CSE 250 Recitation

February 20 - 21: ADTs, Amortized Runtime

List ADT

Discussion: Could these two methods be part of a valid implementation of the List ADT (assume the rest of the methods are defined consistently with these)

```
add(v):
  node = new node(v)
  node->next = head
  head = node
```

```
get(idx):
  curr = last, i = 0
  while i < idx:
    i = i + 1
    curr = curr->prev
  return curr.value
```

List ADT

Discussion: Could these two methods be part of a valid implementation of the List ADT (assume the rest of the methods are defined consistently with these)

Yes! Consider a user that adds 1, 2, 3, 4.

```
get(0)->1
get(1)->2
get(2)->3
get(3)->4
```

```
add(v):
  node = new node(v)
  node->next = head
  head = node
```

```
get(idx):
  curr = last, i = 0
  while i < idx:
    i = i + 1
    curr = curr->prev
  return curr.value
```

Set ADT

Exercise: Describe an implementation of the Set ADT using your **`SortedList`** implementation from PA1.

Reminder: The methods of the Set ADT are **`add`**, **`contains`**, **`remove`**

`add(elem)`: Adds elem to the set if it is not already present in the set

`contains(elem)`: returns true if elem is in the set, false otherwise

`remove(elem):` removes elem and returns true, otherwise returns false

Set ADT

Discussion:

How does this implementation differ than the one from lecture?

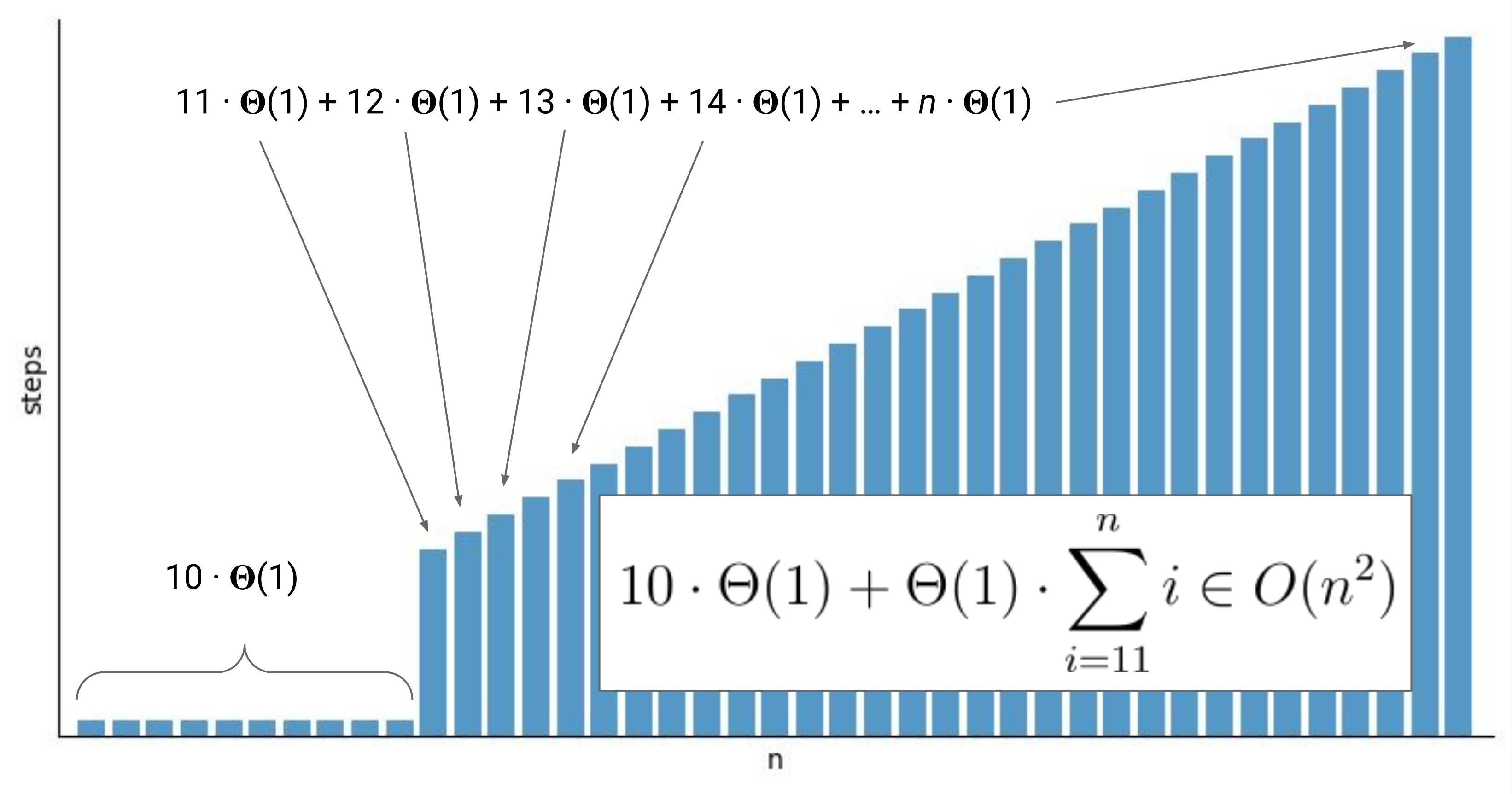
What differs when we implement Bag?

```
SortedList data

add(elem):
  if !data.findRef(elem).isPresent():
    data.insert(elem)

contains(elem):
  return data.findRef(elem).isPresent()

remove(elem):
  node = data.findRef(elem)
  if node.isPresent():
    data.remove(node)
    return true
  return false
```

