This Course Is About... (1/2)

Data structures: how to store data.

Examples: arrays, linked lists.

This course goes way beyond them.

But why? Why other data structures?

Because you need this course to graduate. :)
This Course Is About... (1/2)

Data structures: how to store data.

Examples: arrays, linked lists.

This course goes way beyond them.

But why? Why other data structures?

Because you need this course to graduate. :)

To speed up some kinds of lookups and updates.
This Course Is About... (2/2)

We will learn:

- Several data structures.
- Algorithms that look up and update them.
- Why the algorithms are correct. (Therefore what the data structures are good for.)
- How fast the algorithms are. Sometimes also how much space.
- Making up our own data structures and algorithms. (Or adapting from what we learned.)

The course cannot possibly cover all data structures! There are a lot more you will have to learn outside this course.

And Blackboard.
We now turn to stuff that will be on the exam!
Big $O$

“Linear search takes $O(n)$ time.”

“Binary search takes $O(lg(n))$ time.” ($lg$ means $\log_2$)

“Bubble sort takes $O(n^2)$ time.”

$n^2 + 2n + 1 \in O(n^2)$”

$n^2 + 2n + 1 \notin O(n)$”

?? What are they saying?
Let me scare you a bit first... 

(Assume: for all natural $n$, $f(n) \geq 0$, $g(n) \geq 0$.)
Let me scare you a bit first... 

(Assume: for all natural $n, f(n) \geq 0, g(n) \geq 0$.)

(In this course, 0 is a natural number.)
Let me scare you a bit first... 

(Assume: for all natural $n$, $f(n) \geq 0$, $g(n) \geq 0$.)

(In this course, 0 is a natural number.)

Definition: $f(n) \in O(g(n))$ iff

there exists real $c > 0$, natural $n_0$ such that for all natural $n \geq n_0$,

$$f(n) \leq c \times g(n)$$
Let me scare you a bit first... 

(Assume: for all natural $n$, $f(n) \geq 0$, $g(n) \geq 0$.)

(In this course, 0 is a natural number.)

Definition: $f(n) \in O(g(n))$ iff

there exists real $c > 0$, natural $n_0$ such that

for all natural $n \geq n_0$,

$$f(n) \leq c \times g(n)$$

Help! What is it saying?!
Let me scare you a bit first...

(Assume: for all natural \( n \), \( f(n) \geq 0 \), \( g(n) \geq 0 \).)

(In this course, 0 is a natural number.)

Definition: \( f(n) \in O(g(n)) \) iff

there exists real \( c > 0 \), natural \( n_0 \) such that for all natural \( n \geq n_0 \),

\[ f(n) \leq c \times g(n) \]

Help! What is it saying?!

Let me tell some revisionist history...
How much time does this take?

e = false;
for (i = 0; i < n; i++) {
    for (j = i+1; j < n; j++) {
        if (a[i] == a[j]) { e = true; }
    }
}

On one computer and one compiler:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>read i, j, n</td>
<td>2</td>
</tr>
<tr>
<td>read a[i]</td>
<td>3</td>
</tr>
<tr>
<td>write i, j</td>
<td>3</td>
</tr>
<tr>
<td>arithmetic</td>
<td>1</td>
</tr>
<tr>
<td>two-way branch</td>
<td>1 if continue, 2 if exit</td>
</tr>
<tr>
<td>loop back</td>
<td>1</td>
</tr>
</tbody>
</table>

Total: $12n^2 + 9n + 8$
How much time does this take?

```plaintext
e = false;
for (i = 0; i < n; i++) {
    for (j = i+1; j < n; j++) {
        if (a[i] == a[j]) { e = true; }
    }
}
```

