Lesson 27: List Operations (W08D2)

Balboa High School

Michael Ferraro

October 6, 2015

Do Now

Let's play a game! I have taken the clubs out of a deck of playing cards.

2♣	3♣	4♣	5♣	6♣	7♣	8♣	9♣	10♣	J♣	Q♣	K♣	A♣

Do Now

Let's play a game! I have taken the clubs out of a deck of playing cards.

2♣	3♣	4♣	5♣	6♣	7♣	8♣	9♣	10♣	J♣	Q♣	K♣	A♣

I now shuffle the 13 cards above and arrange them like elements in a list, but this time face down.

?♣	?♣	?♣	?♣	?♣	?♣	?♣	?♣	?♣	?♣	?♣	?♣	?♣

Do Now

Let's play a game! I have taken the clubs out of a deck of playing cards.

2♣	3♣	4♣	5♣	6♣	7♣	8♣	9♣	10♣	J♣	Q♣	K♣	A♣

I now shuffle the 13 cards above and arrange them like elements in a list, but this time face down.

?♣	?♣	?♣	?♣	?♣	?♣	?♣	?♣	?♣	?♣	?♣	?♣	?♣

---

1. If you start flipping cards over, from left to right, how many cards, on average, would you need to flip over to find any particular card you might be looking for?
2. Would your answer to #1 change if you were to start flipping over cards randomly (i.e., not in a left-to-right manner)?

Aim

Students will learn about two kinds of list search operations:

- Sequential Search
- Binary Search

Why do you use. . . ?

Think of your favorite Internet search engine. Why do you favor it over others that are available?

Why is *searching* important?

- Most useful programs, web-based or otherwise, have to find information that you need.

Why is *searching* important?

- Most useful programs, web-based or otherwise, have to find information that you need.
- Is that which you need ultimately stored in a database? The programmers of that database system are acutely aware of search problems!

Why is *searching* important?

- Most useful programs, web-based or otherwise, have to find information that you need.
- Is that which you need ultimately stored in a database? The programmers of that database system are acutely aware of search problems!
- Algorithms need to behave predictably, and search algorithms are no exception. Today we learn how to predict how two search methods work.

Sequential Search

- When we searched cards from left to right, we were performing a **sequential search**

Sequential Search

- When we searched cards from left to right, we were performing a **sequential search**
- Assuming that there is only one of what you're looking (e.g., we want the $\boxed{4\clubsuit}$ card and there's only one such card in our deck), and there are n elements in the list, it'll take $\approx \frac{n}{2}$ accesses to find the desired element — on average

Sequential Search

- When we searched cards from left to right, we were performing a **sequential search**
- Assuming that there is only one of what you're looking (e.g., we want the $\boxed{4\clubsuit}$ card and there's only one such card in our deck), and there are n elements in the list, it'll take $\approx \frac{n}{2}$ accesses to find the desired element — on average
  - sometimes you're lucky and find it the first time

Sequential Search

- When we searched cards from left to right, we were performing a **sequential search**
- Assuming that there is only one of what you're looking (e.g., we want the $\boxed{4\clubsuit}$ card and there's only one such card in our deck), and there are n elements in the list, it'll take $\approx \frac{n}{2}$ accesses to find the desired element — on average
  - sometimes you're lucky and find it the first time
  - and sometimes you're really unlucky and have to search every element

Sequential Search

- When we searched cards from left to right, we were performing a **sequential search**
- Assuming that there is only one of what you're looking (e.g., we want the $\boxed{4\clubsuit}$ card and there's only one such card in our deck), and there are n elements in the list, it'll take $\approx \frac{n}{2}$ accesses to find the desired element — on average
  - sometimes you're lucky and find it the first time
  - and sometimes you're really unlucky and have to search every element
- But what if you're looking for an element you're not even sure is present? How many accesses before you know that it's not in the list?

Is there a better way?

- If a list is not in any kind of order (i.e., the list is shuffled or randomized), are there any techniques that are better than a sequential search?

Is there a better way?

- If a list is not in any kind of order (i.e., the list is shuffled or randomized), are there any techniques that are better than a sequential search?

