CSE 373: Asymptotic Analysis, BSTs

Michael Lee
Friday, Jan 12, 2018
Warmup: True or false:

- $5n + 3 \in O(n)$
- $n \in O(5n + 3)$
- $5n + 3 = O(n)$
- $O(5n + 3) = O(n)$
- $O(n^2) = O(n)$
- $n^2 \in O(1)$
- $n^2 \in O(n)$
- $n^2 \in O(n^2)$
- $n^2 \in O(n^3)$
- $n^2 \in O(n^{100})$
Warmup: True or false:

- $5n + 3 \in O(n)$: True
- $n \in O(5n + 3)$: True
- $5n + 3 = O(n)$: True (by convention)
- $O(5n + 3) = O(n)$: True
- $O(n^2) = O(n)$: False
- $n^2 \in O(1)$: False
- $n^2 \in O(n)$: False
- $n^2 \in O(n)$: False
- $n^2 \in O(n^2)$: True
- $n^2 \in O(n^3)$: True
- $n^2 \in O(n^{100})$: True
Definition: Dominated by

A function $f(n)$ is dominated by $g(n)$ when...

- There exists two constants $c > 0$ and $n_0 > 0$...
- Such that for all values of $n \geq n_0$...
- $f(n) \leq c \cdot g(n)$ is true

Definition: Big-$\mathcal{O}$

$\mathcal{O}(f(n))$ is the “family” or “set” of all functions that are dominated by $f(n)$
Definitions: Dominates

\[ f(n) \in O(g(n)) \] is like saying “\( f(n) \) is less than or equal to \( g(n) \)”.

Is there a way to say “greater than or equal to”?
Definitions: Dominates

\( f(n) \in \mathcal{O}(g(n)) \) is like saying “\( f(n) \) is less then or equal to \( g(n) \)”.

Is there a way to say “greater then or equal to”? Yes!

**Definition: Dominates**

We say \( f(n) \) **dominates** \( g(n) \) when:

- There exists two constants \( c > 0 \) and \( n_0 > 0 \)... 
- Such that for all values of \( n \geq n_0 \)... 
- \( f(n) \geq c \cdot g(n) \) is true
Definitions: Dominates

\[ f(n) \in \mathcal{O}(g(n)) \] is like saying “\( f(n) \) is less then or equal to \( g(n) \)”.

Is there a way to say “greater then or equal to”? Yes!

**Definition: Dominates**

We say \( f(n) \) **dominates** \( g(n) \) when:

- There exists two constants \( c > 0 \) and \( n_0 > 0 \)...  
- Such that for all values of \( n \geq n_0 \)...  
- \( f(n) \geq c \cdot g(n) \) is true

**Definition: Big-Ω**

\( \Omega(f(n)) \) is the family of all functions that **dominates** \( f(n) \).
A few more questions...

True or false?

- $4n^2 \in \Omega(1)$
- $4n^2 \in \Omega(n)$
- $4n^2 \in \Omega(n^2)$
- $4n^2 \in \Omega(n^3)$
- $4n^2 \in \Omega(n^4)$
A few more questions...

<table>
<thead>
<tr>
<th>Question</th>
<th>True or False</th>
</tr>
</thead>
<tbody>
<tr>
<td>$4n^2 \in \Omega(1)$</td>
<td>True</td>
</tr>
<tr>
<td>$4n^2 \in \Omega(n)$</td>
<td>True</td>
</tr>
<tr>
<td>$4n^2 \in \Omega(n^2)$</td>
<td>True</td>
</tr>
<tr>
<td>$4n^2 \in \Omega(n^3)$</td>
<td>False</td>
</tr>
<tr>
<td>$4n^2 \in \Omega(n^4)$</td>
<td>False</td>
</tr>
<tr>
<td>$4n^2 \in O(1)$</td>
<td>False</td>
</tr>
<tr>
<td>$4n^2 \in O(n)$</td>
<td>False</td>
</tr>
<tr>
<td>$4n^2 \in O(n^2)$</td>
<td>True</td>
</tr>
<tr>
<td>$4n^2 \in O(n^3)$</td>
<td>True</td>
</tr>
<tr>
<td>$4n^2 \in O(n^4)$</td>
<td>True</td>
</tr>
</tbody>
</table>
Definition: Big-$\Theta$

We say $f(n) \in \Theta(g(n))$ when both:

- $f(n) \in O(g(n))$ and...
- $f(n) \in \Omega(g(n))$

...are true.

