COMP 250

Lecture 18

big Theta  Θ

best and worst cases

limits (revisited)

Oct. 20, 2017
“small theta” $\theta$

“big theta” $\Theta$
Definition of Big Theta ($\Theta$)

Let $t(n)$ and $g(n)$ be two functions of $n \geq 0$.

We say $t(n)$ is $\Theta(g(n))$ if $t(n)$ is $O(g(n))$ and $t(n)$ is $\Omega(g(n))$.
Definition of Big Theta (Θ)

Let \( t(n) \) and \( g(n) \) be two functions of \( n \geq 0 \).

We say \( t(n) \) is \( \Theta(g(n)) \), if there exist three positive constants \( n_0, c_1, c_2 \) such that for all \( n \geq n_0 \),

\[
c_1 \ g(n) \leq t(n) \leq c_2 \ g(n)
\]
Example

\[ t(n) \text{ is } \Theta(g(n)). \]
Definition of Big Theta ($\Theta$)

Let $t(n)$ and $g(n)$ be two functions of $n \geq 0$.

We say $t(n)$ is $\Theta(g(n))$, if there exist three positive constants $n_0$, $c_1$, $c_2$ such that for all $n \geq n_0$,

$$c_1 \cdot g(n) \leq t(n) \leq c_2 \cdot g(n)$$

$t(n)$ is $O(g(n))$
Definition of Big Theta ($\Theta$)

Let $t(n)$ and $g(n)$ be two functions of $n \geq 0$.

We say $t(n)$ is $\Theta(g(n))$, if there exist three positive constants $n_0$, $c_1$, $c_2$ such that for all $n \geq n_0$,

$$c_1 \ g(n) \leq t(n) \leq c_2 \ g(n)$$

$t(n)$ is $\Omega(g(n))$
Example

Let \( t(n) = 4 + 17 \log_2 n + 3n + 9 n \log_2 n + \frac{n(n-1)}{2} \)

Claim: \( t(n) \) is \( \Theta(n^2) \).

Proof:
Example

Let $t(n) = 4 + 17 \log_2 n + 3n + 9n \log_2 n + \frac{n(n-1)}{2}$

Claim: $t(n)$ is $\Theta(n^2)$.

Proof:

$$\frac{n^2}{4} \leq t(n) \leq (4 + 17 + 3 + 9 + \frac{1}{2}) n^2$$
For every $t(n)$, does there exist a “simple” $g(n)$ such that $t(n)$ is $\Theta()$?
For every $t(n)$, does there exist a “simple” $g(n)$ such that $t(n)$ is $\Theta()$?

No, as this contrived example shows:

Let $t(n) = \begin{cases} 
  n, & \text{n is odd} \\
  n^2, & \text{n is even.}
\end{cases}$

$t(n)$ is $O(n^2)$, but $t(n)$ is not $O(n)$.

$t(n)$ is $\Omega(n)$, but $t(n)$ is not $\Omega(n^2)$. 
Sets of $\Theta()$ functions

If $t(n)$ is $\Theta(g(n))$, we often write $t(n) \in \Theta(g(n))$,

That is, $t(n)$ is a member of the set of functions that are $\Theta(g(n))$. 

$\Theta(1)$  $\Theta(\log_2 n)$  $\Theta(n)$  ...  $\Theta(2^n)$  $\Theta(n!)$
COMP 250

Lecture 18

big Theta \( \Theta \)

best and worst cases

limits (revisited)

Oct. 20, 2017
The time it takes for an algorithm to run depends on:

- constant factors (often implementation dependent)
- the size \( n \) of the input
- ?
The time it takes for an algorithm to run depends on:

- constant factors (often implementation dependent)
- the size $n$ of the input
- the values of the input, including arguments if applicable

What are the best and worst cases?
For any algorithm,

let $t_{\text{best}}(n)$ be the runtime on the best case input(s).

let $t_{\text{worst}}(n)$ be the runtime for the worse case input(s).
e.g. Consider removing an element from an arbitrary position in an arraylist:

```java
arraylist.remove (i)
```

In the best case, ....