On another computer and another compiler:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>read i, j, n</td>
<td>1 ns</td>
</tr>
<tr>
<td>read a[i]</td>
<td>5 ns</td>
</tr>
<tr>
<td>write i, j</td>
<td>1 ns</td>
</tr>
<tr>
<td>arithmetic</td>
<td>1 ns</td>
</tr>
<tr>
<td>two-way branch</td>
<td>0 ns if continue, 2 ns if exit</td>
</tr>
<tr>
<td>loop back</td>
<td>1 ns</td>
</tr>
</tbody>
</table>

Total: $9.5n^2 + 2.5n + 4$
How much time does this take?

```java
int e = false;
for (int i = 0; i < n; i++) {
    for (int j = i+1; j < n; j++) {
        if (a[i] == a[j]) { e = true; }
    }
}
```

To a theoretician:

<table>
<thead>
<tr>
<th></th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>the j-loop</td>
<td>$D(n - i - 1)$</td>
</tr>
<tr>
<td>the i-loop</td>
<td>$\sum_{i=0}^{n-1} (C + \text{the j-loop})$</td>
</tr>
<tr>
<td>the whole</td>
<td>$B + \text{the i-loop}$</td>
</tr>
</tbody>
</table>

Total: $\frac{D}{2}n^2 + (C + \frac{D}{2})n + B$
How to be both Coarse and Rigorous?

\[ 12n^2 + 9n + 8 \]
\[ 9.5n^2 + 2.5n + 4 \]
\[ \frac{D}{2}n^2 + (C + \frac{D}{2})n + B \]

How to unify them and ignore machine differences?

What is common about them?
How to be both Coarse and Rigorous?

\[ 12n^2 + 9n + 8 \]
\[ 9.5n^2 + 2.5n + 4 \]
\[ \frac{D}{2}n^2 + (C + \frac{D}{2})n + B \]

How to unify them and ignore machine differences?

What is common about them? Quadratic polynomials.
How to be both Coarse and Rigorous?

\[12n^2 + 9n + 8\]
\[9.5n^2 + 2.5n + 4\]
\[\frac{D}{2} n^2 + (C + \frac{D}{2}) n + B\]

How to unify them and ignore machine differences?

What is common about them? Quadratic polynomials.

Poster-boy for quadratic polynomials: \(n^2\).

How to rigorously say: the above functions are “like” \(n^2\)?
How to be both Coarse and Rigorous?

\[ 12n^2 + 9n + 8 \]
\[ 9.5n^2 + 2.5n + 4 \]
\[ \frac{D}{2} n^2 + (C + \frac{D}{2})n + B \]

How to unify them and ignore machine differences?

What is common about them? Quadratic polynomials.

Poster-boy for quadratic polynomials: \( n^2 \).

How to rigorously say: the above functions are “like” \( n^2 \)?

And cubic polynomials are “like” \( n^3 \)?
How to be both Coarse and Rigorous?

12\(n^2\) + 9\(n\) + 8  
9.5\(n^2\) + 2.5\(n\) + 4  
\(\frac{D}{2}n^2 + (C + \frac{D}{2})n + B\)

How to unify them and ignore machine differences?

What is common about them? Quadratic polynomials.

Poster-boy for quadratic polynomials: \(n^2\).

How to rigorously say: the above functions are “like” \(n^2\)?

And cubic polynomials are “like” \(n^3\)?

And \(4n \log_2(n) + 2n + 10\) is “like” \(n \log_2(n)\)?

Etc., etc.
How to be both Coarse and Rigorous!

Watch this mathemagic:

$$12n^2 + 9n + 8 \leq 12n^2 + 9n^2 + 8$$

$$= 21n^2 + 8$$
How to be both Coarse and Rigorous!

Watch this mathemagic:

\[ 12n^2 + 9n + 8 \leq 12n^2 + 9n^2 + 8 \]
\[ = 21n^2 + 8 \]

For all natural \( n \geq 8 \):

\[ 21n^2 + 8 \leq 21n^2 + n \]
\[ \leq 21n^2 + n^2 \]
\[ = 22n^2 \]
How to be both Coarse and Rigorous!