  Not really for unordered lists

Is there a better way?

- If a list is not in any kind of order (i.e., the list is shuffled or randomized), are there any techniques that are better than a sequential search?

  Not really for unordered lists

- What if the list is in some kind of order, however?

Is there a better way?

- If a list is not in any kind of order (i.e., the list is shuffled or randomized), are there any techniques that are better than a sequential search?

  Not really for unordered lists

- What if the list is in some kind of order, however?
- New Game: I'm thinking of an integer between 1 and 100, inclusive. What's the number?

Is there a better way?

- If a list is not in any kind of order (i.e., the list is shuffled or randomized), are there any techniques that are better than a sequential search?

  Not really for unordered lists

- What if the list is in some kind of order, however?
- New Game: I'm thinking of an integer between 1 and 100, inclusive. What's the number?

  $\rightarrow$ *What strategy is best in this case?*

  $\rightarrow$ *What does "best" mean here?*

Binary Search

Searching through a list of 100 possible answers, what's the maximum number of guesses you'd need to make using the *binary search* strategy?

Binary Search

Sample guessing session:

Binary Search

Sample guessing session:

- Is the number 50? *Nope! Try higher...*

Binary Search

Sample guessing session:

- Is the number 50? *Nope! Try higher...*
- Is the number 75? *Nope! Lower...*

Binary Search

Sample guessing session:

- Is the number 50? *Nope! Try higher...*
- Is the number 75? *Nope! Lower...*
- Is the number 63? *Nope! Lower...*

Binary Search

Sample guessing session:

- Is the number 50? *Nope! Try higher...*
- Is the number 75? *Nope! Lower...*
- Is the number 63? *Nope! Lower...*
- Is the number 56? *Nope! Higher...*

Binary Search

Sample guessing session:

- Is the number 50? *Nope! Try higher...*
- Is the number 75? *Nope! Lower...*
- Is the number 63? *Nope! Lower...*
- Is the number 56? *Nope! Higher...*
- Is the number 60? *Nope! Lower...*

Binary Search

Sample guessing session:

- Is the number 50? *Nope! Try higher...*
- Is the number 75? *Nope! Lower...*
- Is the number 63? *Nope! Lower...*
- Is the number 56? *Nope! Higher...*
- Is the number 60? *Nope! Lower...*
- Is the number 58? *Nope! Higher...*

Binary Search

Sample guessing session:

- Is the number 50? *Nope! Try higher...*
- Is the number 75? *Nope! Lower...*
- Is the number 63? *Nope! Lower...*
- Is the number 56? *Nope! Higher...*
- Is the number 60? *Nope! Lower...*
- Is the number 58? *Nope! Higher...*
- Is the number 59? **Yes!**

This worst-case scenario required 7 guesses!

Binary Search

Sample guessing session:

- Is the number 50? *Nope! Try higher...*
- Is the number 75? *Nope! Lower...*
- Is the number 63? *Nope! Lower...*
- Is the number 56? *Nope! Higher...*
- Is the number 60? *Nope! Lower...*
- Is the number 58? *Nope! Higher...*
- Is the number 59? **Yes!**

This worst-case scenario required 7 guesses!

Using sequential search, the worst-case scenario would be 100 guesses!

Binary Search

- There were 100 elements in the list we searched

Binary Search

- There were 100 elements in the list we searched
- Bound 100 between consecutive powers of 2

Binary Search

- There were 100 elements in the list we searched
- Bound 100 between consecutive powers of 2
  - $64 < 100 < 128$

Binary Search

- There were 100 elements in the list we searched
- Bound 100 between consecutive powers of 2
  - $64 < 100 < 128$
  - $2^6 < 100 < 2^7$

Binary Search

- There were 100 elements in the list we searched
- Bound 100 between consecutive powers of 2
  - $64 < 100 < 128$
  - $2^6 < 100 < 2^7$
- $\therefore$ 7 is the worst-case number of guesses (list accesses)

Binary Search

- What if there's an ordered list of 1500 elements — how many accesses, maximum, are required to find an element?

