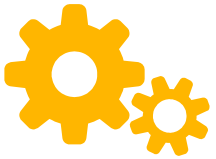
Internet and Big Data

Review

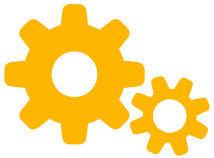


What is this code doing? Suppose it is passed a list of integers.

```
def mystery(lst):
  if len(lst) <= 1:
    return lst
  else:
    return lst[-1:] + mystery(lst[:-1])
```

A) Reverse list
B) Move last number to front
C) Remove last number from list
D) Nothing: infinite recursion.

This unit



The internet has given us data. A LOT of data. For example, Google is able to search through billions of web pages, and Amazon has millions of products.

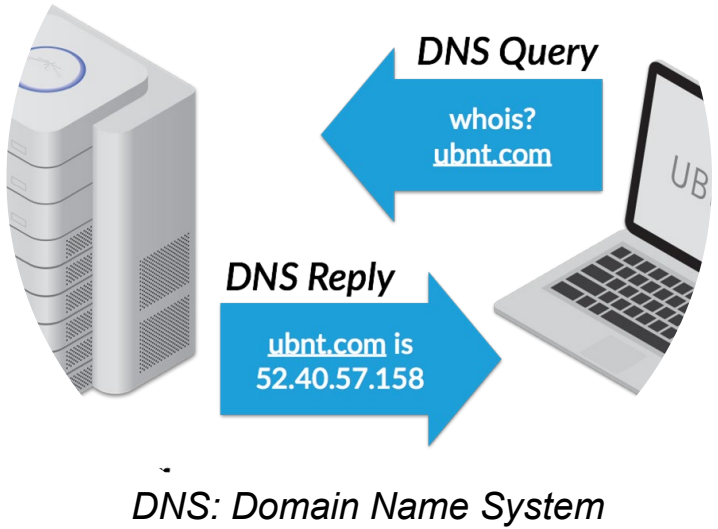
We will learn about searching, sorting and how to do it fast even when there's lots data to crunch.

- The **linear** and **binary search** algorithms and code
- The **selection** and **merge sort** algorithms and code
- The implications and "complexity" of these algorithms

Where is search used?

We need search to be especially efficient for large amounts of data.

- Google
- Domain name system (DNS) servers
- Music databases
- Instagram and Giphy hashtags
- Amazon customer databases
- … everywhere!



Example of addresses for the sfu.ca domain

Unit 5 Internet and Big Data

1/APPLICATIONS

In this unit, we'll learn about algorithms that allows machines to **search** and **sort** data in an efficient manner.

2/ALGORITHMS

We'll learn about algorithms for sorting, searching, and complexity.

3/PROGRAMMING LANGUAGE

In Python 3, we'll be learning the syntax and keywords to implement our algorithms.

4/DOCUMENTATION AND TESTING

We will show how to tell if our program is any good or not!

Any shoes in size 9?

Searching

First, the basic version. Linear Search.

http://interactivepython.org/courselib/static/pythonds/SortSearch/TheSequentialSearch.html

Linear Search - Boolean

Given a list of numbers (e.g. of shoe sizes), and a search term, can you write a function that will return True if the search term is in the list, and False otherwise?

Constraint: You can't use the "if x in list" construct.

```
# Input: List of numbers, number to search for
# Output: Boolean, whether the number is in the list or not
def linear_search_bool(input_list, search_term):
  for item in input_list:
    if item == search_term:
      return True
  return False
```

How can we test this?

Testing vs. *Testing*

```
odd_num_items = [1,2,3,4,5]
even_num_items = [1,2,3,4]
empty_list = []

print(linear_search_bool(odd_num_items,3))
print(linear_search_bool(even_num_items,3))
print(linear_search_bool(odd_num_items,5))
print(linear_search_bool(even_num_items,5))
print(linear_search_bool(odd_num_items,1))
print(linear_search_bool(even_num_items,1))
print(linear_search_bool(odd_num_items,9))
print(linear_search_bool(empty_list,9))
```

Checking all the different edge cases.

Is it easy to tell whether our search works or not?

```
True
True
True
False
True
True
False
False
```

Testing with assert

- Use an assert statement to automatically check that the code behaves as expected.

```
12  odd_num_items = [1, 3, 5]
13  assert linear_search_bool(odd_num_items, 1)
14  assert linear_search_bool(odd_num_items, 5)
15  assert linear_search_bool(odd_num_items, 9) == False
16  assert linear_search_bool(odd_num_items, 0) == False
17  assert linear_search_bool(odd_num_items, 2) == False
18
19  even_num_items = [2, 4, 6]
20  assert linear_search_bool(even_num_items, 2)
21  assert linear_search_bool(even_num_items, 6)
22  assert linear_search_bool(even_num_items, 9) == False
23  assert linear_search_bool(even_num_items, 0) == False
24  assert linear_search_bool(even_num_items, 3) == False
```

Linear Search - Boolean

Version 2 with while

```
# Input: List of numbers, number to search for
# Output: Boolean, whether the number is in the list or not
def linear_search_bool_while(input_list, search_term):
    i = 0
    while i < len(input_list):
        if input_list[i] == search_term:
            return True
        i+=1
    return False
```

Since we already wrote the tests, we can see if this works quite easily!

