CSE 503
Introduction to Computer Science for Non-Majors

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

Day 31
Algorithms: Searching and Sorting

Algorithms

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)

Searching

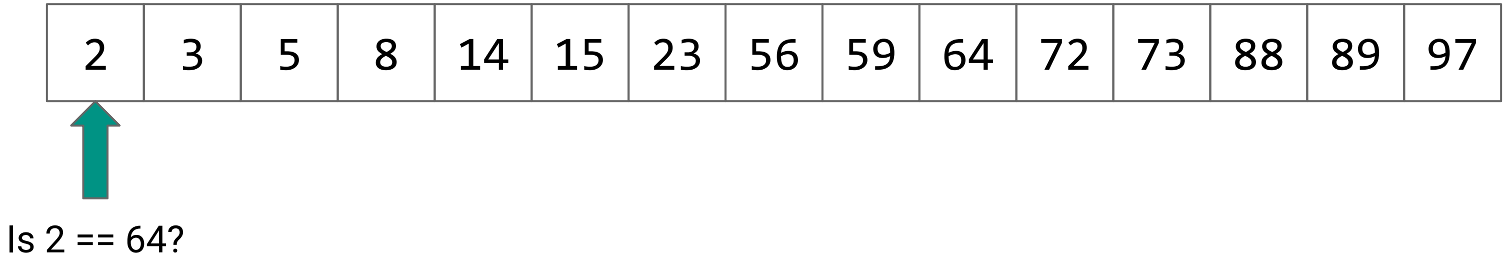
How would we search for a particular item in a list (in Python)?

Linear Search

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

Linear Search

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



Linear Search

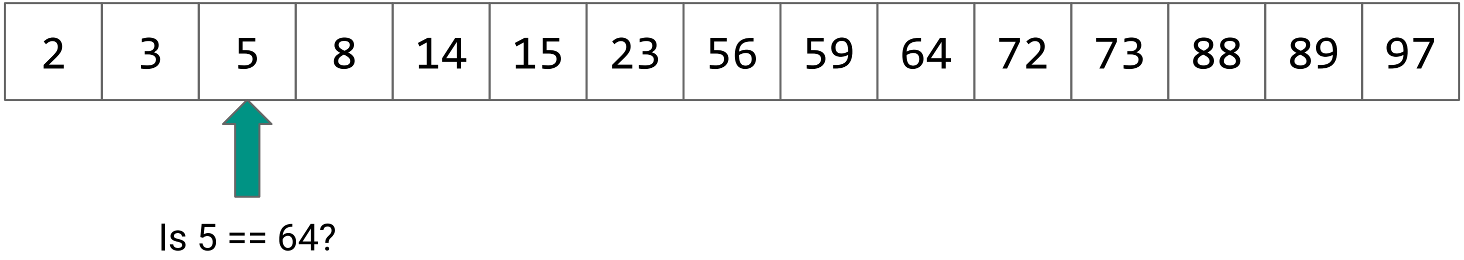
```python
def linearSearch(list, item):
  for x in list:
    if x == item:
      return True
  return False
```

2	3	5	8	14	15	23	56	59	64	72	73	88	89	97

Is 3 == 64?