Note: in industry, it's common for many people to ask for the big-$O$ when they really want the big-$\Theta$!
Definition: Big-Θ

We say $f(n) \in \Theta(g(n))$ when both:

- $f(n) \in O(g(n))$ and...
- $f(n) \in \Omega(g(n))$

...are true.

Note: in industry, it’s common for many people to ask for the big-$O$ when they really want the big-$\Theta$!
Exercise: construct a mathematical function modeling the worst-case runtime in terms of $n$.

Assume the println takes $c$ time.

```java
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        System.out.println("Foo!");
    }
}
```
Exercise: construct a mathematical function modeling the worst-case runtime in terms of $n$.

Assume the println takes $c$ time.

```java
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        System.out.println("Foo!");
    }
}
```

A handwavy answer: $T(n) = 0c + 1c + 2c + 3c + \ldots + (n - 1)c$
Exercise: construct a mathematical function modeling the worst-case runtime in terms of $n$.

Assume the println takes $c$ time.

```java
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        System.out.println("Foo!");
    }
}
```

A handwavy answer: $T(n) = 0c + 1c + 2c + 3c + \ldots + (n - 1)c$

A not-handwavy answer: $T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{i-1} c$
Simplifying summations

Strategies:

- Wolfram Alpha
- Apply summation identities

\[ T(n) = \sum_{i=0}^{n-1} i - \sum_{j=0}^{n-1} c = n \sum_{i=0}^{n-1} i - \sum_{i=0}^{n-1} ci \]

Summation of a constant: \[ = c \sum_{i=0}^{n-1} i \]

Factoring out a constant: \[ = c \cdot n \cdot \sum_{i=0}^{n-1} 1 = c n (n - 1) \]

Gauss’s identity

So, \[ T(n) = n - 1 \sum_{i=0}^{n-1} i - \sum_{j=0}^{n-1} c = c^2 n^2 - c^2 n \]
Simplifying summations

Strategies:

▶ Wolfram Alpha
Simplifying summations

Strategies:

- Wolfram Alpha
- Apply summation identities
Simplifying summations

Strategies:

- Wolfram Alpha
- Apply summation identities

\[ T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{i-1} c = \]
Simplifying summations

Strategies:

- Wolfram Alpha
- Apply summation identities

\[
T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{i-1} c = \sum_{i=0}^{n-1} ci
\]

Summation of a constant
Simplifying summations

Strategies:

- Wolfram Alpha
- Apply summation identities

\[
T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{i-1} c = \sum_{i=0}^{n-1} ci \quad \text{Summation of a constant}
\]

\[
= c \sum_{i=0}^{n-1} i \quad \text{Factoring out a constant}
\]
Simplifying summations

Strategies:

- Wolfram Alpha
- Apply summation identities

\[ T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{i-1} c = \sum_{i=0}^{n-1} ci \]  
  Summation of a constant

\[ = c \sum_{i=0}^{n-1} i \]  
  Factoring out a constant

\[ = c \frac{n(n-1)}{2} \]  
  Gauss’s identity
Simplifying summations

Strategies:

- Wolfram Alpha
- Apply summation identities

\[ T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{i-1} c = \sum_{i=0}^{n-1} ci \]

**Summation of a constant**

\[ = c \sum_{i=0}^{n-1} i \]

**Factoring out a constant**

\[ = c \frac{n(n-1)}{2} \]

**Gauss’s identity**

So, \[ T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{i-1} c = \frac{c}{2} n^2 - \frac{c}{2} n \]
Simplifying summations

Strategies:

- Wolfram Alpha
- Apply summation identities

\[ T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{i-1} c = \sum_{i=0}^{n-1} ci \]

