COSC 2306
Data Programming

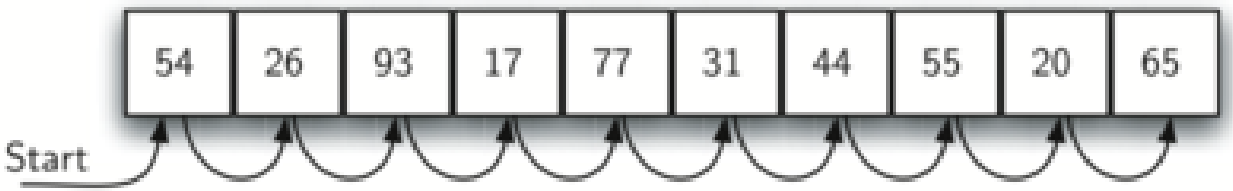
Search

Search

- Find something in data structures
  - Lists
  - Sorted Lists
  - Hash tables

Sequential search

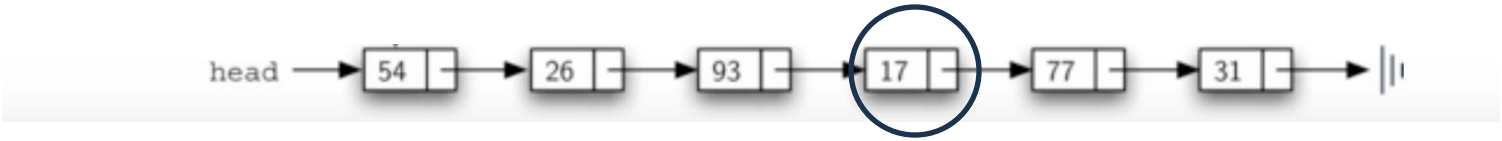
- A sequential search of a list/array begins at the beginning of the list/array and continues until the item is found or the entire list/array has been searched
- The search result can be True/False or return the position of found item



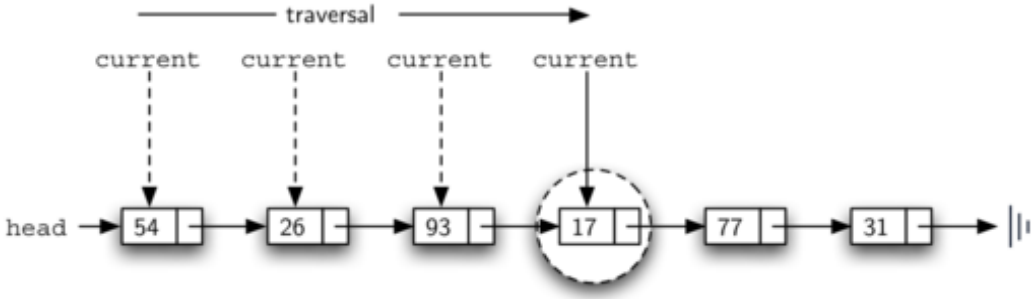
Sequential search

- We have seen it for array lists and linked lists

```
>>> 15 in [3,5,2,4,1]
False
>>> 3 in [3,5,2,4,1]
True
```

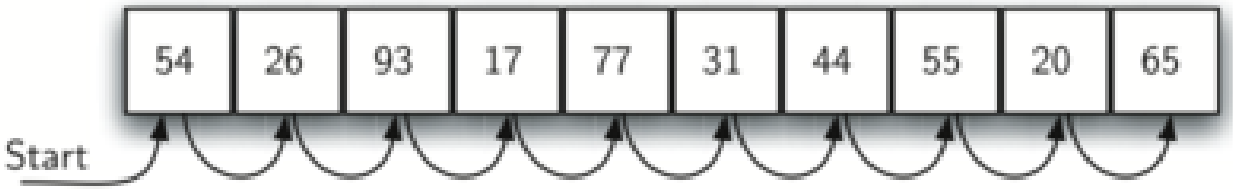


Step1: search the item



Sequential search coding practice

- Define a function SequentialSearch that takes input of a list and a search target
- Return True if the item is found, otherwise return False