Amortized Runtime

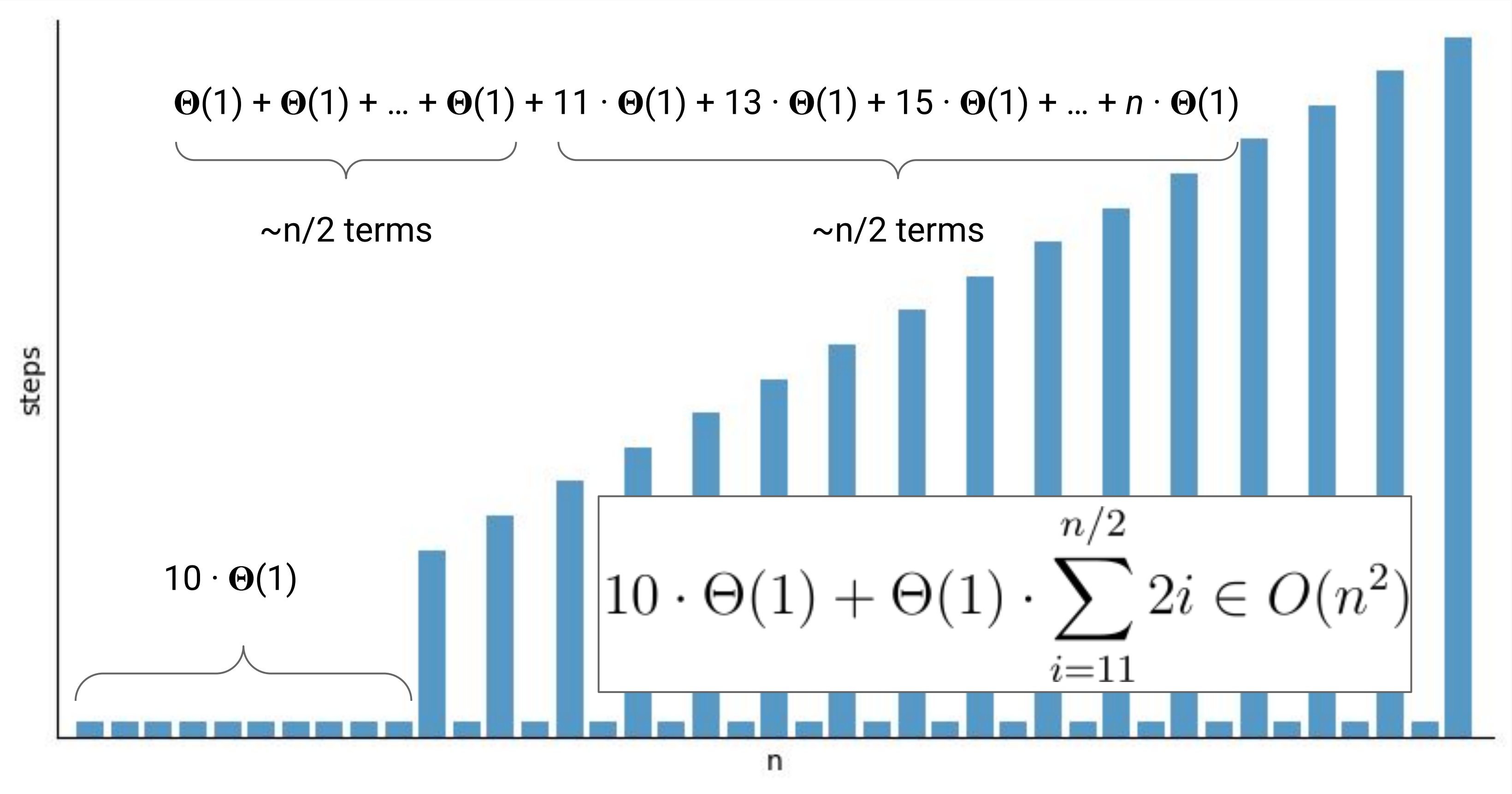
If $\boldsymbol{n}$ calls to a function take $\boldsymbol{\Theta(f(n))}$ steps