Summation of a constant

\[ = c \sum_{i=0}^{n-1} i \]

Factoring out a constant

\[ = c \frac{n(n-1)}{2} \]

Gauss’s identity

So, \[ T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{i-1} c = \frac{c}{2} n^2 - \frac{c}{2} n \]

closed form
Exercise: model the worst-case runtime using summations, find a closed form, find the big-Theta bound.

```java
public void mystery2(int[] arr) {
    for (int i = 5; i < arr.length; i++) {
        int c = 0;
        for (int j = i; j < arr.length; j++) {
            c += arr[j];
        }
        System.out.println(c);
    }
}
```
Exercise: model the worst-case runtime using summations, find a closed form, find the big-Theta bound.

```java
public void mystery2(int[] arr) {
    for (int i = 5; i < arr.length; i++) {
        int c = 0;
        for (int j = i; j < arr.length; j++) {
            c += arr[j];
        }
        System.out.println(c);
    }
}
```

Model: Let $n$ be the array length. Then, $T(n) = \sum_{i=5}^{n-1} \sum_{j=i}^{n-1} 1$
Simplifying summations continued

\[
\sum_{i=5}^{n-1} \sum_{j=i}^{n-1} 1 =
\]
Simplifying summations continued

\[
\sum_{i=5}^{n-1} \sum_{j=i}^{n-1} 1 = \sum_{i=5}^{n-1} \left( \sum_{j=0}^{n-1} 1 - \sum_{j=0}^{i-1} 1 \right)
\]

Normalize lower bound
Simplifying summations continued

\[
\sum_{i=5}^{n-1} \sum_{j=i}^{n-1} 1 = \sum_{i=5}^{n-1} \left( \sum_{j=0}^{n-1} 1 - \sum_{j=0}^{i-1} 1 \right)
\]

Normalize lower bound

\[
= \sum_{i=5}^{n-1} (n - i)
\]

Apply identity
Simplifying summations continued

\[ \sum_{i=5}^{n-1} \sum_{j=i}^{n-1} 1 = \sum_{i=5}^{n-1} \left( \sum_{j=0}^{n-1} 1 - \sum_{j=0}^{i-1} 1 \right) \]

Normalize lower bound

\[ = \sum_{i=5}^{n-1} (n - i) \]

Apply identity

\[ = \sum_{i=0}^{n-1} (n - i) - \sum_{i=0}^{5-1} (n - i) \]

Normalize lower bound
Simplifying summations continued

\[
\sum_{j=0}^{n-1} 1 = \sum_{j=0}^{n-1} \left( \sum_{i=0}^{n-1} 1 - \sum_{j=0}^{i-1} 1 \right)
\]

Normalize lower bound

\[
\sum_{i=0}^{n-1} (n - i) = \sum_{i=0}^{n-1} (n - i) - \sum_{i=0}^{5-1} (n - i)
\]

Apply identity

\[
\sum_{i=0}^{n-1} (n - i) - \sum_{i=0}^{5-1} (n - i) = \sum_{i=0}^{n-1} n - \sum_{i=0}^{n-1} i - \sum_{i=0}^{5-1} n + \sum_{i=0}^{5-1} i
\]

Split summations
Simplifying summations continued

\[
\sum_{i=5}^{n-1} \sum_{j=i}^{n-1} 1 = \sum_{i=5}^{n-1} \left( \sum_{j=0}^{n-1} 1 - \sum_{j=0}^{i-1} 1 \right)
\]

Normalize lower bound

\[
= \sum_{i=5}^{n-1} (n - i)
\]

Apply identity

\[
= \sum_{i=0}^{n-1} (n - i) - \sum_{i=0}^{5-1} (n - i)
\]

Normalize lower bound

\[
= \sum_{i=0}^{n-1} n - \sum_{i=0}^{n-1} i - \sum_{i=0}^{5-1} n + \sum_{i=0}^{5-1} i
\]

Split summations

\[
= n^2 - \frac{n(n - 1)}{2} - 5n + 10
\]

