GLIS 691 | Karyn Moffatt

Computer Programming for Information Professionals

Searching

Find tel. # 514-123-4567. What can you do?

Sequential Search

- Search for a target item `target` in a list `numList` of `n` items.
- Returns the index of the first match, or -1 if no match found

In [ ]:
```
def seqSearch(numList, target):
    index = 0                                   # create a variable to track our position in the list
    while index < len(numList):                 # loop over each item in the list using index
        currentItem = numList[index]            # get the current item from the list
        if numList[index] == target:            # check if the current item matches our target
            return index                        # we found the target --> return the index value
        index += 1                              # increment the index to look at the next item
    print("Stopping after", index+1, "iterations")
    return -1                                   # if we get to the end of the loop, no match
```

In [ ]:
```
# Testing sequential search
nums = [4, 7, 1, 9, 5, 8]
result = seqSearch(nums, 3)
print(result)
```

[Visualize this code
(http://www.pythontutor.com/visualize.html#code=def%20seqSearch%28numList,%20target%29%3A%0A%20%20%20%20index%20%3D%200%
-
%3E%20return%20the%20index%20value%0A%20%20%20%20%20%20%20%20index%20%2B%3D%201%20%20%20%20%20%20%20%2(
1%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%
frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false)]

How many comparisons do we need in the worst case scenario?

Sequential Search — Sorted Listed

Imagine we know the list is sorted (smallest to largest). How might we take advantage of that?

In [ ]:
```
def seqSearch(sortedNumList, target):
    index = 0
    while index<len(sortedNumList):
        currentItem = sortedNumList[index]
        if currentItem == target:                # check if the current item matches our target
            return index                         # If it, we've found our match!
        if currentItem > target:                 # check if the current item is larger than our target
            print("Stopping after", index+1, "iterations")
            return -1                            # If it is, then we can stop looking, we have no match
        index += 1
    print("Stopping after", index+1, "iterations")
    return -1                                    # If we get to the end, then we have no match
                                                 # (and our target is bigger than the biggest item in the list)
```

In [ ]:
```
nums = [1, 4, 5, 7, 8, 9]
result = seqSearch(nums, 3)
print(result)
```

