CSE 250
Data Structures

Dr. Eric Mikida
epmikida@buffalo.edu
Capen 208

Lecture 13
Binary Search

Announcements

- WA2 out now, due Sunday @ 11:59PM
- WA1 grades have been released
- Midterm info and practice exams will be posted by tonight
- PA1 has completed
  - Amnesty policy reminder
  - Imposter Syndrome

Sets and Bags (...so far)

	`LinkedList`	`ArrayList`
`Set.add`	$O(n)$	$O(n)$
`Set.contains`	$O(n)$	$O(n)$
`Set.remove`	$O(n)$	$\Theta(n)$
`Bag.add`	$O(1)$	$O(n)$, Amortized $\Theta(1)$
`Bag.contains`	$\Theta(n)$	$\Theta(n)$
`Bag.remove`	$O(n)$	$\Theta(n)$

Potential Improvements

Does order matter for `Sets`/`Bags`?

Potential Improvements

Does order matter for `Sets`/`Bags`?

No! **`Sets`** and **`Bags`** are unordered...
...so we are not restricted to storing data in any particular order!

Can we take advantage of this?

Set Implementation (w/ArrayList)

```
ArrayList<T> data


remove(elem):
  idx = 0
  while idx < data.size():
    if data[idx] == elem:
      data.remove(idx)
      return true
    idx = idx + 1
  return false
```

Runtime: $\Theta(n)$

Set Implementation (w/ArrayList)

```
ArrayList<T> data


remove(elem):
  idx = 0
  while idx < data.size():
    if data[idx] == elem:
      data.remove(idx)
      return true
    idx = idx + 1
  return false
```

Runtime: $\Theta(n)$

What makes `remove` so expensive?

Set Implementation (w/ArrayList)

```
ArrayList<T> data


remove(elem):
  idx = 0
  while idx < data.size():
    if data[idx] == elem:
      data.remove(idx)
      return true
    idx = idx + 1
  return false
```

Runtime: $\Theta(n)$

What makes `remove` so expensive?

We have to shift over elements to fill the hole *while keeping everything in the same order*!

Can we improve on this?

Set Implementation (w/ArrayList)

```
ArrayList<T> data


remove(elem):
  idx = 0, last = data.size()-1
  while idx < data.size():
    if data[idx] == elem:
      data[idx] = data[last]
      data.remove(last)
      return true
    idx = idx + 1
  return false
```

Runtime: $\Theta(n)$

What makes `remove` so expensive?

We have to shift over elements to fill the hole *while keeping everything in the same order*!

Can we improve on this?

Fill the hole with the last element!

Set Implementation (w/ArrayList)

```
ArrayList<T> data

remove(elem):
  idx = 0, last = data.size()-1
  while idx < data.size():
    if data[idx] == elem:
      data[idx] = data[last]
      data.remove(last)
      return true
    idx = idx + 1
  return false
```

Runtime: ~~$\Theta(n)$~~ $O(n), \Omega(1)$

Runtime now becomes $O(n), \Omega(1)$

What makes `remove` so expensive?

We have to shift over elements to fill the hole *while keeping everything in the same order*!

Can we improve on this?

Fill the hole with the last element!

Going Further

This was just a minor improvement...can we go further?

Going Further

This was just a minor improvement...can we go further?

First we need to understand more about searching...

Searching by Value

Consider searching for a value in an `Array` (or `ArrayList`)...

How long does that search take?

Searching by Value

Consider searching for a value in an `Array` (or `ArrayList`)...

How long does that search take?

- We may find our target immediately: $\Omega(1)$
- We may have to check every element (one at at time): $O(n)$

Searching by Value

Consider searching for a value in an `Array` (or `ArrayList`)...

How long does that search take?

- We may find our target immediately: $\Omega(1)$
- We may have to check every element (one at at time): $O(n)$

This is called a **Linear Search** (it takes linear time)

Searching by Value

Consider searching for a value in an `Array` (or `ArrayList`)...

How long does that search take?

- We may find our target immediately: $\Omega(1)$
- We may have to check every element (one at at time): $O(n)$

This is called a **Linear Search** (it takes linear time)

Could we improve on this if we knew our list had certain characteristics?

What if it was sorted?