Apply identities
Exercise: model the worst-case runtime of this method.

```java
public static int sum(int[] arr) {
    return sumHelper(0, int[] arr);
}

private static int sumHelper(int curr, int[] arr) {
    if (curr == arr.length) {
        return 0;
    } else {
        return arr[curr] + sumHelper(curr + 1);
    }
}
```

Answer: create a recurrence.

\[
T(n) = \begin{cases} 
  c_1 & \text{when } n = 0 \\
  c_2 + T(n-1) & \text{otherwise} 
\end{cases}
\]

Note: here, \(n\) is the number of items we need to visit, and \(c_1\) and \(c_2\) are some constants.
Handling recursive functions

Exercise: model the worst-case runtime of this method.

```java
public static int sum(int[] arr) {
    return sumHelper(0, arr);
}

private static int sumHelper(int curr, int[] arr) {
    if (curr == arr.length) {
        return 0;
    } else {
        return arr[curr] + sumHelper(curr + 1);
    }
}
```

Answer: create a recurrence.

\[
T(n) = \begin{cases} 
    c_1 & \text{when } n = 0 \\
    c_2 + T(n - 1) & \text{otherwise}
\end{cases}
\]

Note: here, \( n \) is the number of items we need to visit, and \( c_1 \) and \( c_2 \) are some constants.
How do we find a closed form for:

\[ T(n) = \begin{cases} 
  c_1 & \text{when } n = 0 \\
  c_2 + T(n - 1) & \text{otherwise}
\end{cases} \]

One method: the “unfolding” method.
Simplifying recurrences

How do we find a *closed form* for:

\[ T(n) = \begin{cases} 
  c_1 & \text{when } n = 0 \\
  c_2 + T(n - 1) & \text{otherwise} 
\end{cases} \]

One method: the “unfolding” method.

Observation: when \( n = 4 \), \( T(n) = c_2 + (c_2 + (c_2 + (c_2 + c_1))) \)
Simplifying recurrences

How do we find a closed form for:

\[
T(n) = \begin{cases} 
  c_1 & \text{when } n = 0 \\
  c_2 + T(n - 1) & \text{otherwise}
\end{cases}
\]

One method: the “unfolding” method.

Observation: when \( n = 4 \), \( T(n) = c_2 + (c_2 + (c_2 + (c_2 + c_1))) \)

We repeat \( c_2 \) four times, so \( T(4) = 4c_2 + c_1 \).

After generalizing: \( T(n) = c_1 + \sum_{i=0}^{n-1} c_2 = c_1 + c_2 n \).
A dictionary contains a bunch of key-value pairs. Every key is unique (no duplicate keys allowed); the values can be arbitrary. A client can provide a key to look up the corresponding value.
The Dictionary ADT

A dictionary contains a bunch of key-value pairs. Every key is unique (no duplicate keys allowed); the values can be arbitrary. A client can provide a key to look up the corresponding value.

Supported operations:

- **get**: Retrieves the value corresponding to the given key
- **put**: Updates the value corresponding to the given key
- **remove**: Removes the given key (and corresponding value)
- **containsKey**: Returns whether dictionary contains given key
- **size**: Returns the number of key-value pairs
The Dictionary ADT

A dictionary contains a bunch of key-value pairs. Every key is unique (no duplicate keys allowed); the values can be arbitrary. A client can provide a key to look up the corresponding value.

Supported operations:

- **get**: Retrieves the value corresponding to the given key
- **put**: Updates the value corresponding to the given key
- **remove**: Removes the given key (and corresponding value)
- **containsKey**: Returns whether dictionary contains given key
- **size**: Returns the number of key-value pairs

Alternative names: map, lookup table
A set is a collection of items. A set cannot contain any duplicate items: each item must be unique.
The Set ADT

A set is a collection of items. A set cannot contain any duplicate items: each item must be unique.

Supported operations:

- **add**: Adds the given item to the set
- **remove**: Removes the given item to the set
- **contains**: Returns 'true' if the set contains this item
- **size**: Returns the number of items in the set
A set is a collection of items. A set cannot contain any duplicate items: each item must be unique.