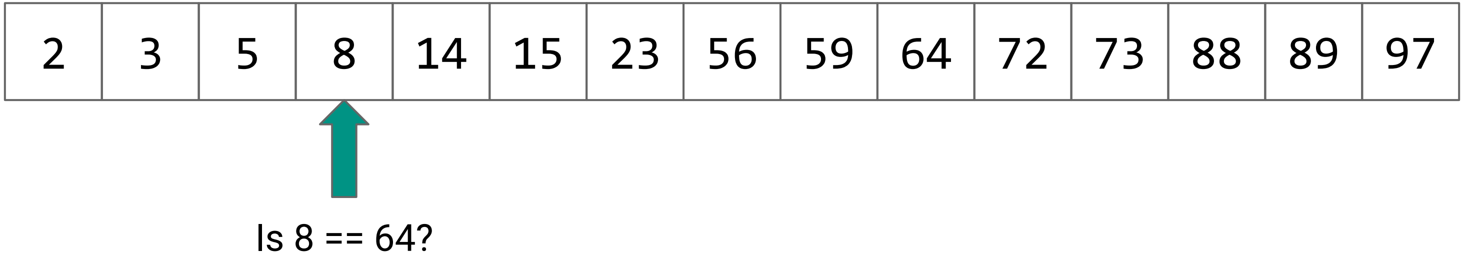
Linear Search

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



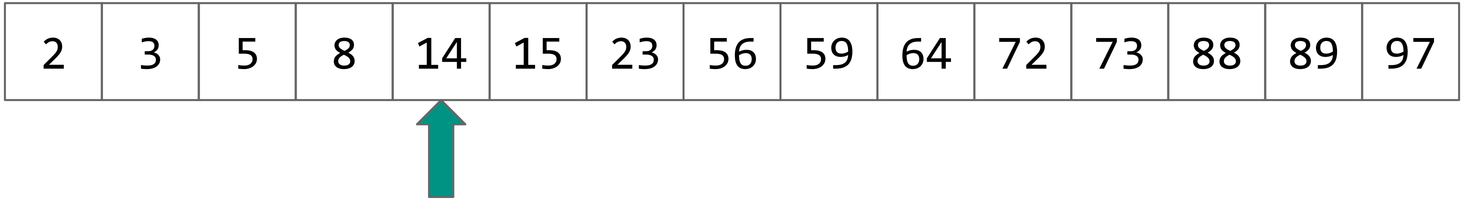
Linear Search

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



Linear Search

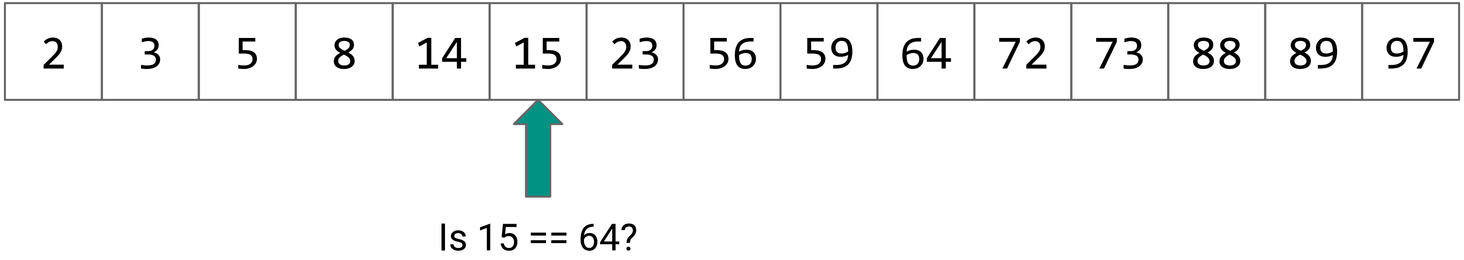
```
def linearSearch(list, item):
  for x in list:
    if x == item:
      return True
  return False
```



Is 14 == 64?