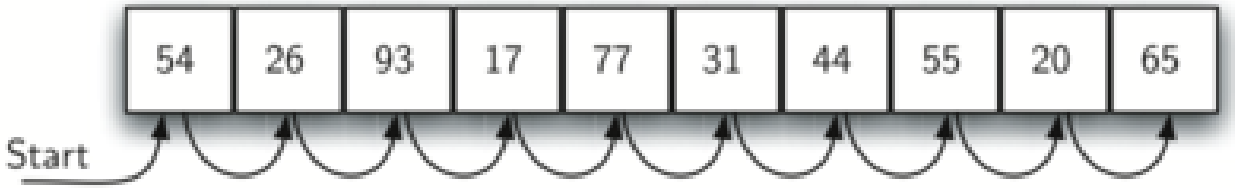


Sequential search

```
def SequentialSearch(alist, item):
    pos = 0
    found = False

    while pos < len(alist) and not found:
        if alist[pos] == item:
            found = True
        else:
            pos = pos + 1

    return found
```

```
numbers = [59, 37, 10, 25, 42]
target = 42

if SequentialSearch(numbers, target):
    print(f"{target} is found in the list.")
else:
    print(f"{target} is not found in the list.")
```

Sequential search analysis



What is the best case and what is the worst case?

Case 1: found. Best: 1
Worst: n

Case 2: not found. Best: n
Worst: n

How about the average case?

Sequential search analysis

Suppose that there are *n* elements in the array. The following expression gives the average number of comparisons:

$$\frac{1+2+\ldots+n}{n}$$

It is known that

$$1+2+\ldots+n=\frac{n(n+1)}{2}$$

Therefore, the following expression gives the average number of comparisons made by the sequential search in the successful case:

$$\frac{1+2+\ldots+n}{n}=\frac{1}{n}\frac{n(n+1)}{2}=\frac{n+1}{2}$$

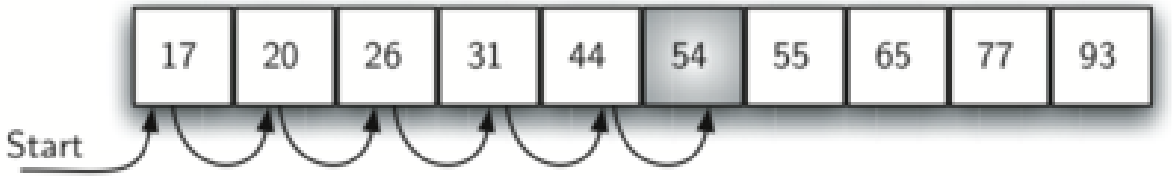
Sequential search analysis

Case	Best Case	Worst Case	Average Case
item is present	1	n	$\frac{n+1}{2}$
item is not present	n	n	n

Complexity: O(n)

Ordered sequential search analysis

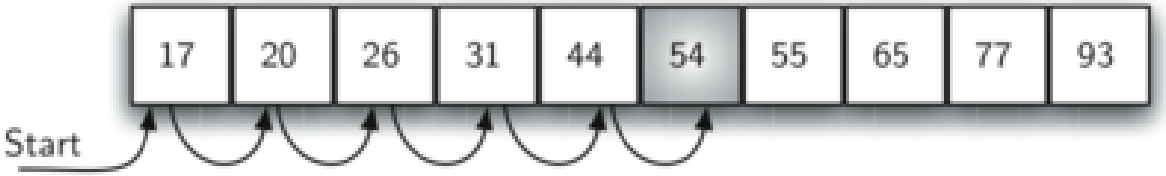
Ordered List



What is the best, the worst, and average case?