Watch this mathemagic:

$$12n^2 + 9n + 8 \leq 12n^2 + 9n^2 + 8$$
$$= 21n^2 + 8$$

For all natural $n \geq 8$:

$$21n^2 + 8 \leq 21n^2 + n$$
$$\leq 21n^2 + n^2$$
$$= 22n^2$$

There exists natural $n_0$ such that for all natural $n \geq n_0$, $12n^2 + 9n + 8 \leq 22n^2$
How to be both Coarse and Rigorous!

Watch this mathemagic:

\[
12n^2 + 9n + 8 \leq 12n^2 + 9n^2 + 8
= \quad 21n^2 + 8
\]

For all natural \( n \geq 8 \):

\[
21n^2 + 8 \leq 21n^2 + n
\leq 21n^2 + n^2
= \quad 22n^2
\]

There exists natural \( n_0 \) such that
for all natural \( n \geq n_0 \), \( 12n^2 + 9n + 8 \leq 22n^2 \)

There exists real \( c > 0 \), natural \( n_0 \) such that
for all natural \( n \geq n_0 \), \( 12n^2 + 9n + 8 \leq cn^2 \)
The Rise of Big $O$

There exists real $c > 0$, natural $n_0$ such that for all natural $n \geq n_0$, $12n^2 + 9n + 8 \leq cn^2$

$12n^2 + 9n + 8 \in O(n^2)$
The Rise of Big $O$

There exists real $c > 0$, natural $n_0$ such that for all natural $n \geq n_0$, $12n^2 + 9n + 8 \leq cn^2$

$12n^2 + 9n + 8 \in O(n^2)$
$9.5n^2 + 2.5n + 4 \in O(n^2)$
$\frac{D}{2}n^2 + (C + \frac{D}{2})n + B \in O(n^2)$
The Rise of Big $O$

There exists real $c > 0$, natural $n_0$ such that for all natural $n \geq n_0$, $12n^2 + 9n + 8 \leq cn^2$

$12n^2 + 9n + 8 \in O(n^2)$
$9.5n^2 + 2.5n + 4 \in O(n^2)$
\[
\frac{D}{2}n^2 + (C + \frac{D}{2})n + B \in O(n^2)
\]

My algorithm takes time in $O(n^2)$
My algorithm takes $O(n^2)$ time
My algorithm takes time on the order of $n^2$
The Rise of Big $O$

There exists real $c > 0$, natural $n_0$ such that for all natural $n \geq n_0$, $12n^2 + 9n + 8 \leq cn^2$

$12n^2 + 9n + 8 \in O(n^2)$
$9.5n^2 + 2.5n + 4 \in O(n^2)$
$\frac{D}{2}n^2 + (C + \frac{D}{2})n + B \in O(n^2)$

My algorithm takes time in $O(n^2)$
My algorithm takes $O(n^2)$ time
My algorithm takes time on the order of $n^2$

$4n \log_2(n) + 2n + 10 \in O(n \log_2(n))$
The Rise of Big $O$

There exists real $c > 0$, natural $n_0$ such that for all natural $n \geq n_0$, $12n^2 + 9n + 8 \leq cn^2$

$12n^2 + 9n + 8 \in O(n^2)$
$9.5n^2 + 2.5n + 4 \in O(n^2)$
$\frac{D}{2}n^2 + (C + \frac{D}{2})n + B \in O(n^2)$

My algorithm takes time in $O(n^2)$
My algorithm takes $O(n^2)$ time
My algorithm takes time on the order of $n^2$

$4n \log_2(n) + 2n + 10 \in O(n \log_2(n))$

The definition is scary sophisticated because it has to drop some information and carefully preserve some other.
But wait, these are also true:

- $n \in O(n^2)$
- $3 \in O(n^2)$

$O(n^2)$ includes quadratic functions as well as “lesser” functions. We need another definition to exclude “lesser” functions.
More or Less

But wait, these are also true:

- $n \in O(n^2)$
- $3 \in O(n^2)$

$O(n^2)$ includes quadratic functions as well as “lesser” functions. We need another definition to exclude “lesser” functions.