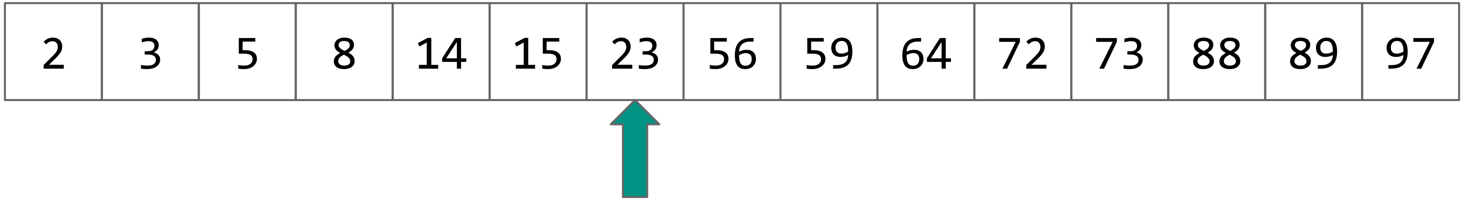
Linear Search

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



Linear Search

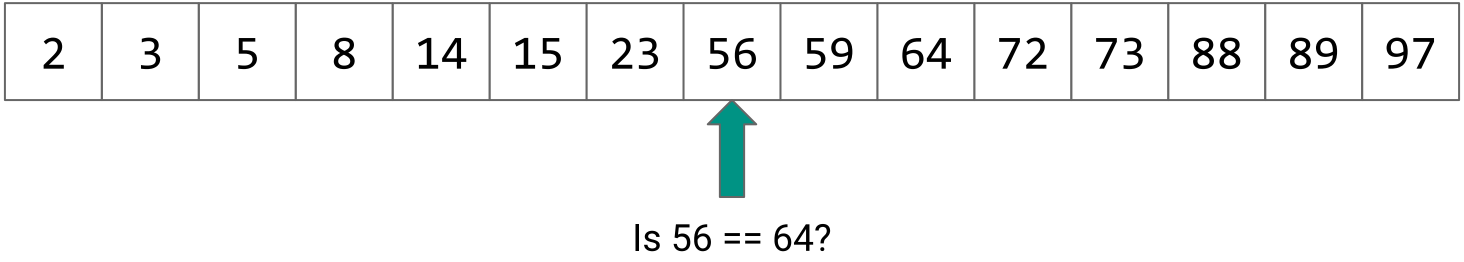
```
def linearSearch(list, item):
  for x in list:
    if x == item:
      return True
  return False
```



Is 23 == 64?