Supported operations:

- **add**: Adds the given item to the set
- **remove**: Removes the given item to the set
- **contains**: Returns 'true' if the set contains this item
- **size**: Returns the number of items in the set

Two questions:

1. Do sets (and dictionaries) need to 'order' items in some way?
2. We can implement a set on top of some dictionary: how?
Algorithm design practice: ArrayDictionary

Ex: consider your ArrayDictionary implementation; fill in table:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description of algorithm</th>
<th>Big-Θ bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>get</td>
<td>Scan through the internal array, see if the key exists. Return value if it does.</td>
<td>Θ (n)</td>
</tr>
<tr>
<td>put</td>
<td>Scan through the internal array, replace the value if we find the key-value pair. Otherwise, add the new pair at the end.</td>
<td>Θ (n)</td>
</tr>
<tr>
<td>remove</td>
<td>Scan through the internal array and find the key-value pair. Remove it, and shift over the remaining elements.</td>
<td>Θ (n)</td>
</tr>
<tr>
<td>containsKey</td>
<td>Scan through the array...</td>
<td>Θ (n)</td>
</tr>
</tbody>
</table>
Ex: consider your ArrayDictionary implementation; fill in table:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description of algorithm</th>
<th>Big-$\Theta$ bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>get</td>
<td>Scan through the internal array, see if the key exists. Return value if it does.</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>put</td>
<td>Scan through the internal array, replace the value if we find the key-value pair.</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td></td>
<td>Otherwise, add the new pair at the end.</td>
<td></td>
</tr>
<tr>
<td>remove</td>
<td>Scan through the internal array and find the key-value pair. Remove it, and shift over the remaining elements.</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>containsKey</td>
<td>Scan through the array...</td>
<td>$\Theta(n)$</td>
</tr>
</tbody>
</table>
Idea: exploit additional property of keys

Observation: sometimes, keys are comparable and sortable.
Idea: exploit additional property of keys

Observation: sometimes, keys are comparable and sortable.

Idea: Can we exploit the “sortability” of these keys?
Suppose we add the following **invariant** to ArrayDictionary:

**SortedArrayDictionary invariant**

The internal array, at all times, **must** remain sorted.

How do you implement get? What’s the big-$\Theta$ bound?
The binary search algorithm

Core algorithm (in *pseudocode*):

```java
public V get(K key):
    return search(key, 0, this.size)

private K search(K key, int lowIndex, int highIndex):
    if lowIndex > highIndex:
        key not found, throw an exception
    else:
        middleIndex = average of lowIndex and highIndex
        pair = this.array[middleIndex]
        if pair.key == key:
            return pair.value
        else if pair.key < key:
            return search(key, lowIndex, middleIndex)
        else if pair.key > key:
            return search(key, middleIndex + 1, highIndex)
```

\[T(n) \approx \begin{cases} 
1 & \text{if } n \leq 0 \\text{c} + T(n/2) & \text{otherwise}
\end{cases} \]
The binary search algorithm

Core algorithm (in *pseudocode*):

```java
public V get(K key):
    return search(key, 0, this.size)

private K search(K key, int lowIndex, int highIndex):
    if lowIndex > highIndex:
        key not found, throw an exception
    else:
        middleIndex = average of lowIndex and highIndex
        pair = this.array[middleIndex]

        if pair.key == key:
            return pair.value
        else if pair.key < key:
            return search(key, lowIndex, middleIndex)
        else if pair.key > key:
            return search(key, middleIndex + 1, highIndex)
```

Ex: model the worst-case runtime. Assume the time needed to compare two keys takes $c$ time. Let $n =$???
The binary search algorithm

Core algorithm (in *pseudocode*):

```java
public V get(K key):
    return search(key, 0, this.size)

private K search(K key, int lowIndex, int highIndex):
    if lowIndex > highIndex:
        key not found, throw an exception
    else:
        middleIndex = average of lowIndex and highIndex
        pair = this.array[middleIndex]
        if pair.key == key:
            return pair.value
        else if pair.key < key:
            return search(key, lowIndex, middleIndex)
        else if pair.key > key:
            return search(key, middleIndex + 1, highIndex)
```

