This is Week 3

Jason Hirschhorn

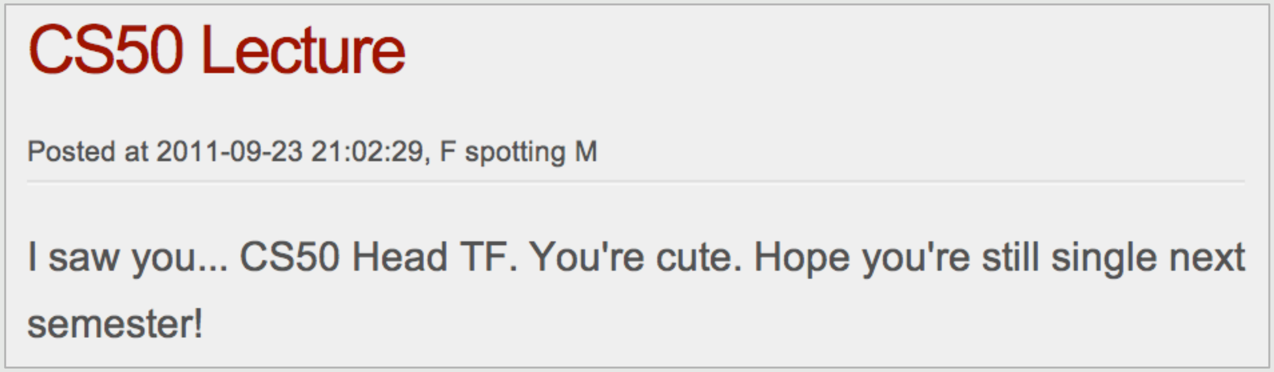
Fall, 2011

Agenda

- CS50 Resources
- Review
  - Problem Set 1
  - Arrays
- GDB
- Running Time
  - Asymptotic Notation
- Search & Sort
  - Linear; Binary
  - Bubble; Selection
- Recursion
  - Call Stack

CS50 Resources

- Problem Set 3 Walkthrough (Sun, 7pm, NW Labs B103) – https://www.cs50.net/psets/
- Office Hours – https://www.cs50.net/ohs/
- Lecture videos, slides, source code, Scribe Notes – https://www.cs50.net/lectures/
- Bulletin Board – http://help.cs50.net
- Me – jchirschhorn@gmail.com
- Problem Set feedback and scores
  - pset0 – all ready sent out!
  - pset1 – Monday
  - pset2 – Friday
- We're here to help you. Plus…



CS50 Lecture

Posted at 2011-09-23 21:02:29, F spotting M

I saw you... CS50 Head TF. You're cute. Hope you're still single next semester!

Review

pset1 – Correctness

- Check for invalid inputs

```
if(argc != 2)
{
      printf(“Enter a key.\n”);
      return 1;
}
```

- Check for corner cases
  - Zero
  - Negatives
  - Characters instead of numbers

pset1 - Design

- Make it easy on yourself! Don’t do unnecessary work
- Don’t check conditions you know are true

```
if(x == 5) { // do this }
else if(x != 5) { // or this}
```

- Don’t create extra variables
  - Bad

```
int y = x + 3;
int z = y % 4;
```

  - Good

```
int result = (x + 3) % 4;
```

pset1 - Design

- Ask yourself, "Is there another way I can solve this problem more efficiently?"
  - Problems have many *right* answers but only a few *good* ones
- So, develop a problem-solving strategy
  - Focus on one task at a time
  - Solve the problem in English
  - Write the pseudo-code
  - Translate it into C
  - Try it
  - Repeat for the next task

Arrays

- A set of elements of the same type
- Each element is accessed with an index value

Quick Quiz

- `./ohai cs50 section pals`
  - What is argc?
  - What is argv[0]?
  - What is argv[1][2]?
  - What is argv[3][4]?

Arrays

- “Passed by reference” (not by value)
  - Pass the location where the original copy is stored
- We tell a function where to find the start

```
int numbers[3] = {4, 5, 6};
int s = sum(numbers);
```

- E.g. mailing address vs. contents of the mailbox

Sum.c

- Concepts to practice – function calls, arrays

```c
#include <stdio.h>

// sums the numbers in a given array
int sum(int array_size, int numbers[]);

int
main(void)
{
      // initialize an array of 5 numbers
      // call the sum function
      // print the result
}
```

GDB

GDB

- GNU Debugger
- Allows you to walk through your program step by step
  - Pause at any step and find out what everything equals
  - Way more powerful than printf*
- To start, type `gdb <program name>` in terminal
- Let’s check out how to walk through a program, gdbexample.c

*Nevertheless, have I always used printf instead? Yes, yes I have.