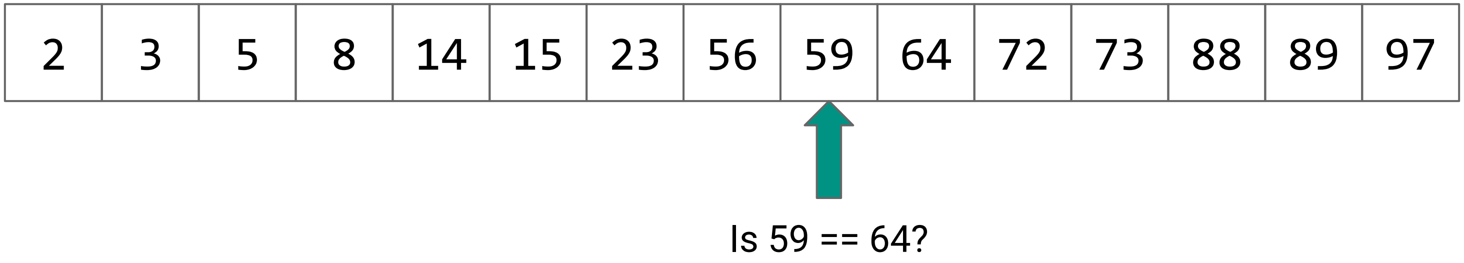
Linear Search

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



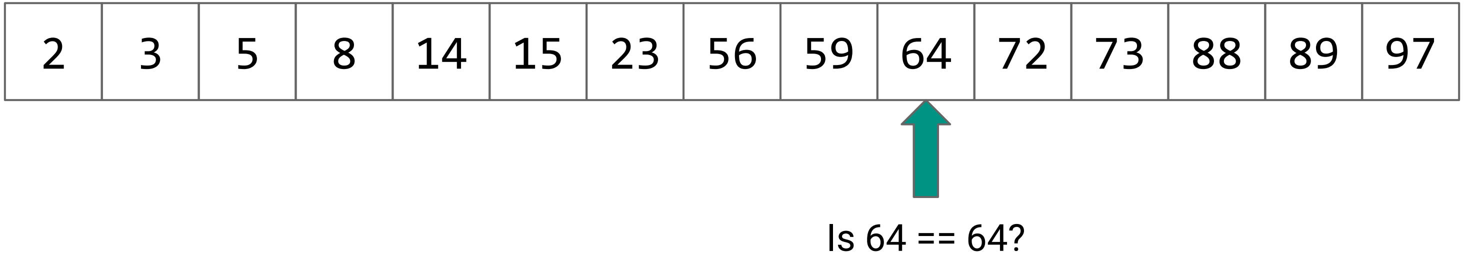
Linear Search

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



Linear Search

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



Linear Search

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

2	3	5	8	14	15	23	56	59	64	72	73	88	89	97

Is 64 == 64?

Return True!

Searching

What if we knew our list was sorted?

(how would you find a page in a book?)

BinarySearch

```
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
```

BinarySearch

```python
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

2	3	5	8	14	15	23	56	59	64	72	73	88	89	97

BinarySearch

```
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
```

2	3	5	8	14	15	23	56	59	64	72	73	88	89	97

Is 64 < 56?

BinarySearch

```
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
```

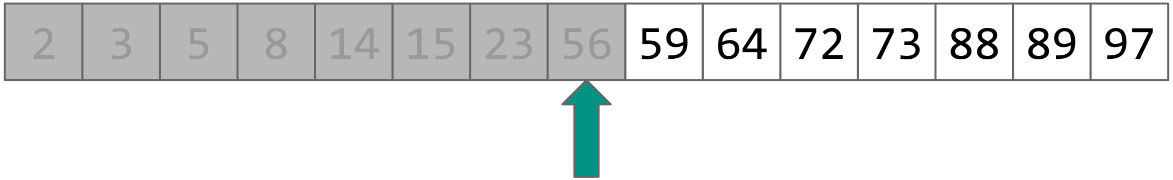
2	3	5	8	14	15	23	56	59	64	72	73	88	89	97

Is 64 > 56?

BinarySearch

```
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 = 15
mid = 7

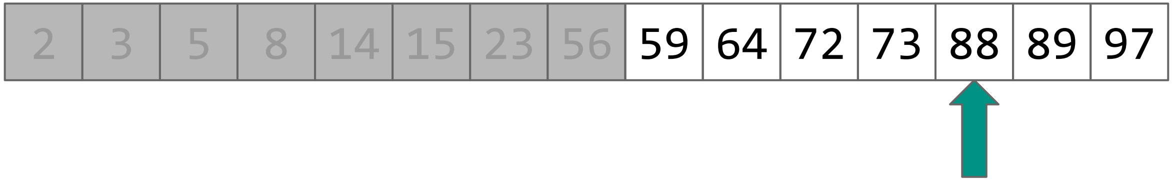


Is 64 > 56?