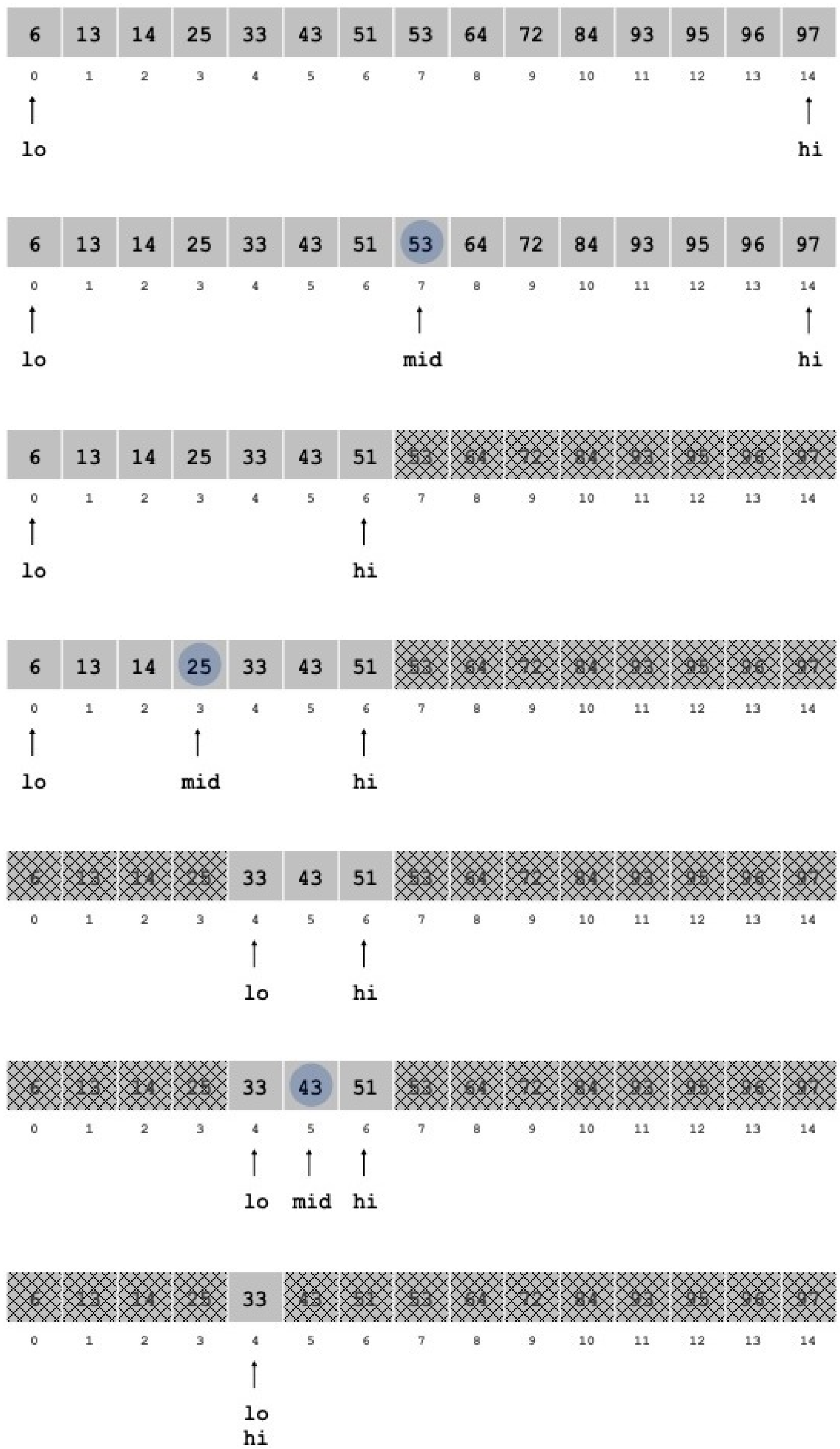
Sequential Search on a Sorted List

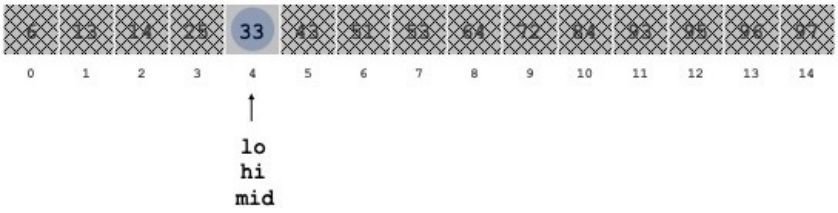
- Better...
  - We can avoid *some* unnecessary searching
- But...
  - How do we sort? *(We'll come back to that later)*
  - Still might have to search all items in the list
  - What if there are millions of items?

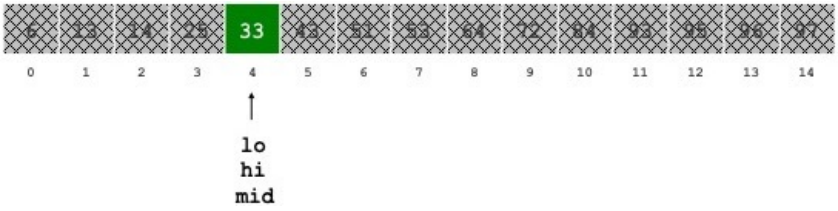
Can we do better?

Divide and Conquer — Binary Search

- Watch the video "Santa's Dirty Socks"
- Binary Search:
  - Given a value `v` and a sorted list `l[]`,
  - find index `i` such that `l[i]==v`, or report that no such value exists.
- Invariant. Algorithm maintains `l[lo]<= value <= l[hi]`
  - Ex. Binary search for 33







Binary Search in Python

- You are not expected to be able to write this...

In [ ]:
```
def binSearch(numList, low, high, target):
    print("run!")
    if (low > high):                                    # if low > high, then we have searched
through everything
        return -1

    mid = (low+high)//2                                 # find the middle index. Double slash
 --> integer division
    currentItem = numList[mid]                          # get the item from the middle index
    if currentItem == target:                           # if current item == target then we ha
ve found it!
        return mid
    if target < currentItem:                            # if target is less than current ite
m...
        return binSearch(numList, low, mid-1, target)   # call binSearch again on lower half o
f the list
    else:                                               # otherwise
        return binSearch(numList, mid+1, high, target)  # call binSearch again on upper half o
f the list
```

In [ ]:
```
# Testing binSearch()
nums = [6, 13, 14, 25, 33, 43, 51, 53, 64, 72, 84, 93, 95, 96, 97]
result = binSearch(nums, 0, len(nums) - 1, 7)
print(result)
```

Visualize this code (http://www.pythontutor.com/visualize.html#code=def%20binSearch%28numList,%20low,%20high,%20target%29%3A%0A%20%20%20%20if%201%0A%20%20%20%20%0A%20%20%20%20mid%20%3D%20%28low%2Bhigh%29%20//%202%0A%20%20%20%20currentItem%20%3D%21,%20target%29%0A%20%20%20%20else%3A%20%0A%20%20%20%20%20%20%20%20%20%20%20%20return%20binSearch%28numList,%20mid%2B1,%2%201,%2033%29%0Aprint%28result%29&cumulative=false&curInstr=0&heapPrimitives=nevernest&mode=display&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false)