Useful Commands

- `run <optional command line args>`
  - Run the loaded program
- `break <function name or line number>`
  - Create a breakpoint (where the program will)
- `step`
  - Execute the next line of code (enter a function)
- `next`
  - Execute the next line of code (w/o entering a function)
- `continue`
  - Go to the next breakpoint
- `list`
  - List the source code around the current line
- `print <variable name>`
  - Display the value of a variable

Running Time

Running time

- How long it takes an algorithm to run
- Not in terms of (nano)seconds
  - That would vary by computer
- In terms of "steps"

Why?

- One algorithm may solve a problem faster than another
  - As the size of the problem increases, it may solve it *way* faster
- Asymptotic notation allows us to represent and compare these running times

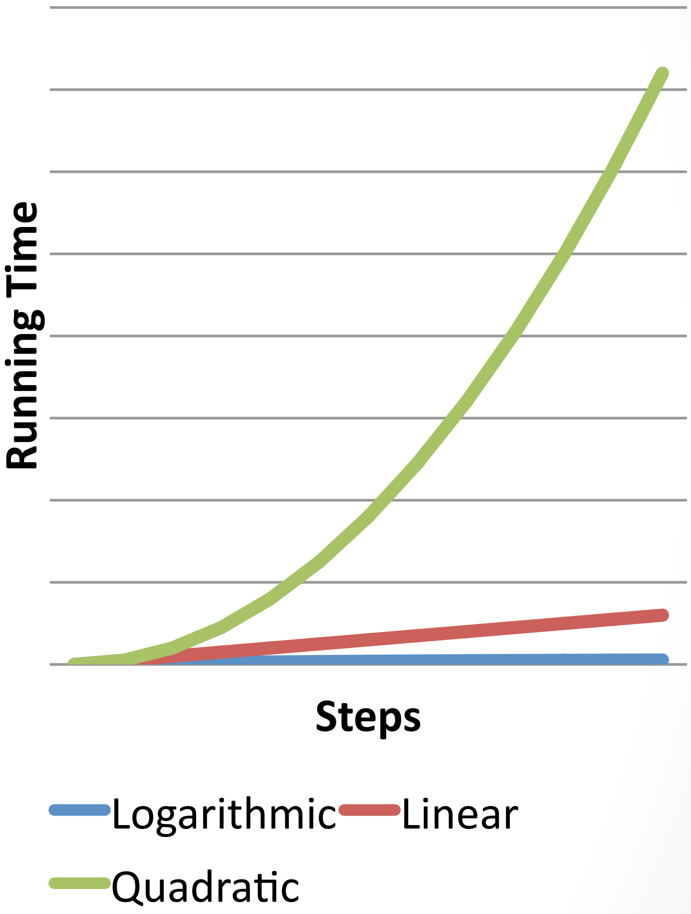
Asymptotic Notation

- O
  - “Big O”
  - Worst case running time (upper bound)
  - Most important to look at when classifying the speed of an algorithm
- Ω
  - “Omega”
  - Best case running time (lower bound)
- Θ
  - “Theta”
  - Average case running time (upper and lower bound combined)

Asymptotic Notation

- $O(1)$ – constant
- $O(\log n)$ – logarithmic
- $O(n)$ – linear
- $O(n^2)$ – quadratic
  - $O(n^c)$ – polynomial
- $O(c^n)$ – exponential
- $O(n!)$ – factorial

- $O(n) = O(kn)$, where k is a constant
- $O(n^c + n^k) = O(n^c)$ where $c > k$



Efficiency Matters

Quick Quiz

- What’s wrong with this code?

```
for(int i = 0; i < strlen(word); i++)
{
        printf(“%c\n”, word[i]);
}
```

- Design decisions like this one matter in terms of how efficiently your code runs
- Complexity is the same way

