Warmup questions

**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
- $n^2 \in O(n)$ True
- $n^2 \in O(n^2)$ True
- $n^2 \in O(n^3)$ True
- $n^2 \in O(n^{100})$ True

---

O is $5n + 3$ dominated by $n$

- True
- True
- Should be false, but is “true”
- True
- False
- False
- F
- T
- T
Warmup questions

**Warmup:** True or false:

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

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

Is there a way to say “greater than or equal to”?
\( f(n) \in \mathcal{O}(g(n)) \) 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 than or equal to \( g(n) \)”.

Is there a way to say “greater than 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: \text{Big-Ω}**

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

True or false?

- $4n^2 \in \Omega(1)$  
  - True

- $4n^2 \in \Omega(n)$  
  - True

- $4n^2 \in \Omega(n^2)$  
  - False

- $4n^2 \in \Omega(n^3)$  
  - True

- $4n^2 \in \Omega(n^4)$  
  - False
True or false?

- $4n^2 \in \Omega(1)$ True
- $4n^2 \in \Omega(n)$ True
- $4n^2 \in \Omega(n^2)$ True
- $4n^2 \in \Omega(n^3)$ False
- $4n^2 \in \Omega(n^4)$ False

- $4n^2 \in O(1)$ False
- $4n^2 \in O(n)$ False
- $4n^2 \in O(n^2)$ True
- $4n^2 \in O(n^3)$ True
- $4n^2 \in O(n^4)$ True
**Definition: Big-Θ**

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

1. \( f(n) \in \mathcal{O}(g(n)) \) and...
2. \( f(n) \in \Omega(g(n)) \)

...are true.

Note: In industry, it's common for many people to ask for the big-\( \mathcal{O} \) 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!";
    }
}
```

A handwavy answer:

$$T(n) = 0c + 1c + 2c + 3c + \ldots + (n-1)c$$

A not-handwavy answer:

$$T(n) = n-1 \sum_{i=0}^{i=n-1} i - \sum_{j=0}^{j=i} c$$
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: $T(n) = 0c + 1c + 2c + 3c + \ldots + (n - 1)c$
Modeling complex loops

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

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

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

Factoring out a constant:
\[ \sum_{i=0}^{n-1} c = c \left( \frac{n(n-1)}{2} \right) \]

Gauss's identity:
So, 
\[ T(n) = \sum_{i=0}^{n-1} i - \sum_{j=0}^{n-1} c = cn - \frac{n^2}{2} - \left( n - 1 \right) c \]

Closed form solution:
Simplifying summations

Strategies:

▶ Wolfram Alpha

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

Strategies:

▶ Wolfram Alpha
▶ Apply summation identities
Simplifying summations

Strategies:

- Wolfram Alpha
- Apply summation identities

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

Summation of a constant
Factoring out a constant
Gauss's identity

So,

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

Closed form
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

\[
c + c + c + \cdots = c(0+1+2+\cdots)
\]
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
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
- Factoring out a constant
- Gauss’s identity

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

\[
= c \frac{n(n-1)}{2}
\]
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);
    }
}
```
Simplifying summations

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

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

\[ \sum_{i=5}^{n-1} (n-1) - \sum_{i=5}^{n-1} (i-1) + 10 \]

Apply identities
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

\[
\begin{align*}
\sum_{i=5}^{n-1} \sum_{j=i}^{n-1} 1 &= \sum_{i=5}^{n-1} \left( \sum_{j=0}^{n-1} 1 - \sum_{j=0}^{i-1} 1 \right) \\
&= \sum_{i=5}^{n-1} (n - i) \\
&= \sum_{i=0}^{n-1} (n - i) - \sum_{i=0}^{5-1} (n - i)
\end{align*}
\]

Normalize lower bound

Apply identity

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

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

\[ T(4) = c_2 + (c_2 + T(3 - 1)) \]
\[ = c_2 + (c_2 + (c_2 + T(2 - 1))) \]
\[ = c_2 + (c_2 + (c_2 + (c_2 + T(1 - 1)))) \]

After generalizing:

\[ T(n) = c_1 + n - 1 \sum_{i=0}^{n-1} c_2 = c_1 + c_2 n. \]
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 \( n = 4 \), \( T(n) = c_2 + (c_2 + (c_2 + (c_2 + c_1))) \)
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.
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
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.
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
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

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>
**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-$\Theta$ bound</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>get</strong></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><strong>put</strong></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><strong>remove</strong></td>
<td>Scan through the internal array and find the key-value pair. Remove it, and shift over</td>
<td>$\Theta (n)$</td>
</tr>
<tr>
<td></td>
<td>the remaining elements.</td>
<td></td>
</tr>
<tr>
<td><strong>containsKey</strong></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.
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-Θ bound?

Look up "m"
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)
```

\[
T(n) = \begin{cases} 
1 & \text{when } n = 0 \\
 \frac{n}{2} + T\left(\frac{n}{2}\right) & \text{otherwise}
\end{cases}
\]
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 $n =$???
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 \( n = highIndex - lowIndex \).
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)

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

\[
T(n) = c + T\left(\frac{n}{2}\right) \\
= c + (c + T\left(\frac{n}{4}\right)) \\
= c + c + (c + T\left(\frac{n}{8}\right)) \\
\vdots
\]

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:

\( T(n) \in \Theta(\log(n)) \)
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?

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

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

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

<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>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ n \approx 2^t \]

\[ t \approx \log(n) \]

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

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

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

<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>
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?
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? \( n \approx 2^{t+1} \)
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?

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

<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 \( t \): \( t \approx \log(n) - 1 \)
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 \( t \): \( t \approx \log(n) - 1 \)

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

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

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>Use binary search to find key. If it doesn't exist, insert into array.</td>
<td>$\Theta (n)$</td>
</tr>
<tr>
<td>remove</td>
<td>Use binary search to find key. Once found, remove it and shift over remaining elements.</td>
<td>$\Theta (n)$</td>
</tr>
<tr>
<td>containsKey</td>
<td>Use binary search.</td>
<td>$\Theta (\log(n))$</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><strong>get</strong></td>
<td>Use binary search.</td>
<td>$\Theta (\log(n))$</td>
</tr>
<tr>
<td><strong>put</strong></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><strong>remove</strong></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><strong>containsKey</strong></td>
<td>Use binary search.</td>
<td>$\Theta (\log(n))$</td>
</tr>
</tbody>
</table>
Idea: Moving away from lists

Observation: *Changing* our array is still difficult
Idea: Moving away from lists

Observation: Changing our array is still difficult

Idea: Use a different data structure optimized for both searching and insertion?
Idea: Moving away from lists

Observation: Changing our array is still difficult

Idea: Use a different data structure optimized for both searching and insertion?

Answer: Use a Binary Search Tree (BST)