Then the **amortized runtime** of that function is $\boldsymbol{\Theta(f(n)/n)}$



$$10 \cdot \Theta(1) + \Theta(1) \cdot \sum_{i=11}^{n} i \in O(n^2)$$

Cost of each call when the initial array size is 10, and newLength = data.length + 1

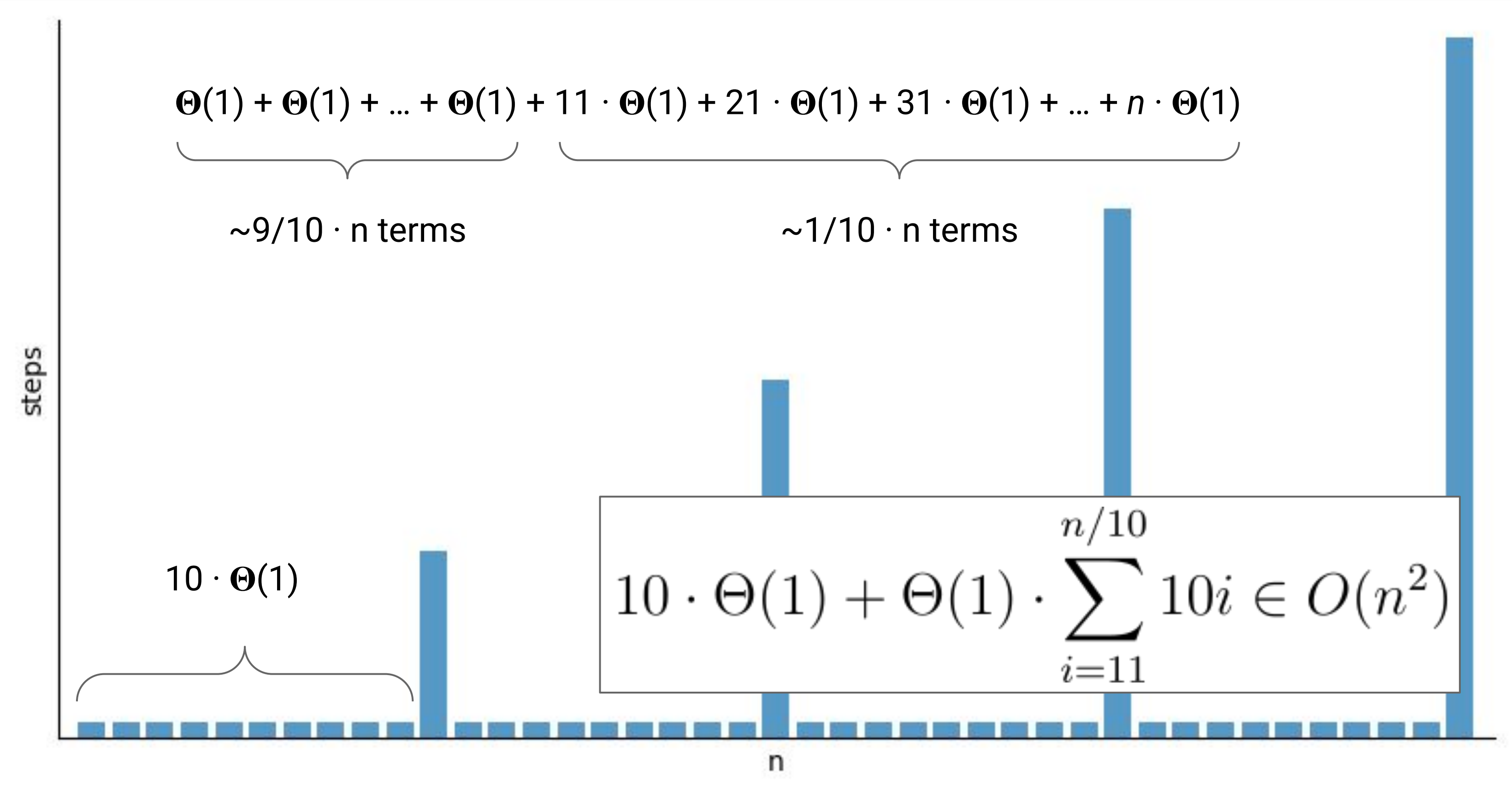


$$\underbrace{\Theta(1) + \Theta(1) + \ldots + \Theta(1)}_{\sim n/2 \text{ terms}} + \underbrace{11 \cdot \Theta(1) + 13 \cdot \Theta(1) + 15 \cdot \Theta(1) + \ldots + n \cdot \Theta(1)}_{\sim n/2 \text{ terms}}$$