Binary Search

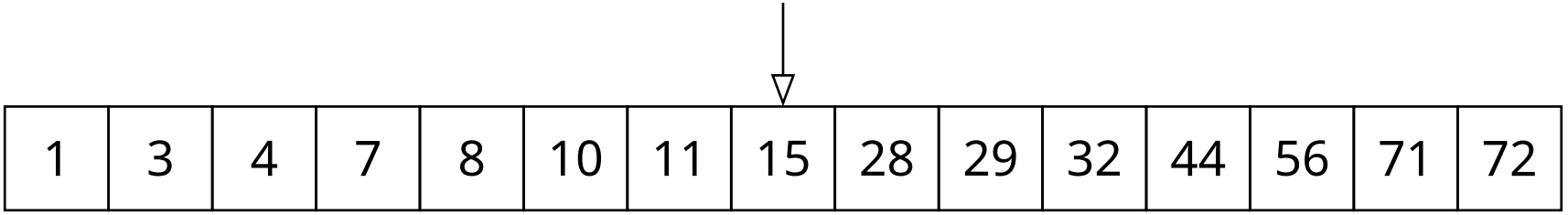
1	3	4	7	8	10	11	15	28	29	32	44	56	71	72

Searching For: 56
Current Index:

Imagine searching this (or a much larger) **`Array`** for a value, ie 56.

Where would ***you*** start?

Binary Search

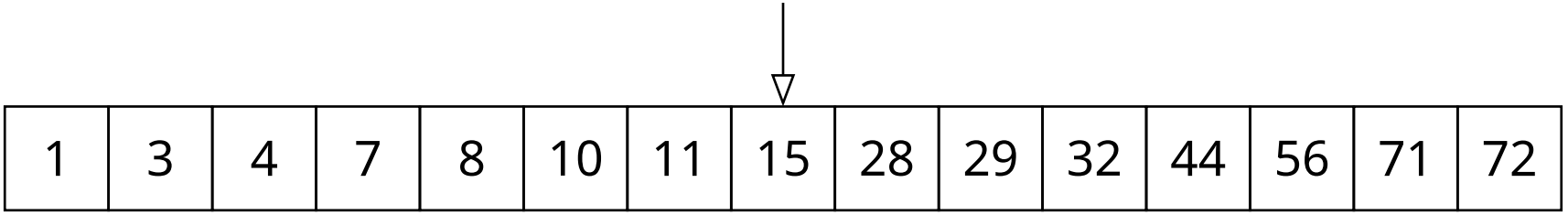


Searching For: 56
Current Index: 7

Imagine searching this (or a much larger) **`Array`** for a value, ie 56.

Where would ***you*** start? The middle!

Binary Search



Searching For: 56

Current Index: 7

What parts of the array can you rule out?

Binary Search

Searching For: 56

Current Index:

What parts of the array can you rule out? The whole left half!

Binary Search

Searching For: 56

Current Index: 11

Now just repeat on this new, smaller array!

Binary Search

Searching For: 56

Current Index: 13

Now just repeat on this new, smaller array!

Binary Search

Searching For: 56
Current Index: 12

Now just repeat on this new, smaller array!

FOUND IT!

Binary Search Runtime

What is the runtime of binary search?

Binary Search Runtime

What is the runtime of binary search?

We cut the search space in half each time...What is the worst case scenario?

(AKA how many times might we have to halve the space?)

Binary Search Runtime

What is the runtime of binary search?

We cut the search space in half each time...What is the worst case scenario?

(AKA how many times might we have to halve the space?)

At most $\log(n)$ times!

Linear vs Binary Search

Linear Search:

- Removes 1 element from consideration each step, $O(n)$
- Does not require list to be sorted
- Does not require constant time random access

Binary Search:

- Removes half of the elements from consideration each step, $O(\log(n))$
- Requires list to be sorted
- Requires constant time random access (binary search on a **`LinkedList`** is $O(n)$)

$O(n)$ vs $O(\log(n))$

How big is the difference between $O(n)$ and $O(\log(n)$?

Imagine you have a sorted `Array` of size 1024:

- How many steps would a linear search take?

$O(n)$ vs $O(\log(n))$

How big is the difference between $O(n)$ and $O(\log(n)$?

Imagine you have a sorted `Array` of size 1024:

- How many steps would a linear search take? **1024** (in the worst case)
- How many steps would a binary search take?

$O(n)$ vs $O(\log(n))$

How big is the difference between $O(n)$ and $O(\log(n)$?

Imagine you have a sorted `Array` of size 1024:

- How many steps would a linear search take? **1024** (in the worst case)
- How many steps would a binary search take? **10** (in the worst case)

That's a pretty big difference! ...but there's more...