$n \in O(n^2)$ because it only requires

there exists ... for all ... $n \leq cn^2$

It’s only a one-sided inequality.

The definition in the next slide uses a two-sided inequality to rule out this.
The Rise of Big $\Theta$

(Assume: for all natural $n$, $f(n) \geq 0$, $g(n) \geq 0$.)

Definition: $f(n) \in \Theta(g(n))$ iff

there exists real $b > 0$, real $c > 0$, natural $n_0$ such that for all natural $n \geq n_0$,

$$b \times g(n) \leq f(n) \leq c \times g(n)$$

$\Theta$ is Greek capital theta.
The Rise of Big $\Theta$

(Assume: for all natural $n$, $f(n) \geq 0$, $g(n) \geq 0$.)

Definition: $f(n) \in \Theta(g(n))$ iff

there exists real $b > 0$, real $c > 0$, natural $n_0$ such that
for all natural $n \geq n_0$,

$$b \times g(n) \leq f(n) \leq c \times g(n)$$

$\Theta$ is Greek capital theta.

Example: $12n^2 + 9n + 8 \in \Theta(n^2)$ because
for all $n \geq 8$, $1n^2 \leq 12n^2 + 9n + 8 \leq 22n^2$
The Rise of Big $\Theta$

(Assume: for all natural $n, f(n) \geq 0, g(n) \geq 0$.)

Definition: $f(n) \in \Theta(g(n))$ iff

there exists real $b > 0$, real $c > 0$, natural $n_0$ such that
for all natural $n \geq n_0$,

$$b \times g(n) \leq f(n) \leq c \times g(n)$$

$\Theta$ is Greek capital theta.

Example: $12n^2 + 9n + 8 \in \Theta(n^2)$ because
for all $n \geq 8$, $1n^2 \leq 12n^2 + 9n + 8 \leq 22n^2$

Example: $n \not\in \Theta(n^2)$ because
(sketch) you can’t make $bn^2 \leq n$ to work.
Using Limits to Prove Big $O$

Theorem: If $\lim_{n \to \infty} \frac{f(n)}{g(n)}$ exists and is finite, then $f(n) \in O(g(n))$.

Example: Prove $\frac{n(n+1)}{2} \in O(n^2)$.

$$\lim_{n \to \infty} \frac{n(n+1)}{2n^2} = \lim_{n \to \infty} \frac{1}{2} \frac{1}{n} = 0$$

Therefore $\frac{n(n+1)}{2} \in O(n^2)$.

Example: Prove $\ln(n) \in O(n)$.

$$\lim_{n \to \infty} \frac{\ln(n)}{n} = \lim_{n \to \infty} \frac{1}{n} = 0$$

Therefore $\ln(n) \in O(n)$. 

13 / 30
Using Limits to Prove Big $O$

Theorem: If $\lim_{n \to \infty} \frac{f(n)}{g(n)}$ exists and is finite, then $f(n) \in O(g(n))$.

Example: Prove $n(n + 1)/2 \in O(n^2)$

$$\lim_{n \to \infty} \frac{n(n + 1)/2}{n^2} = \frac{1}{2}$$

Therefore $n(n + 1)/2 \in O(n^2)$

Example: Prove $\ln(n) \in O(n)$
Using Limits to Prove Big $O$

Theorem: If $\lim_{n \to \infty} \frac{f(n)}{g(n)}$ exists and is finite, then $f(n) \in O(g(n))$.

Example: Prove $n(n + 1)/2 \in O(n^2)$

$$\lim_{n \to \infty} \frac{n(n + 1)/2}{n^2} = \frac{1}{2}$$

Therefore $n(n + 1)/2 \in O(n^2)$

Example: Prove $\ln(n) \in O(n)$

$$\lim_{n \to \infty} \frac{\ln(n)}{n} = \lim_{n \to \infty} \frac{1/n}{1} = 0$$

Therefore $\ln(n) \in O(n)$
Using Limits to Disprove Big $O$

Theorem: If $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$, then $f(n) \not\in O(g(n))$. 