$10 \cdot \Theta(1)$

$$\boxed{10 \cdot \Theta(1) + \Theta(1) \cdot \sum_{i=11}^{n/2} 2i \in O(n^2)}$$

Cost of each call when the initial array size is 10, and newLength = data.length + 2



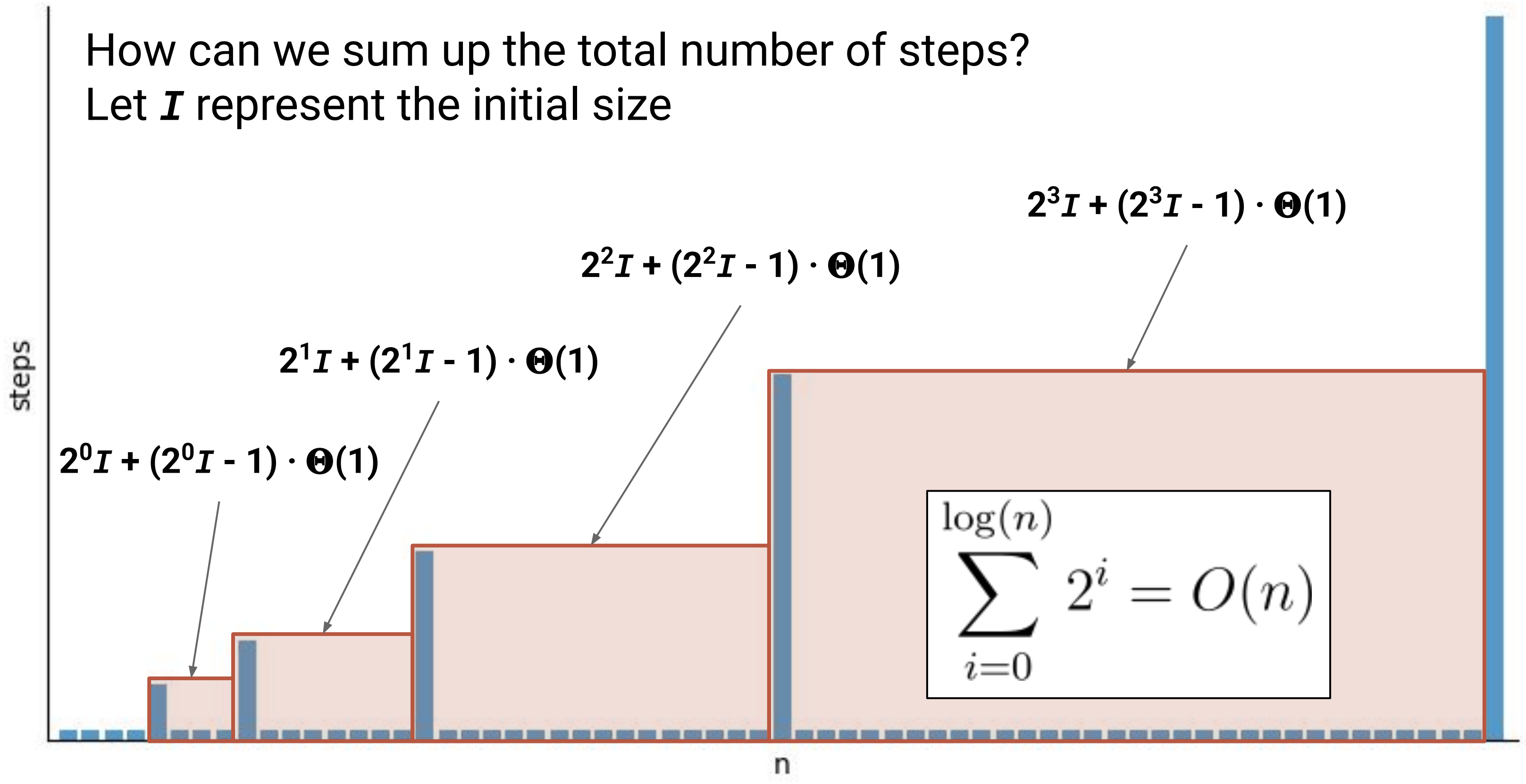
$$\underbrace{\Theta(1) + \Theta(1) + \ldots + \Theta(1)}_{\sim 9/10 \cdot \text{n terms}} + \underbrace{11 \cdot \Theta(1) + 21 \cdot \Theta(1) + 31 \cdot \Theta(1) + \ldots + n \cdot \Theta(1)}_{\sim 1/10 \cdot \text{n terms}}$$