In the worst case, ....
e.g. Consider removing an element from an arbitrary position in an arraylist:

```java
arraylist.remove(i)
```

In the best case, the element is removed from the end of and so it can be done in constant time. So,

$$t_{best}(n) \text{ is } \Theta(1).$$

In the worst case, ...
Consider removing an element from an arbitrary position in an arraylist:

\[
\text{arraylist.remove } (i)
\]

In the best case, the element is removed from the end of and so it can be done in constant time. So,

\[
t_{\text{best}}(n) \text{ is } \Theta(1).
\]

In the worst case, the element is removed from the start of the arraylist, so all elements must be shifted. So,

\[
t_{\text{worst}}(n) \text{ is } \Theta(n).
\]
e.g. Consider finding an element in a doubly linked list:

```python
def get(e):
    pass
```

In the best case, ...

In the worst case, ...
Consider finding an element in a doubly linked list:

```python
linkedlist.get(e)
```

In the best case, the element is at the front of the list and so it can be found in constant time. So,

\[ t_{best}(n) \text{ is } \Theta(1). \]

In the worst case, ...
Consider finding an element in a doubly linked list:

\[ \text{linkedlist.get}(e) \]

In the best case, the element is at the front of the list and so it can be found in constant time. So,

\[ t_{\text{best}}(n) \text{ is } \Theta(1). \]

In the worst case, the element is at the opposite end of the list from where we start. So,

\[ t_{\text{worst}}(n) \text{ is } \Theta(n). \]
e.g. Consider sorting a list using quicksort:

```python
list.quicksort()
```

In the best case, ...

In the worst case, ...
e.g. Consider sorting a list using quicksort:

```
list.quicksort()
```

In the best case, each choice of pivot partitions the list into two roughly equal parts. So,

\[ t_{best}(n) \text{ is } \Theta(n \log n). \]

In the worst case, ...
e.g. Consider sorting a list using quicksort:

```python
class list:
    def quicksort(self):
```

In the best case, each choice of pivot produces two lists of roughly equal size. So,

\[ t_{\text{best}}(n) \text{ is } \Theta(n \log n). \]

In the worst case, each choice of pivot produces an empty list and a list with one fewer element. So,

\[ t_{\text{worst}}(n) \text{ is } \Theta(n^2). \]
<table>
<thead>
<tr>
<th>Operations/Algorithms for Lists</th>
<th>$t_{\text{best}}(n)$</th>
<th>$t_{\text{worst}}(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>add, remove, find an element</td>
<td>$\Theta(1)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>(array list)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>add, remove, find an element</td>
<td>$\Theta(1)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>(doubly linked list)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>insertion sort</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td>selection sort</td>
<td>$\Theta(n^2)$</td>
<td>$\Theta(n^2)$</td>
</tr>
<tr>
<td>binary search</td>
<td>$\Theta(\log n)$</td>
<td>$\Theta(\log n)$</td>
</tr>
<tr>
<td>(sorted array list)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mergesort</td>
<td>$\Theta(n \log n)$</td>
<td>$\Theta(n \log n)$</td>
</tr>
<tr>
<td>quick sort</td>
<td>$\Theta(n \log n)$</td>
<td>$\Theta(n^2)$</td>
</tr>
</tbody>
</table>
Heads up!

When people want to express that an algorithm has different asymptotic behavior in best versus worst cases, they sometimes say that the algorithm is $\Omega( g_{best}(n) )$ and $O( g_{worst}(n) )$.

e.g. “Quicksort is $\Omega( n \log_2 n )$ and $O( n^2 )$.”

The statement is technically correct. However, there is natural tendency to equate $\Omega$ with best case and $O$ with worst case. Technically speaking, this is incorrect.
Rather, for any algorithm,

t_{\text{best}}(n) \text{ is the runtime on the best case input(s).}

t_{\text{worst}}(n) \text{ is the runtime for the worse case input(s).}

\text{t_{best}(n) and t_{worst}(n) each typically belong to some } \Theta() \text{ set, possibly the same.}

\Theta(1) \quad \Theta(\log_2 n) \quad \Theta(n) \quad \text{...} \quad \Theta(2^n) \quad \Theta(n!)

COMP 250

Lecture 18

big Theta

best and worst cases

limits (revisited)

Oct. 20, 2017
Q: Can we use limits to prove asymptotic bounds?

A: Yes, if we apply certain rules.
Recall: Definition of Big O

Let $t(n)$ and $g(n)$ be two functions, where $n \geq 0$.