Example: Disprove $n^2 \in O(n)$.

$$\lim_{n \to \infty} \frac{n^2}{n} = \lim_{n \to \infty} n = \infty$$

Therefore $n^2 \not\in O(n)$.

Example: Disprove $n \in O(\ln(n))$.

$$\lim_{n \to \infty} \frac{n}{\ln(n)} = \lim_{n \to \infty} \frac{1}{\frac{1}{n}} = \lim_{n \to \infty} n = \infty$$

Therefore $n \not\in O(\ln(n))$. 
Using Limits to Disprove Big $O$

Theorem: If $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$, then $f(n) \notin O(g(n))$.

Example: Disprove $n^2 \in O(n)$

\[
\lim_{n \to \infty} \frac{n^2}{n} = \lim_{n \to \infty} n = \infty
\]

Therefore $n^2 \notin O(n)$

Example: Disprove $n \in O(\ln(n))$

\[
\lim_{n \to \infty} \frac{n}{\ln(n)} = \lim_{n \to \infty} \frac{1}{\frac{1}{n}} = \lim_{n \to \infty} n = \infty
\]

Therefore $n \notin O(\ln(n))$
Using Limits to Disprove Big $\mathcal{O}$

Theorem: If $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$, then $f(n) \not\in \mathcal{O}(g(n))$.

Example: Disprove $n^2 \in \mathcal{O}(n)$

$$\lim_{n \to \infty} \frac{n^2}{n} = \lim_{n \to \infty} n = \infty$$

Therefore $n^2 \not\in \mathcal{O}(n)$

Example: Disprove $n \in \mathcal{O}(\ln(n))$

$$\lim_{n \to \infty} \frac{n}{\ln(n)} = \lim_{n \to \infty} \frac{1}{1/n} = \lim_{n \to \infty} n = \infty$$

Therefore $n \not\in \mathcal{O}(\ln(n))$
When Limits Don’t Help

Theorem: If \( \lim_{n \to \infty} \frac{f(n)}{g(n)} \) exists and is finite, then . . .

Theorem: If \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty \), then . . .

Which case has not been covered?
When Limits Don’t Help

Theorem: If $\lim_{n \to \infty} \frac{f(n)}{g(n)}$ exists and is finite, then . . .

Theorem: If $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$, then . . .

Which case has not been covered?

If $\lim_{n \to \infty} \frac{f(n)}{g(n)}$ does not exist and is not $\infty$, then no conclusion.

Hopefully you won’t run into this case.
Examples When Limits Don’t Help

Define: \( \text{drunk}(n) = \) if \( n \) is even then 1 else \( n \)

(Plot it to see what it looks like!)

Example:

\[
\lim_{n \to \infty} \frac{\text{drunk}(n)}{n}
\]

does not exist and is not \( \infty \), and still \( \text{drunk}(n) \in O(n) \).

Example:

\[
\lim_{n \to \infty} \frac{\text{drunk}(n)}{1}
\]

does not exist and is not \( \infty \), and this time \( \text{drunk}(n) \notin O(1) \)
Using Limits for $\Theta$

Theorem: $f(n) \in \Theta(g(n))$ iff

$$f(n) \in O(g(n)) \text{ and } g(n) \in O(f(n))$$

Handy when you want to use limits.
(Not handy when you want to prove the $\exists \forall$ directly.)
Using Limits for $\Theta$

Theorem: $f(n) \in \Theta(g(n))$ iff

$$f(n) \in O(g(n)) \text{ and } g(n) \in O(f(n))$$

Handy when you want to use limits.
(Not handy when you want to prove the $\exists \forall$ directly.)

