# Data Structures in C

Prof. Georg Feil

## Search Algorithms

Winter 2018

# Acknowledgement


- These lecture slides are based on slides by Professor Simon Hood
- Additional sources are cited separately

# Reading Assignment (required)

- Read Data Structures (recommended textbook)
  - Chapter 1 sections 1.6 – 1.9
  - Chapter 2 sections 2.4 and 2.6

  Note the textbook does a few things we might consider poor style, for example one-letter variable names and using int for Boolean values.

# Background

- In programming we often need to find an item in a list
  - Any list or collection of items probably needs a way to search it
    - Data not stored by index #, or
    - Data stored by index # (e.g. array) but index # of item not known
  - Database lookups
- We use computers to do complex searches every day
  - Web searches
  - Search files on hard disk for specific name or content
- Most efficient searching involves using tree data structures
  - Even a regular array can be treated as a tree if you're careful

# Linear Search

- The simplest way to search is to look at all the items in order

1. Begin at the first item and check if it's the one we want
2. If it is, stop
3. If it isn't, go to the next item and check it
4. Repeat steps 2 & 3

Everyone should be able to code a linear search!

# Exercise 1

1. Starting with the C code given below, add logic to look up the number the user entered in the array
   - If the number is found, print "Number found at index N"
   - If the number is not found, print "Number not found".

```
int main(int argc, char** argv) {
    int arr[] = { 90, 88, 56, 100, 2, 25 };
    printf("Enter the number to search for: ");
    int num;
    scanf("%d", &num);
    …
}
```

2. Now put your search in a separate function that returns the index number if found, or -1 if not found (it should not print anything)

# Linear Search Complexity

- Obviously, this has a complexity of O(n)
  - We might have to search all the way to the last number, stopping at every other number, before we find the one we're looking for
- It's the easiest search to implement
- This is considered the worst searching algorithm in terms of performance, but it some situations we have no choice
  - Data not organized in a way that lets us search it faster, e.g. random
- We can do better!
  - Must store data in a way that allows faster algorithms

# Sorted vs. Random

- In a random list there are not many options for searching – we may have to use linear search
- We can change how data is stored to make it quicker to search
  - Sorted
  - Indexed
- We know at least six different ways to sort a list quickly!
- In a sorted or indexed list, search performance can be improved dramatically

# Binary Search

- With a sorted list we can perform a binary search
  - Similar to how you find a word in a dictionary
- We can jump to the middle of the list and check if the item we want is higher or lower
  - If it's higher, jump halfway between the middle and the end
  - If it's lower, jump halfway between the beginning and the end
- Each time we repeat with half the previous range
- Stop when the desired item is found

# Binary Search Code

- One possible implementation

```
int binarySearch(int arr[], int n, int value) {
    int first = 0, last = n-1, index = -1;
    while (first <= last) {
        int middle = (first + last) / 2;
        if (arr[middle] == value) {
            index = middle;
            break;
        } else if (arr[middle] > value) {
            last = middle - 1;
        } else {
            first = middle + 1;
        }
    }
    return index;  // Return index of value, -1 means not found
}
```

# Binary Search Complexity

- After each step the size of the remaining list is cut in half
- This gives us a logarithmic complexity, O(log n)
  - In this case we know the base of the logarithm is two
- This is much faster than a linear search!
- For example, searching 1 million numbers
  - Linear search worst case = 1000000 comparisons
  - Binary search worst case = $\text{Log}_2(1000000)$ = ~20 comparisons

# Exercise 2

- Change your program from Exercise 1 to use the binary search function on the previous slide
- The program should have the same output as before
- Remember that a binary search only works on sorted data, so change the array initialization to be in sorted order!

# MCQ

Suppose you have a list to search that's not sorted, for example a list of all Sheridan students. The fastest way to search for a particular item is

a) Sort the list first using the most efficient sorting algorithm, then use binary search
b) Linear search
c) Both are equal

# Exercise 3

- You're given the sorted list of numbers shown below

  { 2, 11, 25, 37, 56, 61, 63, 88, 90, 100 }

1. Suppose you're searching for the number 61 using a linear search (starting at the 1st number). Write down all the numbers checked, in order.
2. Now write down all the numbers checked for a binary search, in order. Use the algorithm from a few slides back.