CS 106
Introduction to Computer Science I

07 / 14 / 2026

Instructor: Michael Eckmann

Today's Topics

- Questions / comments?
- Review searching
  - Best case and worst case of
    - Linear search
    - Binary search
- Sorting

Searching lists (Linear search)

- n = length of the list (the number of elements in list)
- Let's analyze the linear search.
- How many comparisons does it take to find the value or to be able to say it is not found?
  - BEST CASE: What's the minimum number of comparisons it would take?
  - WORST CASE: What's the maximum number of comparisons it would take?

Searching lists (Linear search)

- n = length of the list (the number of elements in list)
- Let's analyze the linear search.
- How many comparisons does it take to find the value or to be able to say it is not found?
  - BEST CASE: **1 compare**
    - Happens when the key is found in the first slot
  - WORST CASE: **n compares**
    - Happens when the key is not found (or it is found in the last slot)

Searching lists (Binary search)

- Let's analyze the binary search. Recall, it works only if the list is sorted low to high before doing binary search.
- To simplify the discussion, we can count the 2 comparisons in the if/elif/else together to be 1 comparison.
- How many comparisons does it take to find the value or to be able to say it is not found?
  - BEST CASE: What's the minimum number of comparisons it would take?
  - WORST CASE: What's the maximum number of comparisons it would take?

Searching lists (Binary search)

- Let's analyze the binary search.
- How many comparisons does it take to find the value or to be able to say it is not found?
  - BEST CASE: **1 compare**
    - Happens when the key is found in the middle of the list (the first place binary search looks)
  - WORST CASE: about **lg(n) compares**
    - Happens when the key is not found (or when it is found at the last place binary search looks)

Searching lists

	Best Case	Worst Case
Linear Search	1	n
Binary Search	1	lg(n)

Nested loops

- Let’s make sure we understand how a loop within a loop would execute

Sorting

- Sorting is simply putting a list of items in some order (the meaning of an item being less than another item is necessary).
- There are a myriad of ways to sort. We'll discuss several algorithms for doing this.
- We’ll assume the data we want to sort is stored in a list.
- Sorting can be done in ascending or descending order.

Sorting

- Original unsorted: 38, 41, 22, 12, 67
- The result of sorting should be: 12, 22, 38, 41, 67
- To accomplish this, one way would be to:
  - Go through the whole list once and find the lowest value.
  - Take it out and put it in a new list.
  - Go through the remaining list of numbers and find the lowest
  - Take that one out and put it at the end of the new list.
  - And so on ... until there's nothing left in the list.
- Does everyone agree that this will achieve a sorted list.
- We didn't say how exactly we'd find the lowest value each time --- the next slide describes a different way to sort and is more detailed.

Sorting (BubbleSort)

- An algorithm for sorting is the BubbleSort:
  - Compares adjacent numbers in the list
  - Swaps them if the two numbers are not in ascending order
  - Compare each adjacent pair of numbers in the first pass, swapping when necessary.
  - Next pass, start at first again but go only up until the next to last element, and so on…
  - Do n – 1 passes, where n is length of the list
  - what's true after first pass?

Sorting (BubbleSort)

- Original unsorted: 38, 41, 22, 12, 67

- compare 38 to 41. 38 is not > 41, so leave them in place.
- compare 41 to 22. Swap them. So, now list is 38, 22, 41, 12, 67
- compare 41 to 12. Swap them. So, now list is 38, 22, 12, 41, 67
- compare 41 to 67. Leave them.

- After first pass: 38, 22, 12, 41, 67

Sorting (BubbleSort)

- After first pass: 38, 22, 12, 41, 67

- Now start at first position again.
- compare 38 to 22. Swap them. So, now list is 22, 38, 12, 41, 67
- compare 38 to 12. Swap them. So, now list is 22, 12, 38, 41, 67
- compare 38 to 41. Leave them.
- (don't need to compare the 4th and 5th positions) – why???
- After second pass: 22, 12, 38, 41, 67

Sorting (BubbleSort)

- After second pass: 22, 12, 38, 41, 67

- Now start at first position again.

- compare 22 to 12. Swap them. So, now list is 12, 22, 38, 41, 67
- compare 22 to 38. Leave them.

- After third pass: 12, 22, 38, 41, 67

Sorting (BubbleSort)

- After third pass: 12, 22, 38, 41, 67
- Now start at first position again.
- compare 12 to 22. Leave them.
- Done.
- After fourth pass: 12, 22, 38, 41, 67

Sorting

- Here's a page that shows some sorting algorithms graphically.

- https://math.hws.edu/eck/js/sorting/xSortLab.html

Sorting

Let’s implement BubbleSort

Then

We’ll implement SelectionSort

Sorting

Let’s figure out how to count up the number of comparisons done in each of those sorting algorithms by adding counters in the code

Can count up how many copies also