Example: $n^2 + n^{3/2} \in \Theta(n^2)$

- $n^2 + n^{3/2} \in O(n^2)$ by a limit
- $n^2 \in O(n^2 + n^{3/2})$ by a similar limit
Using Limits for $\Theta$

Theorem: $f(n) \in \Theta(g(n))$ iff

\[ f(n) \in O(g(n)) \text{ and } g(n) \in O(f(n)) \]

Handy when you want to use limits.  
(Not handy when you want to prove the $\exists \forall$ directly.)

Example: $n^2 + n^{3/2} \in \Theta(n^2)$

- $n^2 + n^{3/2} \in O(n^2)$ by a limit
- $n^2 \in O(n^2 + n^{3/2})$ by a similar limit

Example: $\ln(n) \not\in \Theta(n)$

You saw $n \not\in O(\ln(n))$ by a limit.
Big $O$, Big $\Theta$ May Miss Something

$10^{100} n \in \Theta(n)$

$n + 10^{100} \in \Theta(n)$

Can’t say these are practical algorithm times. But $O$, $\Theta$ can’t detect them. This is a price for ignoring machine differences.

These pathological cases are rare. $O$ and $\Theta$ are usually informative.
Big $O$, Big $\Theta$ Gain Simplification

```java
    e = false;
    for (i = 0; i < n; i++) {
        for (j = i+1; j < n; j++) {
            if (a[i] == a[j]) { e = true; }
        }
    }
```

“$12n^2 + 9n + 8$ ns”
Took me 15 minutes to figure out.

“The inner loop takes $O(n - i - 1)$ time, which is $O(n)$. The outer loop takes $n$ iterations. Altogether $O(n^2)$ time.”
This only takes 30 seconds to figure out.

It is one of the reasons we use $O$ and $\Theta$. 
$O$ vs $\Theta$

My friend says: “my algorithm takes $O(n^2)$ time.”

- It may mean $9n^2 + 4n + 13$

$3n + 2 \in O(n^2)$ is still true.

- It may take 45 time.

$45 \in O(n^2)$ is still true.

My opinion: You should say $\Theta$ as much as you can.

Popular opinion: Just say $O$.

A good use of $O$: Assignment or contract says: “Write a $O(n^2)$-time algorithm.” It allows you to do better.
My friend says: “my algorithm takes $O(n^2)$ time.”

- It may mean $9n^2 + 4n + 13$
- It may mean $3n + 2$ time.
  
  $3n + 2 \in O(n^2)$ is still true.
My friend says: “my algorithm takes $O(n^2)$ time.”

- It may mean $9n^2 + 4n + 13$
- It may mean $3n + 2$ time. $3n + 2 \in O(n^2)$ is still true.
- It may take 45 time. $45 \in O(n^2)$ is still true.

My opinion: You should say $\Theta$ as much as you can.

Popular opinion: Just say $O$.

A good use of $O$: Assignment or contract says: “Write a $O(n^2)$-time algorithm.” It allows you to do better.
My friend says: “my algorithm takes $O(n^2)$ time.”

- It may mean $9n^2 + 4n + 13$
- It may mean $3n + 2$ time. $3n + 2 \in O(n^2)$ is still true.
- It may take 45 time. $45 \in O(n^2)$ is still true.

My opinion: You should say $\Theta$ as much as you can.
Popular opinion: Just say $O$.

A good use of $O$: Assignment or contract says: “Write a $O(n^2)$-time algorithm.” It allows you to do better.
Worse Case, Best Case

e = false;
for (i = 0; i < n; i++) {
    for (j = i+1; j < n; j++) {
        if (a[i] == a[j]) { e = true; break; }
    }
}
if (e) { break; }
Worse Case, Best Case

\begin{verbatim}
e = false;
for (i = 0; i < n; i++) {
    for (j = i+1; j < n; j++) {
        if (a[i] == a[j]) { e = true; break; }
    }
    if (e) { break; }
}
\end{verbatim}

Depending on what's in a:
Anywhere from $\Theta(1)$ time to $\Theta(n^2)$ time.