$O(n)$ vs $O(\log(n))$

How big is the difference between $O(n)$ and $O(\log(n)$?

Imagine you have a sorted `Array` of size 1024:

- How many steps would a linear search take? **1024** (in the worst case)
- How many steps would a binary search take? **10** (in the worst case)

That's a pretty big difference! ...but there's more...

Imagine we just doubled the size of our `Array` to hold 2048 elements:

- Now how many steps does linear search take?

$O(n)$ vs $O(\log(n))$

How big is the difference between $O(n)$ and $O(\log(n)$?

Imagine you have a sorted `Array` of size 1024:

- How many steps would a linear search take? **1024** (in the worst case)
- How many steps would a binary search take? **10** (in the worst case)

That's a pretty big difference! ...but there's more...

Imagine we just doubled the size of our `Array` to hold 2048 elements:

- Now how many steps does linear search take? **2048** (literally double the steps!)
- What about binary search?

$O(n)$ vs $O(\log(n))$

How big is the difference between $O(n)$ and $O(\log(n)$?

Imagine you have a sorted `Array` of size 1024:

- How many steps would a linear search take? **1024** (in the worst case)
- How many steps would a binary search take? **10** (in the worst case)

That's a pretty big difference! ...but there's more...

Imagine we just doubled the size of our `Array` to hold 2048 elements:

- Now how many steps does linear search take? **2048** (literally double the steps!)
- What about binary search? **11** (only added one extra step!)

$O(\log(n)$ scales MUCH better than $O(n)$

Implications for Set

Can we take advantage of Binary Search in our Set/Bag implementations?

Implications for Set

Can we take advantage of Binary Search in our Set/Bag implementations?

Let's store our data in a sorted **`ArrayList`**!

Set Implementation w/Sorted `ArrayList`

`boolean contains(elem):`

1. Use binary search to find **`elem`**	$O(\log(n))$, $\Omega(1)$

Total: $O(\log(n))$, $\Omega(1)$

Our **`contains`** implementation can now just use Binary Search, great!

What about **`add`**/**`remove`**?

Set Implementation w/Sorted `ArrayList`

`void add(elem):`

1. Use binary search to find index of **`elem`**	$O(\log(n))$, $\Omega(1)$
2. If **`elem`** is not there, insert it at that index	$O(n)$, $\Omega(1)$

Total: $O(n)$, $\Omega(1)$

Binary Search requires data to always be sorted!

- We can find the insertion point quickly 😊
- We need to shift elements over to make room and maintain order 😔

Set Implementation w/Sorted `ArrayList`

`boolean remove(elem):`

1. Use binary search to find index of **`elem`**	$O(\log(n))$, $\Omega(1)$
2. If **`elem`** is there, remove it by index	$O(n)$, $\Omega(1)$

Total: $O(n)$, $\Omega(???)$

Binary Search requires data to always be sorted!

- We can find the element to remove quickly 😊
- We need to shift elements over to fill the hole and maintain order 😔

What is the best case runtime?

Set Implementation w/Sorted `ArrayList`

`boolean remove(elem):`

1. Use binary search to find index of **`elem`**	$O(\log(n))$, $\Omega(1)$
2. If **`elem`** is there, remove it by index	$O(n)$, $\Omega(1)$

Total: $O(n)$, $\Omega(\log(n))$

Binary Search requires data to always be sorted!

- We can find the element to remove quickly 😊
- We need to shift elements over to fill the hole and maintain order 😔

What is the best case runtime? $\Omega(\log(n))$

- If we find the element fast, we have to shift over more elements
- If the element is near the end, it will take $\log(n)$ steps to find it

Sets and Bags (...so far)

	`LinkedList`	`ArrayList`	`ArrayList` *(sorted)*
`Set.add`	$O(n)$	$O(n)$	$O(n)$
`Set.contains`	$O(n)$	$O(n)$	$O(\log(n))$
`Set.remove`	$O(n)$	$O(n)$	$O(n)$
`Bag.add`	$O(1)$	$O(n)$, Amortized $\Theta(1)$	???
`Bag.contains`	$\Theta(n)$	$\Theta(n)$	???
`Bag.remove`	$O(n)$	$O(n)$	???

What about the **Bag** methods?