Ex: model the worst-case runtime. Assume the time needed to compare two keys takes \( c \) time. Let \( n = \text{highIndex} - \text{lowIndex} \).
The binary search algorithm

Core algorithm (in \textit{pseudocode}):  

\begin{verbatim}
public V get(K key):
    return search(key, 0, this.size)

private K search(K key, int lowIndex, int highIndex):
    if lowIndex > highIndex:
        key not found, \textbf{throw} an exception
    else:
        middleIndex = average of lowIndex and highIndex
        pair = this.array[middleIndex]
        if pair.key == key:
            return pair.value
        else if pair.key < key:
            return search(key, lowIndex, middleIndex)
        else if pair.key > key:
            return search(key, middleIndex + 1, highIndex)
\end{verbatim}

Answer: \(T(n) \approx \begin{cases} 
1 & \text{When } n \leq 0 \\
 c + T \left( \frac{n}{2} \right) & \text{Otherwise}
\end{cases}\)
Finding a closed form

Our answer: $T(n) \approx \begin{cases} 1 & \text{When } n \leq 0 \\ c + T \left( \frac{n}{2} \right) & \text{Otherwise} \end{cases}$

Question: how do we find a closed form?

Try unfolding:

$T(n) = c + (c + (c + \ldots + c + 1)) = c + c + \ldots + c + 1$

$t = \text{Num times}$

$t = 0, 2, 4, 6, 8, 10, 12, 16, \ldots, 32, \ldots, 64$

What's the relationship?

$n \approx 2^t + 1$

Solve for $t$:

$t \approx \log(n) - 1$

Final model:

$T(n) \approx c \left( \log(n) - 1 \right) + 1$

So, we conclude:

$T(n) \in \Theta \left( \log(n) \right)$
Finding a closed form

Our answer: \( T(n) \approx \begin{cases} 
1 & \text{When } n \leq 0 \\
c + T\left(\frac{n}{2}\right) & \text{Otherwise}
\end{cases}\)

Question: how do we find a closed form? Try unfolding?

\[
T(n) = c + (c + (c + \ldots + (c + 1))),
\]

where \( t \) is the Num times

\[
t = \log(n) - 1
\]

Final model:

\[
T(n) \approx c(\log(n) - 1) + 1
\]
Finding a closed form

Our answer: \( T(n) \approx \begin{cases} 1 & \text{When } n \leq 0 \\ c + T\left(\frac{n}{2}\right) & \text{Otherwise} \end{cases} \)

Question: how do we find a closed form? Try unfolding?

\[
T(n) = c + (c + (c + \ldots + (c + 1))) = \underbrace{c + c + \ldots + c}_\text{Num times} + 1
\]
Finding a closed form

Our answer: \( T(n) \approx \begin{cases} 
1 & \text{When } n \leq 0 \\
\quad c + T(\frac{n}{2}) & \text{Otherwise}
\end{cases} \)

Question: how do we find a closed form? Try unfolding?

\[
T(n) = c + (c + (c + \ldots + (c + 1))) = \underbrace{c + c + \ldots + c + 1}_{t=\text{Num times}}
\]

<table>
<thead>
<tr>
<th>( n )</th>
<th>0</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>16</th>
<th>\ldots</th>
<th>32</th>
<th>\ldots</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td>\quad</td>
<td>\quad</td>
<td>\quad</td>
<td>\quad</td>
<td>\quad</td>
<td>\quad</td>
<td>\quad</td>
<td>\quad</td>
<td>\quad</td>
<td>\quad</td>
<td>\quad</td>
<td>\quad</td>
</tr>
<tr>
<td>( t )</td>
<td>\quad</td>
<td>\quad</td>
<td>\quad</td>
<td>\quad</td>
<td>\quad</td>
<td>\quad</td>
<td>\quad</td>
<td>\quad</td>
<td>\quad</td>
<td>\quad</td>
<td>\quad</td>
<td>\quad</td>
</tr>
</tbody>
</table>
Finding a closed form

