CSE 503
Introduction to Computer Science for Non-Majors

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Day 31
Algorithms: Searching and Sorting
An algorithm is

"a set of rules for solving a problem in a finite number of steps"

https://www.dictionary.com/browse/algorithm
Algorithms

Two common problems we might want to solve:

**Searching** (Finding a particular element in a collection)

**Sorting** (Rearranging a collection in a specific order)
How would we search for a particular item in a list (in Python)?
def linearSearch(list, item):
    for x in list:
        if x == item:
            return True
    return False
def linearSearch(list, item):
    for x in list:
        if x == item:
            return True
    return False
def linearSearch(list, item):
    for x in list:
        if x == item:
            return True
    return False

Is 3 == 64?
def linearSearch(list, item):
    for x in list:
        if x == item:
            return True
    return False

Is 5 == 64?
def linearSearch(list, item):
    for x in list:
        if x == item:
            return True
    return False

Is 8 == 64?
def linearSearch(list, item):
    for x in list:
        if x == item:
            return True
    return False
def linearSearch(list, item):
    for x in list:
        if x == item:
            return True
    return False
def linearSearch(list, item):
    for x in list:
        if x == item:
            return True
    return False

Is 23 == 64?
Linear Search

def linearSearch(list, item):
    for x in list:
        if x == item:
            return True
    return False

Is 56 == 64?
def linearSearch(list, item):
    for x in list:
        if x == item:
            return True
    return False

Is 59 == 64?
def linearSearch(list, item):
    for x in list:
        if x == item:
            return True
    return False

Is 64 == 64?
def linearSearch(list, item):
  for x in list:
    if x == item:
      return True
  return False

Is 64 == 64?
Return True!
What if we knew our list was sorted?
(how would you find a page in a book?)
def binarySearch(list, item):
    left = 0
    right = len(list)
    while (right - left) > 0:
        mid = (left + right)//2
        if item < list[mid]:
            right = mid
        elif item > list[mid]:
            left = mid+1
        else:
            return True
    return False
def binarySearch(list, item):
    left = 0
    right = len(list)
    while (right - left) > 0:
        mid = (left + right)//2
        if item < list[mid]:
            right = mid
        elif item > list[mid]:
            left = mid+1
        else:
            return True
    return False
def binarySearch(list, item):
    left = 0
    right = len(list)
    while (right - left) > 0:
        mid = (left + right)//2
        if item < list[mid]:
            right = mid
        elif item > list[mid]:
            left = mid+1
        else:
            return True
    return False

left = 0
right = 15
mid = 7

Is 64 < 56?
def binarySearch(list, item):
    left = 0
    right = len(list)
    while (right - left) > 0:
        mid = (left + right)//2
        if item < list[mid]:
            right = mid
        elif item > list[mid]:
            left = mid+1
        else:
            return True
    return False
def binarySearch(list, item):
    left = 0
    right = len(list)
    while (right - left) > 0:
        mid = (left + right)//2
        if item < list[mid]:
            right = mid
        elif item > list[mid]:
            left = mid+1
        else:
            return True
    return False
def binarySearch(list, item):
    left = 0
    right = len(list)
    while (right - left) > 0:
        mid = (left + right) // 2
        if item < list[mid]:
            right = mid
        elif item > list[mid]:
            left = mid + 1
        else:
            return True
    return False
def binarySearch(lst, item):
    left = 0
    right = len(lst)
    while (right - left) > 0:
        mid = (left + right)//2
        if item < lst[mid]:
            right = mid
        elif item > lst[mid]:
            left = mid + 1
        else:
            return True
    return False
def binarySearch(list, item):
    left = 0
    right = len(list)
    while (right - left) > 0:
        mid = (left + right)//2
        if item < list[mid]:
            right = mid
        elif item > list[mid]:
            left = mid+1
        else:
            return True
    return False
def binarySearch(list, item):
    left = 0
    right = len(list)
    while (right - left) > 0:
        mid = (left + right) // 2
        if item < list[mid]:
            right = mid
        elif item > list[mid]:
            left = mid + 1
        else:
            return True
    return False
def binarySearch(list, item):
    left = 0
    right = len(list)
    while (right - left) > 0:
        mid = (left + right)//2
        if item < list[mid]:
            right = mid
        elif item > list[mid]:
            left = mid+1
        else:
            return True
    return False
def binarySearch(list, item):
    left = 0
    right = len(list)
    while (right - left) > 0:
        mid = (left + right)//2
        if item < list[mid]:
            right = mid
        elif item > list[mid]:
            left = mid+1
        else:
            return True
    return False

