CS 106
Introduction to Computer Science I

07 / 16 / 2021

Instructor: Michael Eckmann

Today's Topics

- Questions / comments?
- Searching
  - Linear search
  - Binary search
- Sorting

Searching

Given an item and a list, determine whether the item is in the list or not.

Linear search: compare the item to the first element of the list (if not ==), compare to the second and so on, until we find it (return True) or we go through the entire list and don't find it (return False)

This is like the contains function from lab

Searching lists (Linear search)

- 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

Suppose we have a sorted list (from low to high) can we make fewer comparisons for search in the worst case and on average?

Where's the best place to start your compares?

Searching

Binary search

- Given a sorted list and a key to look for
- Compare the key to the middle element of the list
  - If it is there, return True
  - If key < that middle element, only focus on the left half
  - If key > that middle element, only focus on the right half

  - Do the same thing we did to the whole list where it says to focus on half
  - return False when there is no sublist to search

Binary Search

- Any idea how we might write code to implement this algorithm?

- Let's discuss some ideas before we get right to the code.
  - What parameters might our function have?
  - What element to compare to first?
    - How do we calculate that index?
  - How do we determine what part of the array to now do a search?
- Let's take a look at an implementation and do an example call.

Binary Search

```
# function to perform binary search of a list
def binary_search(nums_list, key )
      low = 0;                  // low element subscript
      high = len(nums_list) - 1;  // high element subscript
      middle = (low + high) // 2

    # loop until low subscript is greater than high subscript
      while  low <= high:
          # determine middle element subscript
          middle = ( low + high ) // 2

          # if key matches middle element, return middle location
          if  key == nums_list[ middle ]:
              return middle
          elif key < nums_list[ middle ]:
              high = middle - 1
          else:
               low = middle + 1

     return -1
```

Searching lists (Binary search)

- Let's analyze the 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?

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