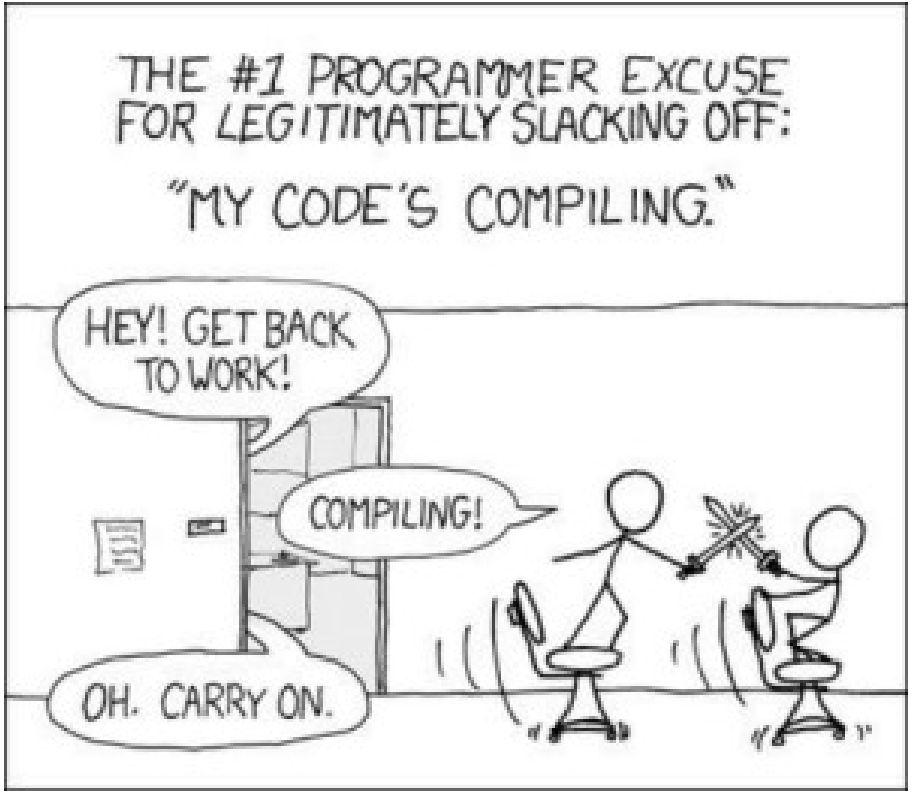

THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."
HEY! GET BACK TO WORK!
COMPILING!
OH. CARRY ON.

CS2308
Gentry Atkinson

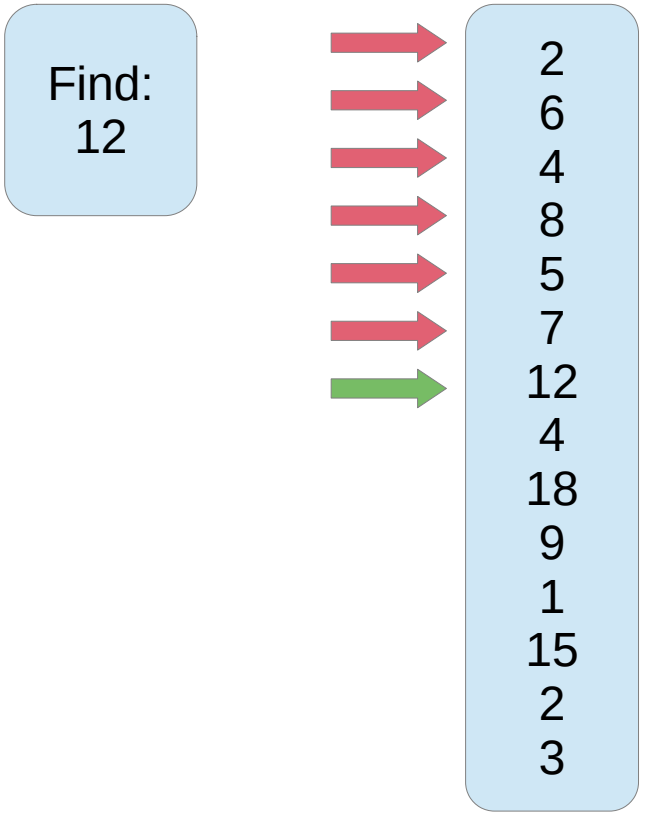
Lecture 2.2
Searching

Searching a List

- Imagine you:
  - Want to see if a movie you like is available on Netflix.
  - Want to see if a class you like is being taught in the Fall.
  - Want to see if a payment you made to your phone company has appeared on your bank statement yet.
- Each of these tasks is algorithmicly identical: searching.