Sets and Bags (...so far)

	`LinkedList`	`ArrayList`	`ArrayList` *(sorted)*
`Set.add`	$O(n)$	$O(n)$	$O(n)$
`Set.contains`	$O(n)$	$O(n)$	$O(\log(n))$
`Set.remove`	$O(n)$	$O(n)$	$O(n)$
`Bag.add`	$O(1)$	$O(n)$, Amortized $\Theta(1)$	$O(n)$
`Bag.contains`	$\Theta(n)$	$\Theta(n)$	$O(n)$
`Bag.remove`	$O(n)$	$O(n)$	$O(n)$

Only change is that **`contains`** needs to count the number of occurrences

Which ADT/Data Structure is Best?

Scenario #1: Users logging into an online game need to be added to a login queue in the order they log in. From time to time you must also check the whole queue for any users that may have left so you can clear them out.

Which ADT/Data Structure is Best?

Scenario #1: Users logging into an online game need to be added to a login queue in the order they log in. From time to time you must also check the whole queue for any users that may have left so you can clear them out.

Step 1: Do **NOT** think about runtimes yet! First we have to pick an ADT.

- What operations do we need to solve the problem?
- What characteristics do we need the data to have?

Which ADT/Data Structure is Best?

Scenario #1: Users logging into an online game need to be added to a login queue in the order they log in. From time to time you must also check the whole queue for any users that may have left so you can clear them out.

Step 1: Do **NOT** think about runtimes yet! First we have to pick an ADT.

- What operations do we need to solve the problem? append, remove
- What characteristics do we need the data to have? ordered, variable size

ADT:

Which ADT/Data Structure is Best?

Scenario #1: Users logging into an online game need to be added to a login queue in the order they log in. From time to time you must also check the whole queue for any users that may have left so you can clear them out.

Step 1: Do **NOT** think about runtimes yet! First we have to pick an ADT.

- What operations do we need to solve the problem? append, remove
- What characteristics do we need the data to have? ordered, variable size

ADT: `List`

- We need to be able to add/remove data, so we cannot use **`Sequence`**
- Order matters, so we cannot use **`Set`** or **`Bag`**

Which ADT/Data Structure is Best?

Scenario #1: Users logging into an online game need to be added to a login queue in the order they log in. From time to time you must also check the whole queue for any users that may have left so you can clear them out.

Step 2: Choose the Data Structure that best implements the needed operations

- Append:
- Remove:

Which ADT/Data Structure is Best?

Scenario #1: Users logging into an online game need to be added to a login queue in the order they log in. From time to time you must also check the whole queue for any users that may have left so you can clear them out.

Step 2: Choose the Data Structure that best implements the needed operations

- Append: **`ArrayList`** is $O(n)$, **`LinkedList`** is $O(1)$
- Remove: **`ArrayList`** is $O(n)$, **`LinkedList`** is $O(1)$, since index is alway 0

Data Structure:

Which ADT/Data Structure is Best?

Scenario #1: Users logging into an online game need to be added to a login queue in the order they log in. From time to time you must also check the whole queue for any users that may have left so you can clear them out.

Step 2: Choose the Data Structure that best implements the needed operations

- Append: **`ArrayList`** is $O(n)$, **`LinkedList`** is $O(1)$
- Remove: **`ArrayList`** is $O(n)$, **`LinkedList`** is $O(1)$, since index is alway 0

Data Structure: **`LinkedList`**

- Both of our required operations are asymptotically faster

Which ADT/Data Structure is Best?

Scenario #2: You need to read in the lines of a CSV file, store them in a list, and later access them based on their original line number.

Which ADT/Data Structure is Best?

Scenario #2: You need to read in the lines of a CSV file, store them in a list, and later access them based on their original line number.

Step 1: Do **NOT** think about runtimes yet! First we have to pick an ADT.

- What operations do we need to solve the problem?
- What characteristics do we need the data to have?

Which ADT/Data Structure is Best?

Scenario #2: You need to read in the lines of a CSV file, store them in a list, and later access them based on their original line number.

Step 1: Do **NOT** think about runtimes yet! First we have to pick an ADT.

- What operations do we need to solve the problem? append, access by position
- What characteristics do we need the data to have? ordered, variable size

ADT:

Which ADT/Data Structure is Best?

Scenario #2: You need to read in the lines of a CSV file, store them in a list, and later access them based on their original line number.

Step 1: Do **NOT** think about runtimes yet! First we have to pick an ADT.