Ordered sequential search analysis

Ordered List

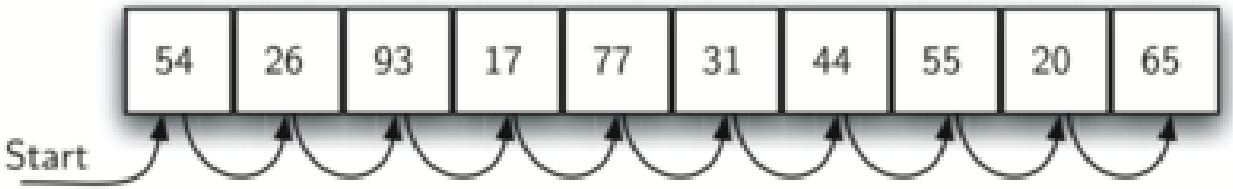


Case	Best Case	Worst Case	Average Case
item is present	1	n	$\frac{n+1}{2}$
item is not present	1	n	$\frac{n}{2}$

Complexity: O(n)

Ordered sequential search coding practice

- Define a function orderedSequentialSearch that takes input of a list and a search target
- Return True if the item is found, otherwise return False



Ordered sequential search

```
def orderedSequentialSearch(alist, item):
    pos = 0
    found = False
    stop = False

    while pos < len(alist) and not found and not stop:
        if alist[pos] == item:
            found = True
        else:
            if alist[pos] > item:
                stop = True
            else:
                pos = pos + 1

    return found
```

```
numbers = [10, 25, 37, 42, 59]
target = 42

if orderedSequentialSearch(numbers, target):
    print(f"{target} is found in the list.")
else:
    print(f"{target} is not found in the list.")
```

Search

Can we do better search in ordered data?

Binary search

- "Divide-and-conquer" strategy
- Assumes that the items in the array are **sorted**
- The algorithm **begins at the middle** of the array
- If the item we are looking for **is less than the item in the middle**, we know that the item won't be in the second half of the array
- **Repeat** by examining the middle element of "interesting half"
- The process continues with each comparison cutting in half the portion of the array where the item might be

Binary search

[0]	ant
[1]	cat
[2]	chicken
[3]	cow
[4]	deer
[5]	dog
[6]	fish
[7]	goat
[8]	horse
[9]	lion
[10]	snake

Searching for cat

BinarySearch(0, 10)	middle: 5	cat < dog	
BinarySearch(0, 4)	middle: 2	cat < chicken	
BinarySearch(0, 1)	middle: 0	cat > ant	
BinarySearch(1, 1)	middle: 1	cat = cat	**Return: true**

Searching for zebra

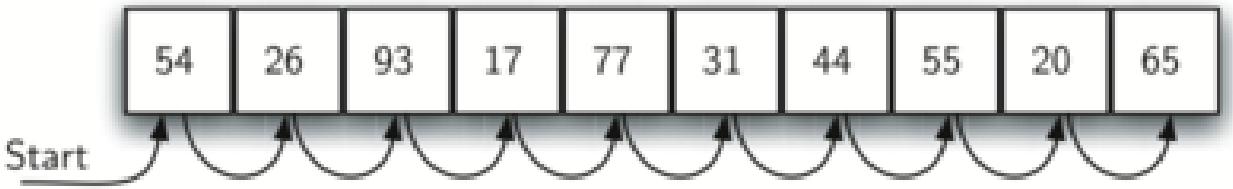
BinarySearch(0, 10)	middle: 5	zebra > dog	
BinarySearch(6, 10)	middle: 8	zebra > horse	
BinarySearch(9, 10)	middle: 9	zebra > lion	
BinarySearch(10, 10)	middle: 10	zebra > snake	
BinarySearch(11, 10)		last > first	**Return: false**

Searching for fish

BinarySearch(0, 10)	middle: 5	fish > dog	
BinarySearch(6, 10)	middle: 8	fish < horse	
BinarySearch(6, 7)	middle: 6	fish = fish	**Return: true**

Binary search coding practice

- Define a function BinearySearch that takes input of a list and a search target
- Return True if the item is found, otherwise return False



Binary search

```
def BinarySearch(alist, item):
    first = 0
    last = len(alist) - 1
    found = False

    while first <= last and not found:
        midpoint = (first + last) // 2
        if alist[midpoint] == item:
            found = True
        elif item < alist[midpoint]:
            last = midpoint - 1
        else:
            first = midpoint + 1

    return found
```

```
numbers = [1, 3, 5, 7, 9, 11, 13]
target = 7
result = BinarySearch(numbers, target)
print(f"Item {target} found? {result}")
```

Binary search coding practice

- Define a **recursive** function BinearySearch that takes input of a list and a search target
- Return True if the item is found, otherwise return False