How to Search a List

- Proposed algorithm:
  - Select a target value.
  - Check every value in the list, one at a time, and compare it to the target value.
  - If you get through the whole list, then your target isn't there.



Sequential Search

- Given a list of values **L** and a target value **T**.
- For each element in L:
  - If **Li** matches **T**, output the position **i** and exit
- Output -1 to indicate that **T** is not in **L**

```
int L[] = {2, 6, 3, 8, 4);
int T = 3;
for(int i = 0; i < SIZE; i++){
    if(L[i] == T)
        return i;
}
return -1;
```

Complexity of Sequential Search

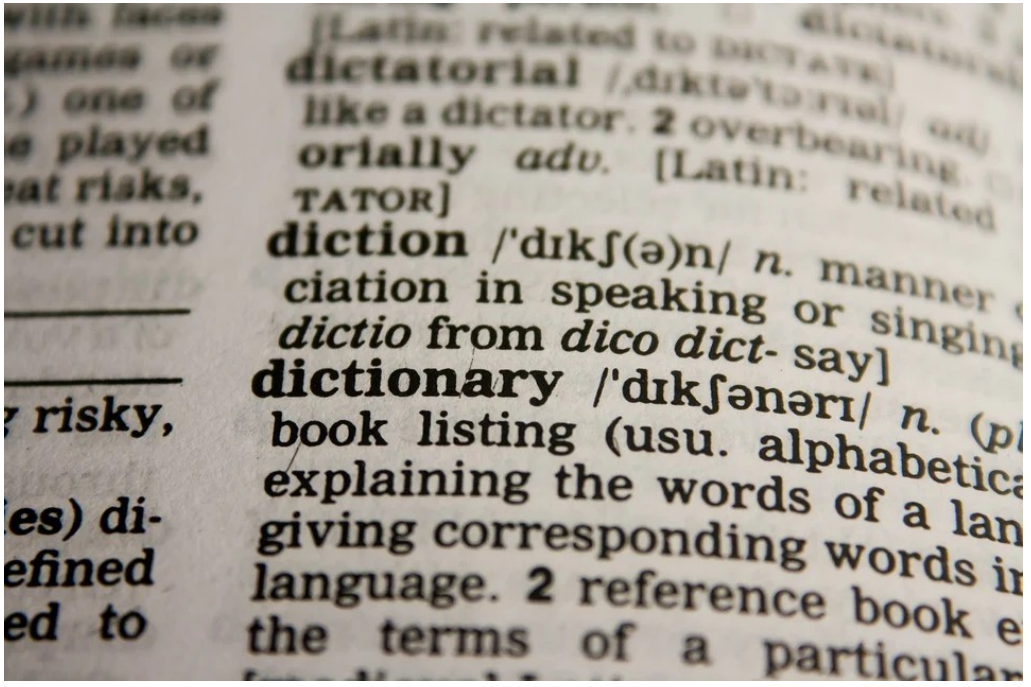
- Most of the “work” of the algorithm is performed in the for loop.
- The loop runs until the target value is found.
  - Best case, the first element is the target. The loop only runs once.
  - Worst case, the target is not in the list. The loop runs **n** times.
  - Average Case, the loop will run about **0.5n** times.
- Ignoring the constant 0.5, we find that Sequential search is O(n), or linear complexity.

Is O(n) good enough?

- Algorithms with linear complexity add a fixed number of extra actions every time the size of **n** (the list size) increases by 1:
  - n = 1000 → 5000 actions
  - n = 10000 → 50000 actions
- Modern computers can do billions of operations per second, so using sequential search on a very long list is usually fine.
- But consider a list which must be searched *many times*
  - E.g a website with 1,000,000 users that must check to see if a user name is valid several times a minute.