Search & Sort

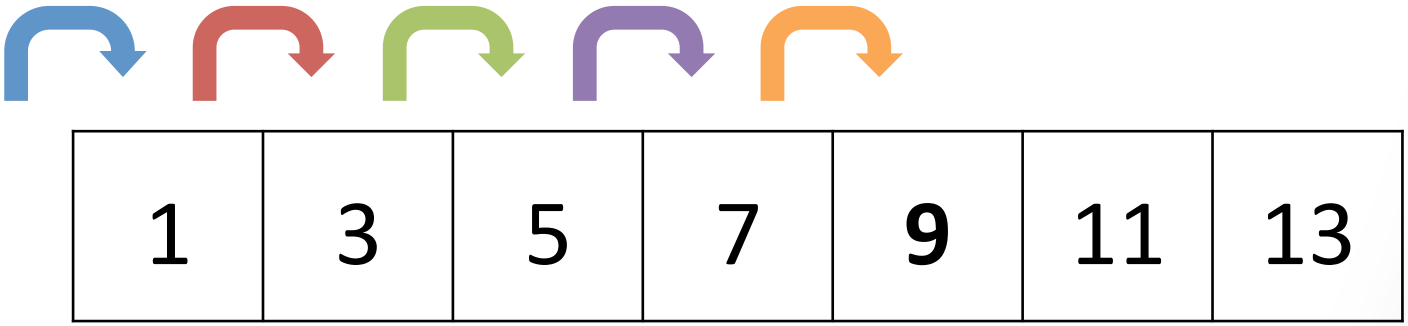
Linear Search

Method

- Iterate through each element in a list until we find the one we want
  - List may or may not be sorted

Big O

- O(n), Ω(1)



1	3	5	7	**9**	11	13

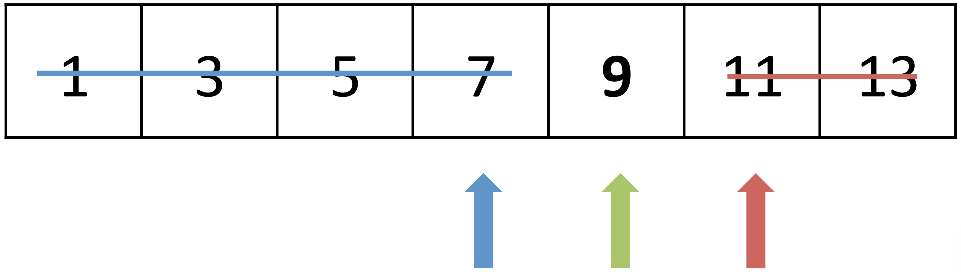
Binary Search

Method (must have sorted list)

- Start in the middle
- If this is the right number
  - All done!
- Else if too high
  - Divide in half
  - Ignore right half
  - Repeat on left half
- Else if too low
  - Divide in half
  - Ignore left half
  - Repeat on the right half

Big O

- O(log n), Ω(1)



~~1~~	~~3~~	~~5~~	~~7~~	**9**	~~11~~	~~13~~

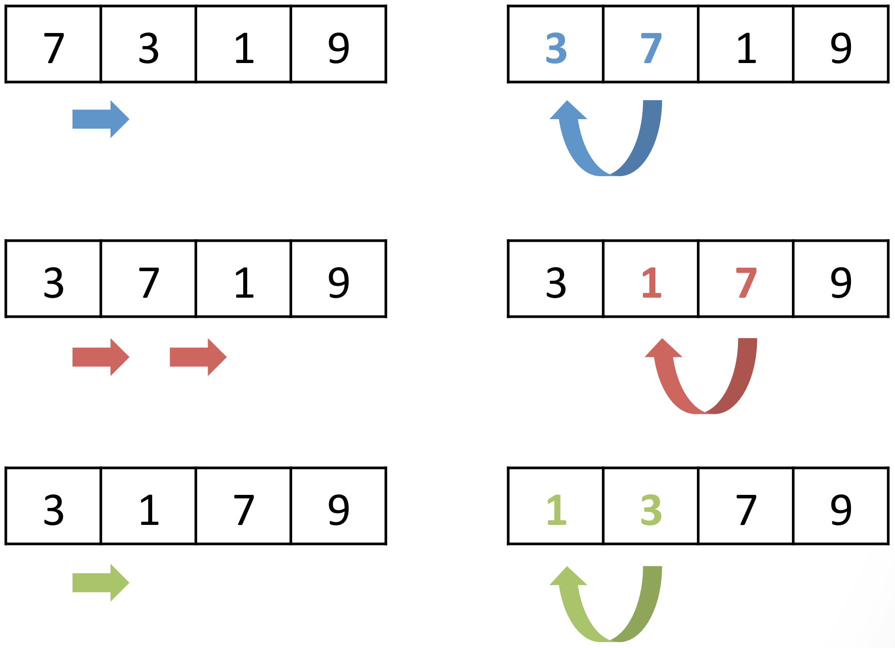
Bubble Sort

Method

- If adjacent elements are out of place, swap them
- Keep going through the list until no swaps are made