left = 8
right = 10
mid = 10

Is 64 < 72?
def binarySearch(list, item):
    left = 0
    right = len(list)
    while (right - left) > 0:
        mid = (left + right)//2
        if item < list[mid]:
            right = mid
        elif item > list[mid]:
            left = mid+1
        else:
            return True
    return False
def binarySearch(list, item):
    left = 0
    right = len(list)
    while (right - left) > 0:
        mid = (left + right)//2
        if item < list[mid]:
            right = mid
        elif item > list[mid]:
            left = mid+1
        else:
            return True
    return False
def binarySearch(list, item):
    left = 0
    right = len(list)
    while (right - left) > 0:
        mid = (left + right) // 2
        if item < list[mid]:
            right = mid
        elif item > list[mid]:
            left = mid + 1
        else:
            return True
    return False
def binarySearch(list, item):
    left = 0
    right = len(list)
    while (right - left) > 0:
        mid = (left + right)//2
        if item < list[mid]:
            right = mid
        elif item > list[mid]:
            left = mid+1
        else:
            return True
    return False
Linear Search vs Binary Search

Checking if \( x == y \) eliminates one element from consideration
Checking if \( x == y \) eliminates one element from consideration

If our input list has \( N \) elements, then we may have to do up to \( N \) comparisons in the worst case
Linear Search vs Binary Search

Checking if $x == y$ eliminates one element from consideration.

If our input list has $N$ elements, then we may have to do up to $N$ comparisons in the worst case.

Checking if $x < y$, $x > y$ eliminates half of the list from consideration.
Linear Search vs Binary Search

Checking if $x == y$ eliminates one element from consideration

If our input list has $N$ elements, then we may have to do up to $N$ comparisons in the worst case.

Checking if $x < y, x > y$ eliminates half of the list from consideration

If our input list has $N$ elements, how many comparisons would we need in the worst case?
Linear Search vs Binary Search

Checking if \( x == y \) eliminates one element from consideration

If our input list has \( N \) elements, then we may have to do up to \( N \) comparisons in the worst case

Checking if \( x < y, x > y \) eliminates half of the list from consideration

If our input list has \( N \) elements, how many comparisons would we need in the worst case?

\[ \log_2(N) \]
Linear Search vs Binary Search

What if we want to search a list of twice the size?
Linear Search vs Binary Search

What if we want to search a list of twice the size?

If $N' = 2N$, how many comparisons will we need to Linear Search a list of size $N'$?
Linear Search vs Binary Search

What if we want to search a list of twice the size?

If $N' = 2N$, how many comparisons will we need to Linear Search a list of size $N'$?

$N' = 2N$ (twice as many...)
Linear Search vs Binary Search

What if we want to search a list of twice the size?

If $N' = 2N$, how many comparisons will we need to Linear Search a list of size $N'$?

$N' = 2N$ (twice as many...)

If $N' = 2N$, how many comparisons will we need to Binary Search a list of size $N'$?
Linear Search vs Binary Search

What if we want to search a list of twice the size?

If \( N' = 2N \), how many comparisons will we need to Linear Search a list of size \( N' \)?

\[ N' = 2N \text{ (twice as many...)} \]

If \( N' = 2N \), how many comparisons will we need to Binary Search a list of size \( N' \)?

\[ \log_2(N') = \log_2(2N) = \log(N) + 1 \]

(just one more comparison...)
Binary Search only works if our list is sorted...