$10 \cdot \Theta(1)$

$$10 \cdot \Theta(1) + \Theta(1) \cdot \sum_{i=11}^{n/10} 10i \in O(n^2)$$

Cost of each call when the initial array size is 10, and newLength = data.length + 10

How can we sum up the total number of steps?
Let I represent the initial size



$2^0 I + (2^0 I - 1) \cdot \Theta(1)$

$2^1 I + (2^1 I - 1) \cdot \Theta(1)$

$2^2 I + (2^2 I - 1) \cdot \Theta(1)$

$2^3 I + (2^3 I - 1) \cdot \Theta(1)$

$$\sum_{i=0}^{\log(n)} 2^i = O(n)$$

Cost of each call when the initial array size is 4, and newLength = data.length x 2

Amortized Runtime Analysis

```java
public class Team {
  private List<Player> players;

  public void addPlayer(Player p) { /* ... */ }
  public void importRoster(File f) { /* ... */ }
  /* ... */
}
```

```java
public void addPlayer(Player p) {
  System.out.println("Welcome to the team " + p.name());
  players.add(p);
}
```

Exercise: What are the unqualified and amortized runtime bounds of the **`addPlayer`** method when **`List`** is a **`LinkedList`**? an **`ArrayList`**?

Amortized Runtime Analysis

```
public class Team {
  private List<Player> players;

  public void addPlayer(Player p) { /*       */ }
  public void importRoster(File
  /* ... */
}
```

	players is LinkedList	players is ArrayList
addPlayer runtime	Unqualified **Θ**(1) Amortized **Θ**(1)	Unqualified ***O***(***n***) Amortized **Θ**(1)

```
public void addPlayer(Player p) {
  System.out.println("Welcome to the team " + p.name());
  players.add(p);
}
```

Exercise: What are the unqualified and amortized runtime bounds of the **addPlayer** method when **List** is a **LinkedList**? an **ArrayList**?

Amortized Runtime Analysis

```java
public void importRoster(File f) {
  BufferedReader br = new BufferedReader(new FileReader(f));
  String line;
  while (br.ready()) {
    String line = br.readLine();
    Player p = new Player(line);
    addPlayer(p);
  }
}
```

	players is LinkedList	players is ArrayList
addPlayer runtime	Unqualified $\Theta(1)$ Amortized $\Theta(1)$	Unqualified $O(n)$ Amortized $\Theta(1)$

Exercise: What are the unqualified and amortized runtime bounds of the **`importRoster`** method when **`List`** is a **`LinkedList`**? an **`ArrayList`**?

*(You can assume that opening the file, reading a line, and creating a **`Player`** are constant-time calls)*

Amortized Runtime Analysis

```
public void importRoster(File f) {
  BufferedReader br = new BufferedReader(new FileReader(f));
  String line;
  while (br.ready()) {
    String line = br.readLine();
    Player p = new Player(line);
    addPlayer(p);
  }
}
```

Exercise: What are t
`importRoster` met

	`players` is `LinkedList`	`players` is `ArrayList`
`addPlayer` runtime	Unqualified **Θ**(1) Amortized **Θ**(1)	Unqualified ***O***(***n***) Amortized **Θ**(1)
`importRoster runtime`	Unqualified **Θ**(***n***) Amortized **Θ**(***n***)	Unqualified **Θ**(***n***) Amortized **Θ**(***n***)

(You can assume that opening the file, reading a line, and creating a ***`Player`*** *are constant-time calls)*