Binary Search

- What if there's an ordered list of 1500 elements — how many accesses, maximum, are required to find an element?
- $2^? < 1500 < 2^{?+1}$

2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1024
–	**1500**
2^{11}	2048

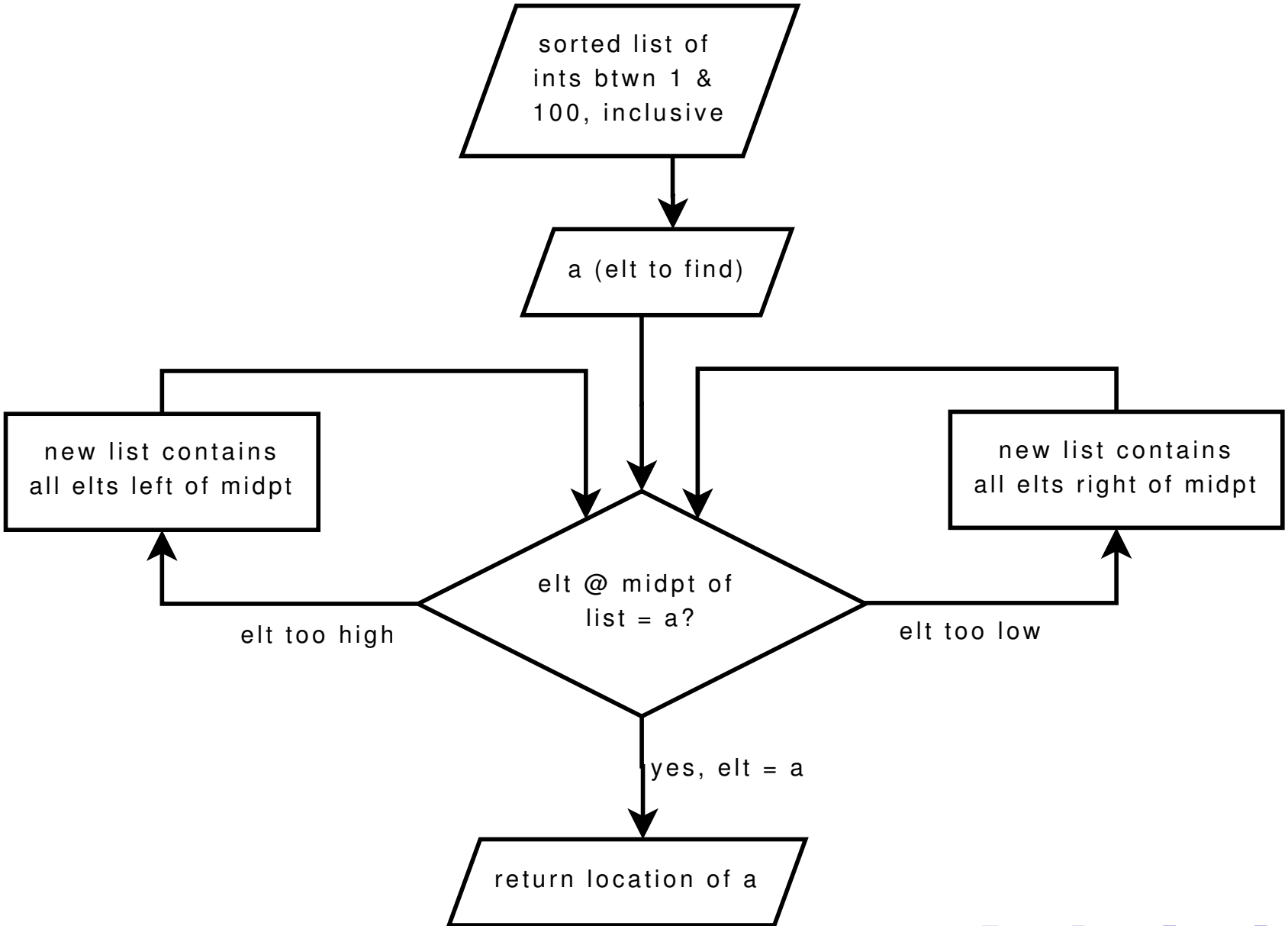
Binary Search

- What if there's an ordered list of 1500 elements — how many accesses, maximum, are required to find an element?
- $2^{?} < 1500 < 2^{?+1}$

2^5	32
2^6	64
2^7	128
2^8	256
2^9	512
2^{10}	1024
–	**1500**
2^{11}	2048

- $\therefore$ 11 accesses, max

Binary Search



Search Scenarios

Assume we're searching through a list of n elements.