So how do we sort a list?
**Goal:** Given a sequence of values that can be ordered (list in Python, array in JS), rearrange the sequence so that the values go from smallest to larger (or largest to smallest).

**Example:**

\[ [12, 56, 4, 8, 19, 16, 37, 23] \rightarrow [4, 8, 12, 16, 19, 23, 37, 56] \]
Both Python and JavaScript have built-in sorting functions.

If `a` is a sequence:

```python
a = [12, 56, 4, 8, 19, 16, 37, 23]
```

then `a.sort()` (in both Python and JavaScript) will sort `a` to:

```python
[4, 8, 12, 16, 19, 23, 37, 56]
```
How might we go about implementing sort?

(when you need to sort, just call sort…but it is useful to know how it might work)
Selection Sort involves selecting the smallest element from the list, and appending it to your sorted list:

Input List: [12, 56, 4, 8, 19, 16, 37, 23]

Output List: []
Selection Sort involves selecting the smallest element from the list, and appending it to your sorted list:

**Input List:** [12, 56, 4, 8, 19, 16, 37, 23]

Find the smallest element (4)

**Output List:** [4]
Selection Sort involves selecting the smallest element from the list, and appending it to your sorted list:

Input List: [12, 56, 8, 19, 16, 37, 23]
Remove it from the input...
Output List: [ ]
Selection Sort involves selecting the smallest element from the list, and appending it to your sorted list:

Input List: [12, 56, 8, 19, 16, 37, 23]

Append it to the output

Output List: [4]
Selection Sort involves selecting the smallest element from the list, and appending it to your sorted list:

Input List: [12, 56, 8, 19, 16, 37, 23]

Find the smallest element (8)

Output List: [4]
Selection Sort

Selection Sort involves selecting the smallest element from the list, and appending it to your sorted list:

Input List: [12, 56, 19, 16, 37, 23]
Remove it from the input
Output List: [4]
Selection Sort

Selection Sort involves selecting the smallest element from the list, and appending it to your sorted list:

Input List: [12, 56, 19, 16, 37, 23]

Append it to the output

Output List: [4, 8]
Selection Sort

**Selection Sort** involves *selecting* the smallest element from the list, and appending it to your sorted list:

**Input List:** [12, 56, 19, 16, 37, 23]

Repeat until sorted...

**Output List:** [4, 8]
Selection Sort involves selecting the smallest element from the list, and appending it to your sorted list:

Input List: [12, 56, 19, 16, 37, 23]

Repeat until sorted...

Output List: [4, 8]
Selection Sort

Selection Sort involves selecting the smallest element from the list, and appending it to your sorted list:

Input List: [56, 19, 16, 37, 23]
Repeat until sorted...
Output List: [4, 8, 12]
Selection Sort involves selecting the smallest element from the list, and appending it to your sorted list:

Input List: [56, 19, 16, 37, 23]

Repeat until sorted...

Output List: [4, 8, 12]
Selection Sort involves selecting the smallest element from the list, and appending it to your sorted list:

**Input List:** [56, 19, 37, 23]

Repeat until sorted...

**Output List:** [4, 8, 12, 16]
Selection Sort involves selecting the smallest element from the list, and appending it to your sorted list:

Input List: [56, 19, 37, 23]

Repeat until sorted...

Output List: [4, 8, 12, 16]
Selection Sort

Selection Sort involves selecting the smallest element from the list, and appending it to your sorted list:

Input List: [56, 37, 23]
Repeat until sorted...
Output List: [4, 8, 12, 16, 19]
Selection Sort involves selecting the smallest element from the list, and appending it to your sorted list:

Input List: [56, 37, 23]

Repeat until sorted...

Output List: [4, 8, 12, 16, 19]
Selection Sort

Selection Sort involves selecting the smallest element from the list, and appending it to your sorted list:

Input List: [56, 37]

Repeat until sorted...

