Week 3
This Week

• GDB
• Arrays
• Call Stack
• Recursion
• Search
• Asymptotic Notation & Running Time
• Sort
GDB

break – tell the program to ‘pause’ at a certain point (either a function or a line number)

step – ‘step’ to the next executed statement

next – moves to the next statement WITHOUT ‘stepping into’ called functions

continue – move ahead to the next breakpoint

print – display some variable’s value

backtrace – trace back up function calls
Arrays

• Set of variables of the same type sharing a name.
• Access each individual element in an array using an index value.
• Passed by reference, rather than by value.
Passing By Reference

Pass by Value – passing a copy of the data stored in a variable.

Pass by Reference – passing the address at which the original copy of the data is stored.
Passing By Reference

Mailing Address vs. Contents of Mailbox

## Mail Center
Cambridge, MA
02138

Reference
Value
Arrays

- Arrays are passed by reference.
- Passing an array gives a function the address of the *start of the array*.

```c
int numbers[6];

```

(numbers)
Arrays

- Name of the array refers to starting position.
- `numbers[i]` accesses the $i^{th}$ element

```c
int numbers[6];
```

```
```

(numbers)
Recursion
What is Recursion?

- *Recursion* - a method of defining functions in which the function being defined is applied within its own definition

- A recursive function is a function which *calls itself*
Components of Recursive Function

• Recursive Call – part of function which calls the same function again.

• Base Case – part of function responsible for halting recursion when appropriate.
Recursion Example

- Natural recursive function: Factorial

\[
n! = \begin{cases} 
1 & \text{n = 1} \\
n \times (n-1)! & \text{n > 1}
\end{cases}
\]
Recursion Example

```c
int factorial (int n)
{
    //Base Case
    if (n == 1)
        return 1;

    //Recursive Call
    else
        return n * factorial(n - 1);
}
```
Recursion Example

Diagram showing recursive calls to calculate the factorial of 5.
Call Stack

- Each call to a function gets its own stack ‘frame’ containing that particular instance’s local variables.

- When a function calls a function, it creates a new stack frame ‘below’ the previous one.
Call Stack

main()

func1()

func2()

func3()
We Need To Go Deeper
**Inception**

<table>
<thead>
<tr>
<th>reality?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dream1</strong></td>
</tr>
<tr>
<td><strong>dream2</strong></td>
</tr>
<tr>
<td><strong>dream3</strong></td>
</tr>
</tbody>
</table>

"The Kick"
Linear Search

• Want to find a particular element in an array?
• Search through each element until we find it!

Have you tried linear search?
Binary Search

• Like searching through the phone book.
• Identify whether it’s in the first or second half, dividing the problem in two. Repeat until we’re down to one element.
Asymptotic Notation

• Used to describe the runtime of an algorithm.
• Runtime measured in terms of how the amount of time taken changes as input size increases.
Asymptotic Notation

• Big $O$ notation describes an upper bound on runtime.
• $\Omega$ describes a lower bound on runtime.
• $\Theta$ describes both an upper and lower bound.
Asymptotic Notation

• 1 – Constant Time
• \( \log n \) – Logarithmic Time
• \( n \) – Linear Time
• \( n^2 \) – Quadratic Time
• \( n^p \) – Polynomial Time
• \( 2^n \) – Exponential Time
• \( n! \) – Factorial Time
Runtime of Search

So...
What’s the asymptotic running time of:

• Linear Search?

• Binary Search?
Bubble Sort

• Sorting an array of size $n$.
• Iterate through the array, comparing each pair as you go. If array[$i + 1$] < array[$i$], swap them!
• Repeat up to $n - 1$ times (or until array is completely sorted...).
Selection Sort

• Grab the smallest and swap it with whatever is at the front of the array.
• Now grab the next smallest and swap with what is in position 2.
• Repeat until entire array is sorted (again, up to \( n - 1 \) times).