BinarySearch

```
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
```

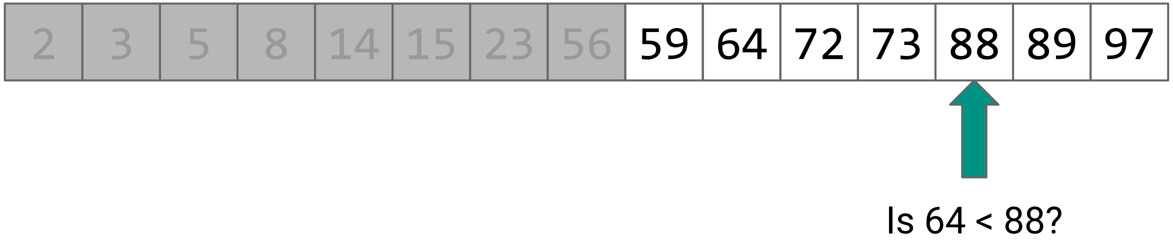
left = 8
right = 15
mid = 12



BinarySearch

```
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
```

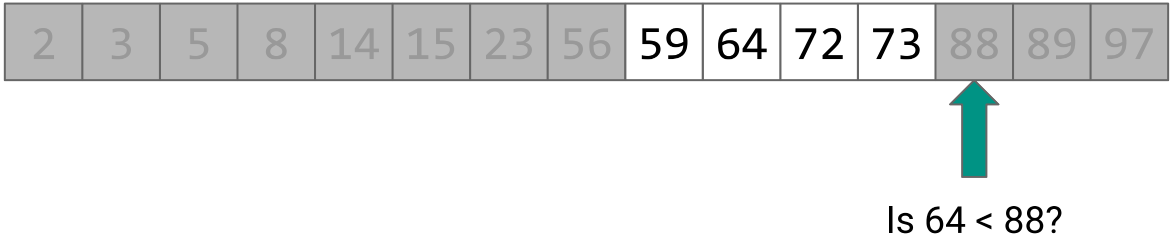
left = 8
right = 15
mid = 12



BinarySearch

```
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
```

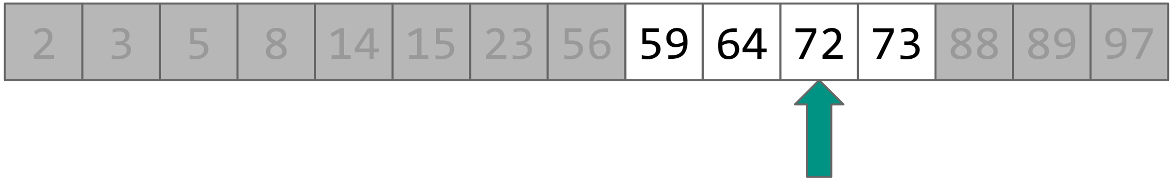
left = 8
right = 12
mid = 12



BinarySearch

```python
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
```