Best case is $\Theta(1)$ time.
Worse case is $\Theta(n^2)$ time.

We look at worst cases in this course mostly.
Myth Buster

Myth: $O$ means worst case time.

Truth: $O$ and $\Theta$ classify functions, do not say what the functions are for.

$9n^2 + 4n + 13$ may be best case time, or worst case time, or best case space, or worst case space, or just a polynomial from nowhere.

$9n^2 + 4n + 13 \in O(n^2)$ is true regardless.

“Best case time is in $O(n^2)$” is allowed. It means: Best case time is some function, that function is in $O(n^2)$. Clearly a sensible statement and possible scenerio.

$O$ and $\Theta$ are good for any function from natural to non-negative real.
Binary Search Tree: Introduction

```java
public class Node<K extends Comparable<K>> {
    public K key;
    public Node<K> left, right;
}
```

Why would you store data this way?
I want to store a set of keys.

I’m given: keys can be compared ($<$, $=$, $>$).

I want these operations to be fast:

- $s$.find($x$): return whether $x$ is in $s$
- $s$.insert($x$): add $x$ to $s$
- $s$.remove($x$): delete $x$ from $s$
Storing a Set: Near Miss

(Reminder: Keys can be compared (<, =, >).)

If I wanted this only:

- `s.find(x)`: return whether x is in s

then you already know how to do it.
Storing a Set: Near Miss

(Reminder: Keys can be compared ($<$, $=$, $>$).)

If I wanted this only:

- $s$.find($x$): return whether $x$ is in $s$

then you already know how to do it.

Store in an array in increasing order:

| 8 | 13 | 20 | 42 | 47 | 85 | 91 |

Use binary search for find. $O(\log_2(n))$ time.
Storing a Set: Near Miss

(Reminder: Keys can be compared (<, =, >).

If I wanted this only:

- \( s \cdot \text{find}(x) \): return whether \( x \) is in \( s \)

then you already know how to do it.

Store in an array in increasing order:

\[
\begin{array}{cccccc}
8 & 13 & 20 & 42 & 47 & 85 & 91
\end{array}
\]

Use binary search for \text{find}. \( O(\log_2(n)) \) time.

But \text{insert} and \text{remove} are icky.
Storing a Set: Binary Search Tree

8 13 20 42 47 85 91

Good news for `s.find(x)`:
left child, right child are exactly what you would try next in binary search!
Binary Search Tree: Definition

A binary search tree:

- is a binary tree
- at each node $v$:
  - $v.key$ is greater than all keys in the left sub-tree
  - $v.key$ is smaller than all keys in the right sub-tree
private Node<K> root;
...
public boolean find(K x) {
    Node<K> v = root;
    while (v != null) {
        int r = v.key.compareTo(x);
        if (r == 0) {
            return true;
        } else {
            v = r > 0 ? v.left : v.right;
        }
    }
    return false;
}
public void insert(K x) {
    if (root == null) {
        root = new Node<K>(x);
    } else {
        Node<K> v = root;
        for (; ;) {
            if (v.key.compareTo(x) == 0) {
                return;
            } else if (v.key.compareTo(x) > 0) {
                if (v.left == null) {
                    v.left = new Node<K>(x);
                    return;
                } else {
                    v = v.left;
                }
            } else {
                // similar, except it’s v.right
            }
        }
    }
}
Binary Search Tree: insert trouble

Start with the empty tree.
Insert 1, 2, \ldots, n in that order.
What do you get?
Binary Search Tree: insert trouble

Start with the empty tree.
Insert 1, 2, \ldots, n in that order.

What do you get?

The tree is leaned on one side.
find, insert, remove are not $O(\log_2(n))$ anymore.

What to do? What to do?

Cliff hanger! Come next time for a solution!