[†] formula: $\lceil log_2 n \rceil$ (the brackets "$\lceil\ \rceil$" represent the *ceiling* function, which rounds up to the next integer)

Search Scenarios

Assume we're searching through a list of n elements.

- Best case for both search methods: You find what you're looking for the first time, so 1 list access

[†] formula: $\lceil log_2 n \rceil$ (the brackets "$\lceil\ \rceil$" represent the *ceiling* function, which rounds up to the next integer)

Search Scenarios

Assume we're searching through a list of n elements.

- Best case for both search methods: You find what you're looking for the first time, so 1 list access
- Worst case performance
  - **sequential search:** n accesses
  - **binary search:** a, where a is the lowest integer power of 2 such that $2^a \geq n$ †

† formula: $\lceil log_2 n \rceil$ (the brackets "⌈ ⌉" represent the *ceiling* function, which rounds up to the next integer)

Search Scenarios

Assume we're searching through a list of n elements.

- Best case for both search methods: You find what you're looking for the first time, so 1 list access
- Worst case performance
  - **sequential search:** n accesses
  - **binary search:** a, where a is the lowest integer power of 2 such that $2^a \geq n$ †
- Average case performance
  - **sequential search:** $\approx \frac{n}{2}$
  - **binary search:** ???

† formula: $\lceil log_2 n \rceil$ (the brackets "$\lceil\ \rceil$" represent the *ceiling* function, which rounds up to the next integer)

Experiment: Find Avg Case

What is the average number of accesses needed for a *sequential search*?

Experiment: Find Avg Case

What is the average number of accesses needed for a *sequential search*?

- We already stated that it's $\approx \frac{n}{2}$ accesses; this was our *intuition*

Experiment: Find Avg Case

What is the average number of accesses needed for a *sequential search*?

- We already stated that it's $\approx \frac{n}{2}$ accesses; this was our *intuition*
- A mathematician would need to write a proof to back up such a conjecture

Experiment: Find Avg Case

What is the average number of accesses needed for a *sequential search*?

- We already stated that it's $\approx \frac{n}{2}$ accesses; this was our *intuition*
- A mathematician would need to write a proof to back up such a conjecture
- A computer scientist may also write a proof, but (s)he could also use a lot of empirical evidence — that which (s)he observes — to make a claim in which (s)he has confidence

Experiment: Find Avg Case

What is the average number of accesses needed for a *sequential search*?

- We already stated that it's $\approx \frac{n}{2}$ accesses; this was our *intuition*
- A mathematician would need to write a proof to back up such a conjecture
- A computer scientist may also write a proof, but (s)he could also use a lot of empirical evidence — that which (s)he observes — to make a claim in which (s)he has confidence

$\rightarrow$ *How might a program work that is designed to figure out the average case for a list of 10 elements? (We are to assume that the list is not in any particular order.)*

Experiment: Find Avg Case

- Download `SequentialSearch.java` from here and import into new project `Lesson27`. **DO NOT RUN THIS PROGRAM YET!**
- With a partner, read through the code, starting in `main()`. Your job: Figure out my approach to demonstrating the average number of list accesses needed for a sequential search.

Experiment: Find Avg Case

- Role play: Let's choose students to play the part of these methods:
  - `main()`
  - `populateList()`
  - `shuffleArrOfInts()`
  - `sequentialSearch()`
  - `reportAvgNumItems()`

Experiment: Find Avg Case

- Let's run this program!
- Modifying # of trials to increase accuracy...

HW

- §6 of PS #4b (make sure you do the required reading!)
- The next quiz is on 10/13 (one class day before PS#4b is due)