Our answer: \( T(n) \approx \begin{cases} 
1 & \text{When } n \leq 0 \\
c + T\left(\frac{n}{2}\right) & \text{Otherwise}
\end{cases} \)

Question: how do we find a closed form? Try unfolding?

\[
T(n) = c + (c + (c + \ldots + (c + 1))) = c + c + \ldots + c + 1
\]

t=\text{Num times}

\[
\begin{array}{c|cccccccccccc}
 n & 0 & 2 & 4 & 6 & 8 & 10 & 12 & 16 & \ldots & 32 & \ldots & 64 \\
\hline
 t & 0 & 2 & 3 & 3 & 4 & 4 & 4 & 5 & \ldots & 6 & \ldots & 7
\end{array}
\]
Finding a closed form

Our answer: \( T(n) \approx \begin{cases} 
1 & \text{When } n \leq 0 \\
c + T \left( \frac{n}{2} \right) & \text{Otherwise} 
\end{cases} \)

Question: how do we find a closed form? Try unfolding?

\[
T(n) = c + (c + (c + \ldots + (c + 1))) = c + c + \ldots + c + 1
\]

\( t = \text{Num times} \)

<table>
<thead>
<tr>
<th>( n )</th>
<th>0</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>16</th>
<th>...</th>
<th>32</th>
<th>...</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>...</td>
<td>6</td>
<td>...</td>
<td>7</td>
</tr>
</tbody>
</table>

What’s the relationship?
Finding a closed form

Our answer: \( T(n) \approx \begin{cases} 
1 & \text{When } n \leq 0 \\
\frac{n}{2} + T\left(\frac{n}{2}\right) & \text{Otherwise}
\end{cases} \)

Question: how do we find a closed form? Try unfolding?

\[
T(n) = c + (c + (c + \ldots + (c + 1))) = c + c + \ldots + c + 1
\]

\( t = \text{Num times} \)

<table>
<thead>
<tr>
<th>( n )</th>
<th>0</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>16</th>
<th>...</th>
<th>32</th>
<th>...</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>...</td>
<td>6</td>
<td>...</td>
<td>7</td>
</tr>
</tbody>
</table>

What’s the relationship? \( n \approx 2^{t+1} \)
Finding a closed form

Our answer:  \( T(n) \approx \begin{cases} 
1 & \text{When } n \leq 0 \\
c + T\left(\frac{n}{2}\right) & \text{Otherwise}
\end{cases} \)

Question: how do we find a closed form? Try unfolding?

\[
T(n) = c + (c + (c + \ldots + (c + 1))) = c + c + \ldots + c + 1
\]

\( t = \text{Num times} \)

<table>
<thead>
<tr>
<th>( n )</th>
<th>0</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>16</th>
<th>...</th>
<th>32</th>
<th>...</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>...</td>
<td>6</td>
<td>...</td>
<td>7</td>
</tr>
</tbody>
</table>

What's the relationship?  \( n \approx 2^{t+1} \)

Solve for \( t \):
Finding a closed form

Our answer: \( T(n) \approx \begin{cases} 1 & \text{When } n \leq 0 \\ c + T\left(\frac{n}{2}\right) & \text{Otherwise} \end{cases} \)

Question: how do we find a closed form? Try unfolding?

\[
T(n) = c + (c + (c + \ldots + (c + 1))) = c + c + \ldots + c + 1
\]
\[
t = \text{Num times}
\]

<table>
<thead>
<tr>
<th>( n )</th>
<th>0</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>16</th>
<th>\ldots</th>
<th>32</th>
<th>\ldots</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>\ldots</td>
<td>6</td>
<td>\ldots</td>
<td>7</td>
</tr>
</tbody>
</table>

What’s the relationship? \( n \approx 2^{t+1} \)