Big O

- $O(n^2)$, $\Omega(n)$



7 3 1 9

3 7 1 9

3 7 1 9

3 1 7 9

3 1 7 9

1 3 7 9

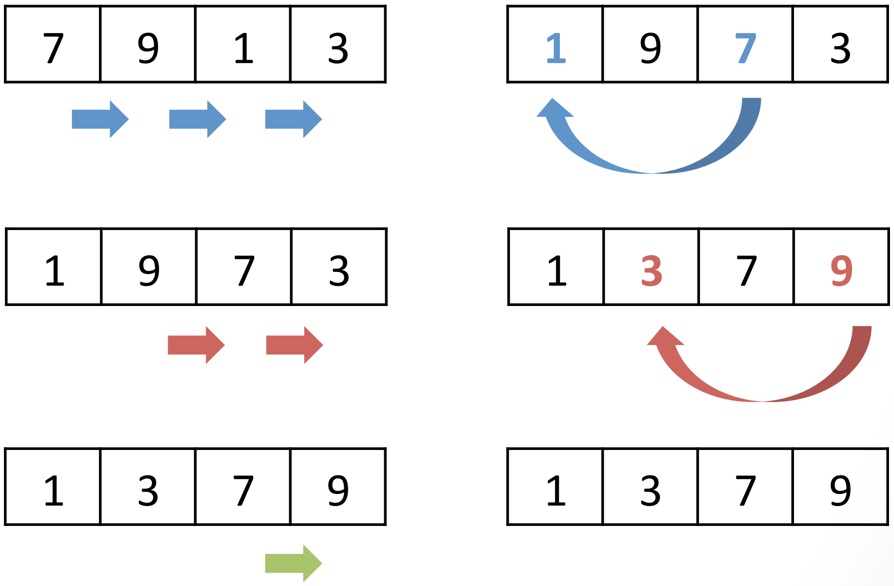
Selection Sort

Method

- Find the smallest element and swap it with the first element
- Find the next smallest element and swap it with the second element
- Repeat until the end of the list

Big O

- $O(n^2)$, $\Omega(n^2)$



7 9 1 3

1 9 7 3

1 9 7 3

1 3 7 9

1 3 7 9

1 3 7 9

Recursion

Recursion

- A function that calls itself
- Base case
  - When the function should stop calling itself
  - Stops the function from calling itself forever
- Recursive call
  - When the function calls itself again

Recursion Example

```
int
length(char *word, int n)
{
     if(word[n] != ‘\0’)
           return 1 +
             length(word, n + 1);
     else
           return 0;
}
```

Recursive call

Base case

Recursion Example

Recursive

```
int
length(char *word, int n)
{
  if(word[n] != ‘\0’)
    return 1 +
      length(word, n + 1);

  else
    return 0;
}
```

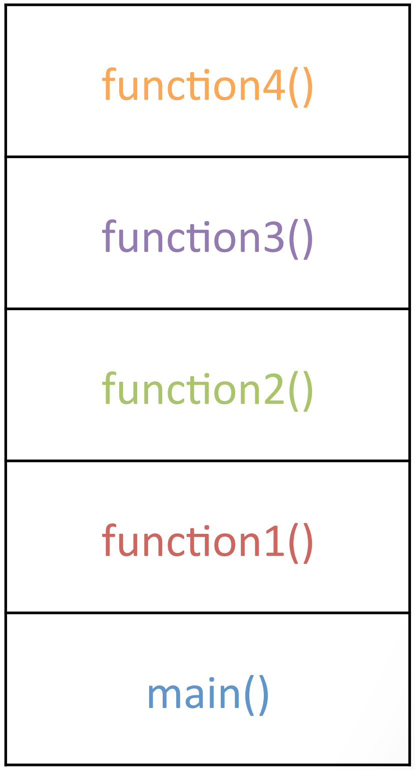
Non-Recursive

```
int
length(char *word, int n)
{
  while(word[n] != ‘\0’)
    n++;

  return n;
}
```

Call Stack

- Every function gets its own space in memory (“frame”)
- When a function is called, it creates a new frame
- Frames stack on top of each other
- Top frame = active frame
  - After it finishes it disappears
  - The frame below it becomes active



function4()

function3()

function2()

function1()

main()

Factorial.c

- Concepts to practice – command line arguments, validating input, function calls, recursion

```
#include <stdio.h>
#include <stdlib.h>


// finds the factorial of a given number
long long factorial(long long n);

int
main(int argc, char *argv[])
{
        // validate user input
        // call the factorial function
        // print the result
}
```

That was Week 3

http://www.youtube.com/watch?v=zlfKdbWwruY