We say $t(n)$ is $O(g(n))$, if there exist two positive constants $n_0$ and $c$ such that, for all $n \geq n_0$,

$$t(n) \leq c \cdot g(n)$$

or equivalently

$$\frac{t(n)}{g(n)} \leq c.$$
A sequence \( t(n) \) is \( O(g(n)) \), if there exists positive constants \( n_0 \) and \( c \) such that, for all \( n \geq n_0 \),

\[
\frac{t(n)}{g(n)} \leq c.
\]

A sequence \( t(n) \) has a limit \( t_\infty \) if, for any \( \varepsilon > 0 \), there exists an \( n_0 \) such that for any \( n \geq n_0 \),

\[
| t(n) - t_\infty | < \varepsilon.
\]
Rule 1

If

\[
\lim_{{n \to \infty}} \frac{t(n)}{g(n)} = 0
\]

then

?
If
\[ \lim_{n \to \infty} \frac{t(n)}{g(n)} = 0 \]

then, by definition,

for any \( \varepsilon > 0 \), there exists an \( n_0 \)

such that for any \( n \geq n_0 \),

\[ \left| \frac{t(n)}{g(n)} - 0 \right| < \varepsilon. \]
If \[ \lim_{n \to \infty} \frac{t(n)}{g(n)} = 0 \]

then, by definition,

in particular, there exists \( \varepsilon > 0 \) and \( \text{for any } \varepsilon > 0 \), there exists an \( n_0 \)

such that for any \( n \geq n_0 \),

\[
\left| \frac{t(n)}{g(n)} - 0 \right| < \varepsilon
\]

and so \( \frac{t(n)}{g(n)} < \varepsilon \) and so \( t(n) \) is \( O( g(n) ) \).
What about the opposite direction?

If \( t(n) \) is \( \text{O}(g(n)) \).

then:

\[
\lim_{n \to \infty} \frac{t(n)}{g(n)} = 0
\]
The opposite direction does not always hold.

e.g. Take \( t(n) = g(n) \). Then \( t(n) \) is \( O(g(n)) \).

\[
\text{but } \frac{t(n)}{g(n)} = 1 \text{ for all } n, \text{ and so the limit isn't 0.}
\]
Rule 1 (more general)

If

\[ \lim_{n \to \infty} \frac{t(n)}{g(n)} = 0 \]

then:

\( t(n) \) is \( O(g(n)) \).

\( t(n) \) is not \( \Omega(g(n)) \)

Why not?

Thus,

\( t(n) \) is not \( \Theta(g(n)) \).
Recall Definition of Big Omega

Let $t(n)$ and $g(n)$ be two functions, where $n \geq 0$.

We say $t(n)$ is $\Omega(g(n))$, if there exist two positive constants $n_0$ and $c$ such that, for all $n \geq n_0$,

$$t(n) \geq c \cdot g(n)$$

or equivalently

$$\frac{t(n)}{g(n)} \geq c.$$
$$\lim_{n \to \infty} \frac{t(n)}{g(n)} = 0$$

$t(n)$ is \( \Omega(g(n)) \), if there exist two constants \( n_0 \) and \( c > 0 \) such that, for all \( n \geq n_0 \), \( \frac{t(n)}{g(n)} \geq c \).

It is impossible that both of the above are true!
If \[ \lim_{n \to \infty} \frac{t(n)}{g(n)} = 0 \]

then:

\[ t(n) \text{ is } O( g(n) ). \]

\[ t(n) \text{ is not } \Omega( g(n) ) \quad \text{Proved!} \]

Thus,

\[ t(n) \text{ is not } \Theta( g(n) ). \]
There are two more limit rules which I state on the next two slides.

In the lecture notes, I give rigorous proofs.
If
\[
\lim_{n \to \infty} \frac{t(n)}{g(n)} = \infty
\]

then:
\[
t(n) \text{ is } \Omega(g(n)).
\]
\[
t(n) \text{ is not } O(g(n)).
\]
Thus,
\[
t(n) \text{ is not } \Theta(g(n)).
\]
If \[ \lim_{n \to \infty} \frac{t(n)}{g(n)} = c, \quad 0 < c < \infty \]

then \( t(n) \) is \( \Theta (g(n)) \).

The opposite direction is not true.

(See lecture notes for counterexample.)
The End