- What operations do we need to solve the problem? append, access by position
- What characteristics do we need the data to have? ordered, variable size

ADT: **`List`**

- We need to be able to add data, so we cannot use **`Sequence`**
- Order matters, so we cannot use **`Set`** or **`Bag`**

Which ADT/Data Structure is Best?

Scenario #2: You need to read in the lines of a CSV file, store them in a list, and later access them based on their original line number.

Step 2: Choose the Data Structure that best implements the needed operations

- Append:
- Access by Index:

Which ADT/Data Structure is Best?

Scenario #2: You need to read in the lines of a CSV file, store them in a list, and later access them based on their original line number.

Step 2: Choose the Data Structure that best implements the needed operations

- Append: **`ArrayList`** is $O(n)$, Amortized $\Theta(1)$, **`LinkedList`** is $O(1)$
- Access by Index: **`ArrayList`** is $O(1)$, **`LinkedList`** is $O(n)$

Data Structure:

Which ADT/Data Structure is Best?

Scenario #2: You need to read in the lines of a CSV file, store them in a list, and later access them based on their original line number.

Step 2: Choose the Data Structure that best implements the needed operations

- Append: **`ArrayList`** is $O(n)$, Amortized $\Theta(1)$, **`LinkedList`** is $O(1)$
- Access by Index: **`ArrayList`** is $O(1)$, **`LinkedList`** is $O(n)$

Data Structure: **`ArrayList`**

- We perform many adds in a row, and the amortized runtime of both **`ArrayList`** and **`LinkedList`** is $\Theta(1)$
- **`ArrayLists`** give much faster access by index

Which ADT/Data Structure is Best?

Scenario #3: You are trying to keep track of all movies in your ever-growing collection. You would like to be able to store data about your collection in such a way that you can quickly determine if you have a given movie.

Which ADT/Data Structure is Best?

Scenario #3: You are trying to keep track of all movies in your ever-growing collection. You would like to be able to store data about your collection in such a way that you can quickly determine if you have a given movie.

Step 1: Do **NOT** think about runtimes yet! First we have to pick an ADT.

- What operations do we need to solve the problem?
- What characteristics do we need the data to have?

Which ADT/Data Structure is Best?

Scenario #3: You are trying to keep track of all movies in your ever-growing collection. You would like to be able to store data about your collection in such a way that you can quickly determine if you have a given movie.

Step 1: Do **NOT** think about runtimes yet! First we have to pick an ADT.

- What operations do we need to solve the problem? add, search
- What characteristics do we need the data to have? unique, search by value

ADT:

Which ADT/Data Structure is Best?

Scenario #3: You are trying to keep track of all movies in your ever-growing collection. You would like to be able to store data about your collection in such a way that you can quickly determine if you have a given movie.

Step 1: Do **NOT** think about runtimes yet! First we have to pick an ADT.

- What operations do we need to solve the problem? add, search
- What characteristics do we need the data to have? unique, search by value

ADT: Set

- Search by value, not position – don't need to maintain order
- Unique elements

Which ADT/Data Structure is Best?

Scenario #3: You are trying to keep track of all movies in your ever-growing collection. You would like to be able to store data about your collection in such a way that you can quickly determine if you have a given movie.

Step 2: Choose the Data Structure that best implements the needed operations

- Add:
- Search by Value:

Which ADT/Data Structure is Best?

Scenario #3: You are trying to keep track of all movies in your ever-growing collection. You would like to be able to store data about your collection in such a way that you can quickly determine if you have a given movie.

Step 2: Choose the Data Structure that best implements the needed operations

- Add: **`ArrayList`** is $O(n)$, **`LinkedList`** is $O(n)$
- Search by Value: **`ArrayList`** is $O(n)$ but $O(\log(n))$ if sorted, **`LinkedList`** is $O(n)$

Data Structure:

Which ADT/Data Structure is Best?

Scenario #3: You are trying to keep track of all movies in your ever-growing collection. You would like to be able to store data about your collection in such a way that you can quickly determine if you have a given movie.

Step 2: Choose the Data Structure that best implements the needed operations

- Add: **`ArrayList`** is $O(n)$, **`LinkedList`** is $O(n)$
- Search by Value: **`ArrayList`** is $O(n)$ but $O(\log(n))$ if sorted, **`LinkedList`** is $O(n)$

Data Structure: Sorted **`ArrayList`**

- We can take advantage of Binary Search to find movies quickly
- Adding is slow, but scenario focused on fast search