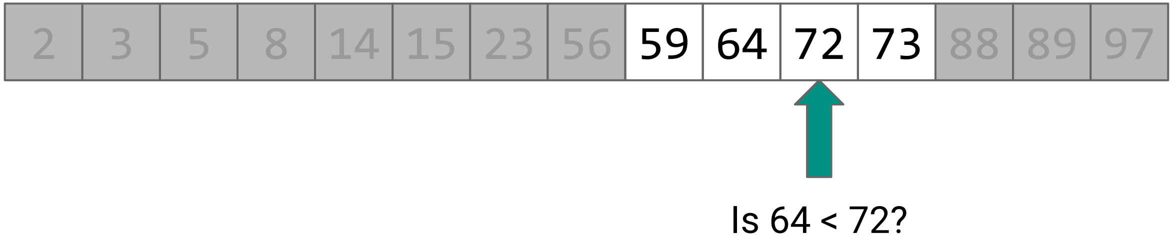
```
left = 8
right = 12
mid = 10
```



BinarySearch

```
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
```

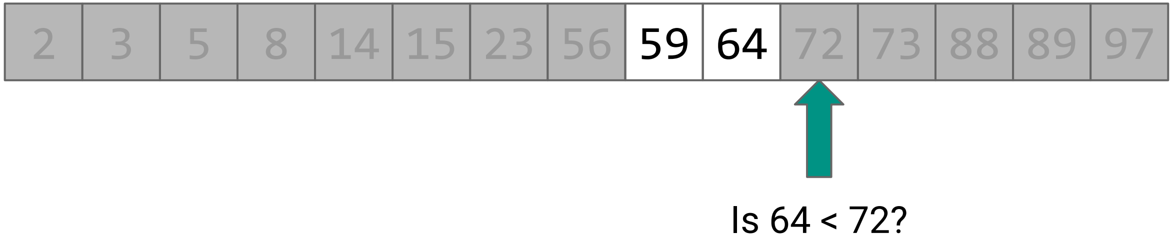
left = 8
right = 12
mid = 10



BinarySearch

```
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
```

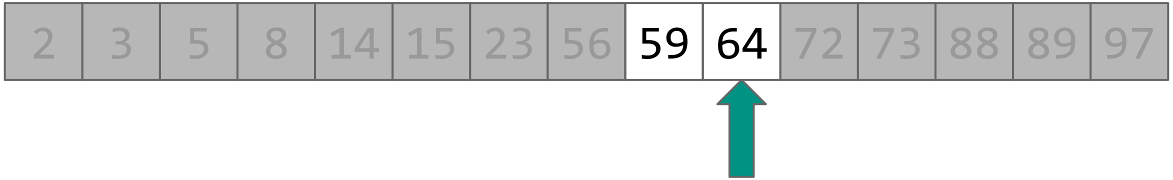
```
left = 8
right = 10
mid = 10
```



BinarySearch

```
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
```

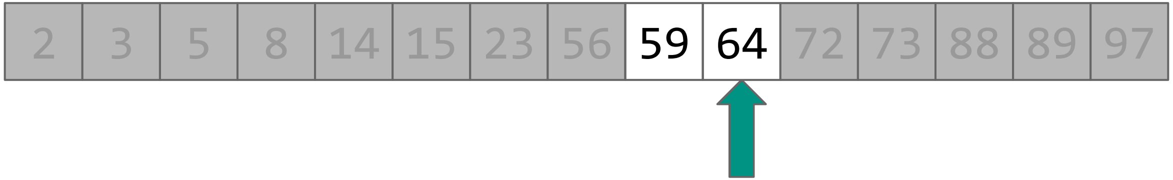
left = 8
right = 10
mid = 9



BinarySearch

```
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 = 9

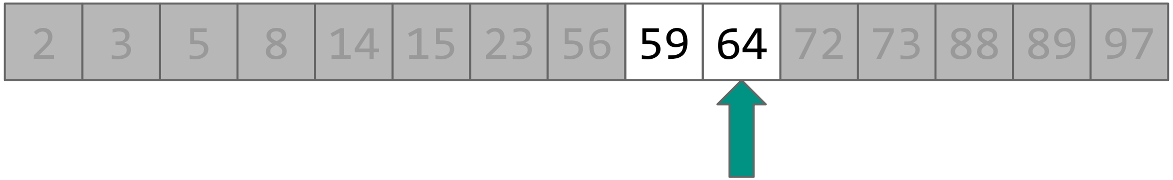


Is 64 < 64?

BinarySearch

```
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 = 9
```

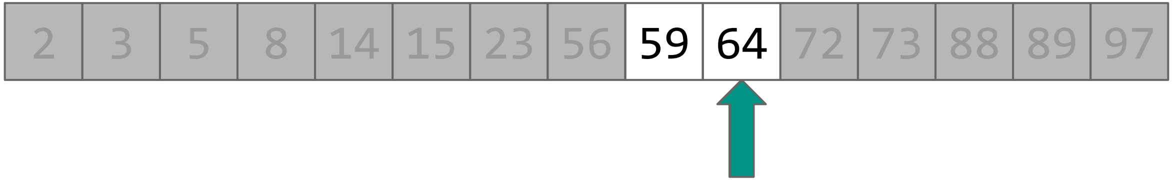


Is 64 > 64?

BinarySearch

```
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 = 9



Is 64 > 64?
Return True!

Linear Search vs Binary Search

Checking if x == y eliminates one element 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

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

N' = 2*N* (twice as many…)

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

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

If N' = **2*N***, 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...)

Sorting

Binary Search only works if our list is sorted…

So how do we sort a list?

Sorting

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]` → `[4, 8, 12, 16, 19, 23, 37, 56]`

Sorting in Python and JavaScript

Both Python and JavaScript have built-in sorting functions.

If **`a`** is a sequence:

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

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

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

Sorting

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

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

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: [ ]

Selection Sort

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

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

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

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

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

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, 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

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

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

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

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

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]`

Selection Sort

```
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
```

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 appent 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 appent 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

Selection Sort Analysis

*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

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

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 $N + (N-1) + (N-2) + \ldots + 2 + 1 = N^2$

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!**