchanged. */
ort(String[] A, int L, int U) {
{
ndexOfLargest(A, L, U);
[k] with A[U] }*/;
ems L to U-1 of A. }*/;


I0<=k<=I1, such that V[k] is largest element among
 V[I1]. Requires I0<=I1. */
lexOfLargest(String[] V, int i0, int i1) {
```


Selection Sort

```
s A[L..U], with all others unchanged. */
ort(String[] A, int L, int U) {
{
ndexOfLargest(A, L, U);
[k] with A[U] }*/;
ems L to U-1 of A. }*/;


I0<=k<=I1, such that V[k] is largest element among
 V[I1]. Requires I0<=I1. */
lexOfLargest(String[] V, int i0, int i1) {
```


Selection Sort

```
s A[L..U], with all others unchanged. */
ort(String[] A, int L, int U) {
{
*( Index s.t. A[k] is largest in A[L],...,A[U] )*/;
[k] with A[U] }*/;
ems L to U-1 of A. }*/;
```

Well, OK, not quite.

Selection Sort

```
s A[L..U], with all others unchanged. */
ort(String[] A, int L, int U) {
{
ndexOfLargest(A, L, U);
[k] with A[U] }*/;
 U-1);      // Sort items L to U-1 of A


I0<=k<=I1, such that V[k] is largest element among
 V[I1]. Requires I0<=I1. */
lexOfLargest(String[] V, int i0, int i1) {
```

Selection Sort

```
s A[L..U], with all others unchanged. */
ort(String[] A, int L, int U) {
{
ndexOfLargest(A, L, U);
 = A[k];  A[k] = A[U]; A[U] = tmp;
 U-1);      // Sort items L to U-1 of A
```

n:

```
U) {
ndexOfLargest(A, L, U);
 = A[k];  A[k] = A[U]; A[U] = tmp;
```


Selection Sort

```
s A[L..U], with all others unchanged. */
ort(String[] A, int L, int U) {
{
ndexOfLargest(A, L, U);
 = A[k];  A[k] = A[U]; A[U] = tmp;
 U-1);      // Sort items L to U-1 of A
```

terative version look like?

Find Largest

```
I0<=k<=I1, such that V[k] is largest element among
 V[I1]. Requires I0<=I1. */
lexOfLargest(String[] V, int i0, int i1) {
1)
{i0 < i1) */ {
```


Find Largest

```
I0<=k<=I1, such that V[k] is largest element among
 V[I1]. Requires I0<=I1. */
lexOfLargest(String[] V, int i0, int i1) {

```


Find Largest

```
I0<=k<=I1, such that V[k] is largest element among
 V[I1]. Requires I0<=I1. */
lexOfLargest(String[] V, int i0, int i1) {
1)
{i0 < i1) */ {
ndexOfLargest(V, i0 + 1, i1);
 whichever of i0 and k has larger value )*/;
```


Find Largest

```
I0<=k<=I1, such that V[k] is largest element among
 V[I1]. Requires I0<=I1. */
lexOfLargest(String[] V, int i0, int i1) {
1)
{i0 < i1) */ {
( index of largest value in V[i0 + 1..i1] )*/;
 whichever of i0 and k has larger value )*/;
```

Find Largest

```
I0<=k<=I1, such that V[k] is largest element among
  V[I1]. Requires I0<=I1. */
lexOfLargest(String[] V, int i0, int i1) {
1)
(i0 < i1) */ {
ndexOfLargest(V, i0 + 1, i1);
[i0].compareTo(V[k]) > 0) ? i0 : k;
0].compareTo(V[k]) > 0) return i0; else return k;
```

into an iterative version is tricky: not tail recursive.

e arguments to `compareTo` the first time it's called?

Iteratively Find Largest

```
I0<=k<=I1, such that V[k] is largest element among
  V[I1]. Requires I0<=I1. */
lexOfLargest(String[] V, int i0, int i1) {
1)
(i0 < i1) */ {
ndexOfLargest(V, i0 + 1, i1);
[i0].compareTo(V[k]) > 0) ? i0 : k;
0].compareTo(V[k]) > 0) return i0; else return k;

/ Deepest iteration
...?; i ...?)
```


Iteratively Find Largest

```
I0<=k<=I1, such that V[k] is largest element among
  V[I1]. Requires I0<=I1. */
lexOfLargest(String[] V, int i0, int i1) {
1)
(i0 < i1) */ {
ndexOfLargest(V, i0 + 1, i1);
[i0].compareTo(V[k]) > 0) ? i0 : k;
0].compareTo(V[k]) > 0) return i0; else return k;

// Deepest iteration
...?; i ...?)
```


Iteratively Find Largest

```
I0<=k<=I1, such that V[k] is largest element among
  V[I1]. Requires I0<=I1. */
lexOfLargest(String[] V, int i0, int i1) {
1)
(i0 < i1) */ {
ndexOfLargest(V, i0 + 1, i1);
[i0].compareTo(V[k]) > 0) ? i0 : k;
0].compareTo(V[k]) > 0) return i0; else return k;

// Deepest iteration
- 1; i >= i0; i -= 1)
.compareTo(V[k]) > 0) ? i : k;
```


Finally, Printing

```
 one line, separated by blanks. */
int(String[] A) {
= 0; i < A.length; i += 1)
.print(A[i] + " ");
println();


oduced a new syntax for the for loop here: */
s : A)
.print(s + " ");
ou like, but let's not stress over it yet! */
```

Your turn

```
Shove {

e elements A[k] to A[A.length-1] one element to the
 where k is the smallest index such that elements
ough A.length-2 are all larger than A[A.length-1].

id moveOver(int[] A) {
 IN
```


Another Problem

of integers, A, of length $N > 0$, find the smallest index,
elements at indices $\geq k$ and $< N - 1$ are greater than
rotate elements k to $N - 1$ right by one. For example,
as

```
3, 0, 12, 11, 9, 15, 22, 12 }
```

as

```
3, 0, 12, 11, 9, 12, 15, 22 }
```

mple,

```
3, 0, 12, 11, 9, 15, 22, -2 }


 4, 3, 0, 12, 11, 9, 15, 22 }
```