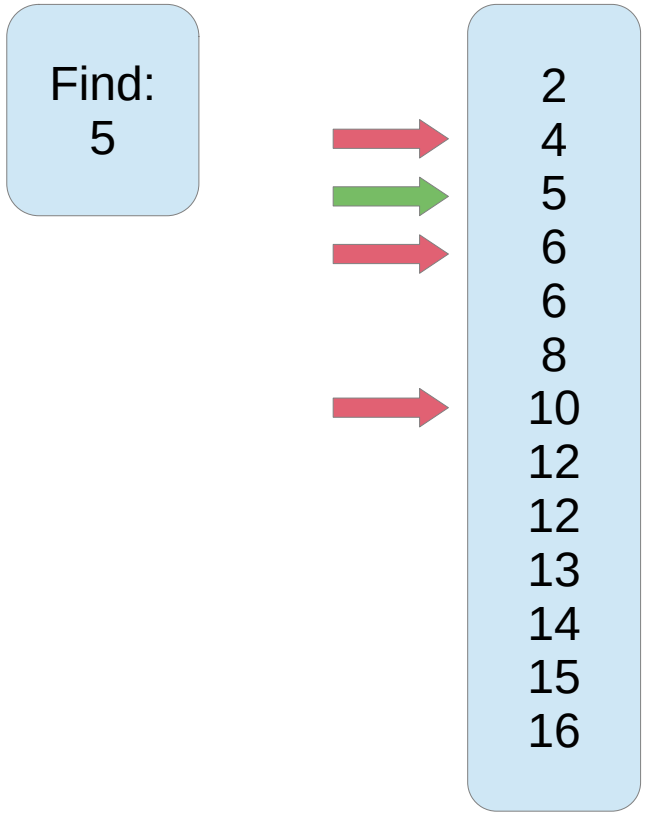
Can we beat O(n)

- Think about looking up the word “bobcat” in the dictionary.
- You wouldn’t check aardvark, Aaron, aback, abacus…
- Instead we jump into the middle, then thumb back, to ‘b’
- We can take advantage of sorted data to improve searching.



Search an Ordered List

- Proposed algorithm:
  - Start in the middle of the list.
  - If you found the target value, you're done.
  - If you've gone past the target value, search the first half of the list.
  - If you have gone far enough, search the second half of the list.



Binary Search

- Given a list **L** of length **n** and a target value **T.**
- Set **i** to 0 and **j** to **n-1**
- While j>i
  - If L(i+j)/2 is equal to T: output (i+j)/2 and exit.
  - If L(i+j)/2 is greater than T, set j to (i+j)/2
  - If L(i+j)/2 is less than T, set i to (i+j)/2
- Output -1 to indicate that T is not in the list and exit.

```
int L[] = {2, 3, 4, 5, 8, 10};

int T = 3;

int i=0, j=5;

while(j>i){
        if (L[(i+j)/2] == T) return (i+j)/2;
        else if (L[(i+j)/2] > T) j = (i+j)/2;
        else i = (i+j)/2;
return -1;
```

Binary Search Walk-through

int L[] = {2, 3, 4, 5, 8, 10, 11};

int T = 3;

i	j	(i+j)/2	L[(i+j)/2]
0	6	3	5
0	3	1	3

Complexity of Binary Search

- Every iteration of the while loops eliminates half of the list.
- Actions required:
  - Best case, the target is in the middle of the of the list and the loop only executes once.
  - Worst case, the item isn't in the list and the loop executes $\log_2(n)$ times.
  - Average case, the loop runs $\log_2(n)/2$ times.
- Binary Search is O(log n) or log time which is a lower order of complexity than linear time.

Logarithm Refresher

- A logarithm is like an anti-exponent.
- 2^4 is 16, so $\log_2(16)$ is 4.
  - $\log_2(32) = 5$
  - $\log_2(64) = 6$
  - $\log_2(128) = 7$
- Every time a value doubles its logarithm base 2 increases by one, so logarithms grow very slowly.

When to Use which Search?

- Always use **Binary Search** for a sorted list.
- Only use **Sequential Search** for unsorted lists.
- If a list will be searched many times, it might be faster to sort the list first and then apply **Binary Search**.

Questions or Comments?