Linear Search - Location

Given a list of numbers (e.g. of shoe sizes), and a search term, can you write a function that will return the **index** of the search term, and None otherwise?

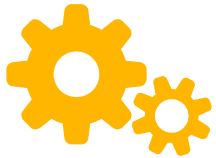
```
# Input: List of numbers, number to search for
# Output: The first index where number is found, None if not in list
def linear_search_index(input_list, search_term):
    for i in range(len(input_list)):
        if input_list[i] == search_term:
            return i
    return None
```

Can you think of how to rewrite this with a while loop?

And now...

We learned how to write a linear search algorithm. Is this the most efficient way to search? Consider your CD collection. How could you organize your information to make searching faster?

Faster searching



It's hard to find items when they're disorganized or out of order. Searching *could* be **quick**, if it's the **first** item you look at, or **slow** if it's the **last** thing you look at.

What if we assumed the items were sorted?

Although it will **take some time to do the initial sorting** (we will see next class), a sorted list will make **looking for an item** much quicker later, every single time.

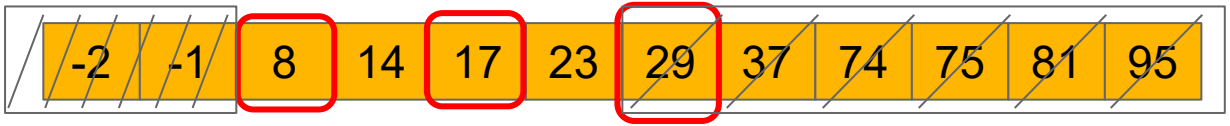
Binary Search

Once we've **sorted a list**, we can do **faster searching** compared to linear search

Recall our guess a number game

Example binary search

Suppose we want to search for 17 and return True if it exists.



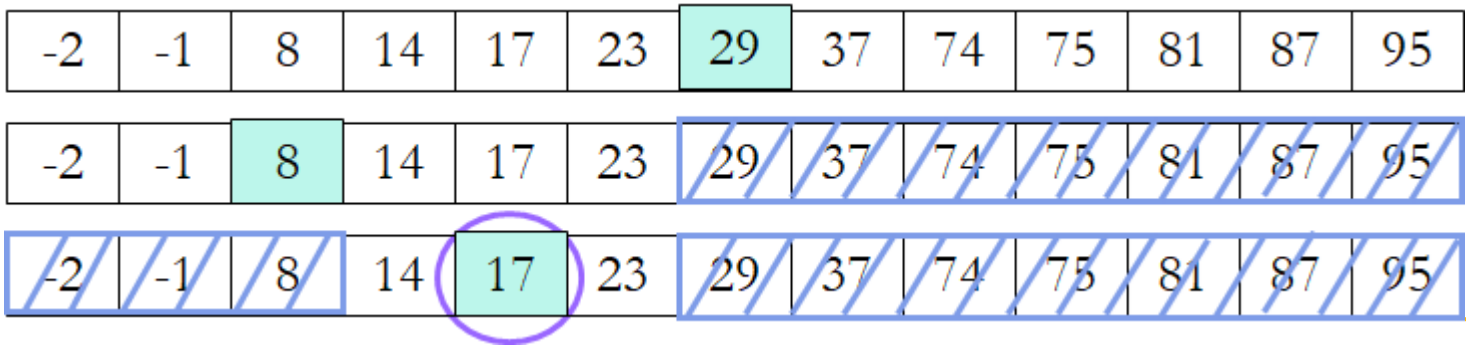
Half the array can quickly be eliminated:

- Check the mid element: 29
- Can ignore half of array depending on which half we should keep actively searching.

Example (cont)

Idea: keep track of the “active” part of the array.

- Look at the middle of the active part.
- Found it: done!
- Not found: throw away one half and repeat



Visualising Binary Search

Binary search works by **reducing** the search space in **half** each time. (like the accumulation dividing by two in today's introduction).

It **requires** the data to be **sorted**.

Use this visualization to understand it better:
https://yongdanielliang.github.io/animation/web/BinarySearchNew.html

Coding 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
        else:
            if item < alist[midpoint]:
                last = midpoint-1
            else:
                first = midpoint+1

    return found
```

Find the middle item

Update the first/last to half the search space

Can you modify this function so it returns the index of the item (and -1 if not found)?

Let's review some concepts

What does DNS stand for?

What is the name of the of **searching algorithms** we learned in this class?

If a list is sorted, what kind of search can we do?

What indices do we need to keep track of for binary search?