Solve for \( t \): \( t \approx \log(n) - 1 \)
Finding a closed form

Our answer: $T(n) \approx \begin{cases} 1 & \text{When } n \leq 0 \\ c + T\left(\frac{n}{2}\right) & \text{Otherwise} \end{cases}$

Question: how do we find a closed form? Try unfolding?

$$T(n) = c + (c + (c + \ldots + (c + 1))) = \underbrace{c + c + \ldots + c + 1}_{t=\text{Num times}}$$

<table>
<thead>
<tr>
<th>$n$</th>
<th>0</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>16</th>
<th>...</th>
<th>32</th>
<th>...</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>...</td>
<td>6</td>
<td>...</td>
<td>7</td>
</tr>
</tbody>
</table>

What’s the relationship? $n \approx 2^{t+1}$

Solve for $t$: $t \approx \log(n) - 1$

Final model: $T(n) \approx c(\log(n) - 1) + 1$
Finding a closed form

Our answer: \[ T(n) \approx \begin{cases} 
1 & \text{When } n \leq 0 \\
1 + T\left(\frac{n}{2}\right) & \text{Otherwise}
\end{cases} \]

Question: how do we find a closed form? Try unfolding?

\[ T(n) = c + (c + (c + \ldots + (c + 1))) = \underbrace{c + c + \ldots + c + 1}_{t=\text{Num times}} \]

<table>
<thead>
<tr>
<th>(n)</th>
<th>0</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>16</th>
<th>\ldots</th>
<th>32</th>
<th>\ldots</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t)</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>\ldots</td>
<td>6</td>
<td>\ldots</td>
<td>7</td>
</tr>
</tbody>
</table>

What's the relationship? \( n \approx 2^{t+1} \)

Solve for \( t \): \( t \approx \log(n) - 1 \)

Final model: \[ T(n) \approx c(\log(n) - 1) + 1 \]
Finding a closed form

Our answer: \( T(n) \approx \begin{cases} 1 & \text{When } n \leq 0 \\ c + T\left(\frac{n}{2}\right) & \text{Otherwise} \end{cases} \)

Question: how do we find a closed form? Try unfolding?

\[
T(n) = c + (c + (c + \ldots + (c + 1))) = c + c + \ldots + c + 1
\]

\[
t = \text{Num times}
\]

<table>
<thead>
<tr>
<th>( n )</th>
<th>0</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>16</th>
<th>\ldots</th>
<th>32</th>
<th>\ldots</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>\ldots</td>
<td>6</td>
<td>\ldots</td>
<td>7</td>
</tr>
</tbody>
</table>

What’s the relationship? \( n \approx 2^{t+1} \)

Solve for \( t \): \( t \approx \log(n) - 1 \)

Final model: \( T(n) \approx c(\log(n) - 1) + 1 \)

So, we conclude: \( T(n) \in \Theta(\log(n)) \)
Fill in the remainder of this table for SortedArrayDictionary:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description of algorithm</th>
<th>Big-$\Theta$ bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>get</td>
<td>Use binary search.</td>
<td>$\Theta (\log(n))$</td>
</tr>
<tr>
<td>put</td>
<td></td>
<td></td>
</tr>
<tr>
<td>remove</td>
<td></td>
<td></td>
</tr>
<tr>
<td>containsKey</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fill in the remainder of this table for SortedArrayDictionary:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description of algorithm</th>
<th>Big-$\Theta$ bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>get</td>
<td>Use binary search.</td>
<td>$\Theta (\log(n))$</td>
</tr>
<tr>
<td>put</td>
<td>Use binary search to find key.</td>
<td>$\Theta (n)$</td>
</tr>
<tr>
<td></td>
<td>If it doesn’t exist, insert into array.</td>
<td></td>
</tr>
<tr>
<td>remove</td>
<td>Use binary search to find key.</td>
<td>$\Theta (n)$</td>
</tr>
<tr>
<td></td>
<td>Once found, remove it and shift over remaining elements.</td>
<td></td>
</tr>
<tr>
<td>containsKey</td>
<td>Use binary search.</td>
<td>$\Theta (\log(n))$</td>
</tr>
</tbody>
</table>