Output List: [4, 8, 12, 16, 19, 23]
Selection Sort involves selecting the smallest element from the list, and appending it to your sorted list:

Input List: [56, 37]

Repeat until sorted...

Output List: [4, 8, 12, 16, 19, 23]
Selection Sort involves selecting the smallest element from the list, and appending it to your sorted list:

Input List: [56]

Repeat until sorted...

Output List: [4, 8, 12, 16, 19, 23, 37]
Selection Sort involves selecting the smallest element from the list, and appending it to your sorted list:

Input List: [56]

Repeat until sorted...

Output List: [4, 8, 12, 16, 19, 23, 37]
Selection Sort involves selecting the smallest element from the list, and appending it to your sorted list:

Input List: [ ]
Repeat until sorted...
Output List: [4, 8, 12, 16, 19, 23, 37, 56]
**Selection Sort**

**Selection Sort** involves *selecting* the smallest element from the list, and appending it to your sorted list:

**Input List:** [ ]

**Output List:** [4, 8, 12, 16, 19, 23, 37, 56]
def selectionSort(unsorted):
    sorted = []
    while len(unsorted) > 0:
        x = removeSmallest(unsorted)
        sorted.append(x)
    return sorted

def removeSmallest(aList):
    smallest = aList[0]
    for value in aList:
        if value < smallest:
            smallest = value
    aList.remove(smallest)
    return smallest
def selectionSort(unsorted):
    sorted = []
    while len(unsorted) > 0:
        x = removeSmallest(unsorted)
        sorted.append(x)
    return sorted

As long as our unsorted list still has elements, remove the smallest and append it to our sorted list

def removeSmallest(aList):
    smallest = aList[0]
    for value in aList:
        if value < smallest:
            smallest = value
    aList.remove(smallest)
    return smallest
Selection Sort

def selectionSort(unsorted):
    sorted = []
    while len(unsorted) > 0:
        x = removeSmallest(unsorted)
        sorted.append(x)
    return sorted

As long as our unsorted list still has elements, remove the smallest and append it to our sorted list

def removeSmallest(aList):
    smallest = aList[0]
    for value in aList:
        if value < smallest:
            smallest = value
    aList.remove(smallest)
    return smallest

Look through each value (linearly) in the list to find the smallest, then remove it
How many steps does our selection sort take with a list of size $N$?
Selection Sort Analysis

How many steps does our selection sort take with a list of size $N$?

Finding the smallest item uses a linear search
How many steps does our selection sort take with a list of size $N$?

Finding the smallest item uses a linear search: $N$ steps
Selection Sort Analysis

How many steps does our selection sort take with a list of size $N$?

Finding the smallest item uses a linear search: $N$ steps

How many times do we have to find the smallest item?
Selection Sort Analysis

How many steps does our selection sort take with a list of size $N$?

Finding the smallest item uses a linear search: $N$ steps

How many times do we have to find the smallest item?

$N$ times (once for each item in the list)
Selection Sort Analysis

How many steps does our selection sort take with a list of size $N$?

Finding the smallest item uses a linear search: $N$ steps

How many times do we have to find the smallest item?

$N$ times (once for each item in the list)

Total number of steps: $N \times N = N^2$
Selection Sort Analysis

How many steps does our selection sort take with a list of size $N$?

Finding the smallest item uses a linear search: $N$ steps

How many times do we have to find the smallest item?

$N$ times (once for each item in the list)

Total number of steps: $N \times N = N^2$

This isn't... 100% accurate, but intuitively it gets the point across.

In reality, finding the smallest takes $N$ steps, then $N-1$ steps, then $N-2$ steps...

But $N + (N-1) + (N-2) + ... + 2 + 1 = N^2$
Sorting

\(N^2\) grows pretty fast...

If our list doubles in size, the sort will take 4 times as long!

*Can we do better?*
Sorting

$N^2$ grows pretty fast...

If our list doubles in size, the sort will take 4 times as long!

Can we do better? YES!