Topics Covered

CS50 Study

The material for this week may get a little confusing. Make sure to use your resources, such as CS50 Study`https://study.cs50.net/binary_search?toc=binary_search,bubble_sort,insertion_sort,selection_sort,merge_sort`

Arrays Review

What data types can you use arrays for?

Do you need a \0 after every type of array?

Asymptotic Notation

We measure computational complexity (aka "Big O") by comparing how quickly the number of operations ittakes to complete the function or program grows as we increase the size of the input to some arbitrarily largevalue. This lets us know how the well the algorithm scales with larger problems. When we talk about "inputsize", we mean whatever makes sense with respect to the algorithm. (For a string, probably the length of the string; for an array, probably the number of elements in the array...) Just like the upper bound ("big O") we also pay attention to the lower bound Ω, on an algorithm's running time. The lower bound corresponds to an algorithm's best case scenario.
Some examples of Upper/Lower bounds:

O(n), O(1), O(n^2), O(log(n)), O(n log(n))

Sorting Algorithms

Example 1: Linear Search
What is the upper bound O, and the lowerbound Ω?
Hint: Think back to David's phone book example from class.

When you perform a linear search in the phone book, it could be on the first page, and therefore, it would only take one operation. It could also be on the last page, in which case it would take n operations.

Example 2: Binary Search
Binary search is an algorithm than runs, in the worst case, in O(log n) time and in the best case, (1) time. Think back to the phone book example David did in class: if you get lucky and the first page you choose includes the person youre looking for, that took only 1 operation. Otherwise, you will have to do max log(n) operations to find them.

This only works if the list youre looking through is sorted. Otherwise, it would be impossible to know for certain that the halves of the list youre discarding dont have what youre looking for. to free variables, we have

Example 3: Bubble Sort Bubble Sort runs in O(n2) time complexity. It works on an array of size n by iterating across the unsorted part of the array, swapping adjacent items that are out of place. In this way, larger elements tend to bubble to the top.

For example, lets say youre given array [4, 1, 7, 10, 3]: what are the steps?

The lower-bound on runtime is (n), though this requires us to use a swap counter or flag to know if we didnt make any swaps (in which case the array is already sorted!)
Why do you need to know how many swaps were made?

Can you explain the math behind the runtime?

For practice, write out the pseudocode!

Example 4: Selection Sort Selection sort runs in O(n2) time complexity. It works on an array of size n, building the final sorted array by removing, one at a time, the smallest element in the unsorted array and placing it at the head of the sorted array.
For example, with array [50, 1, 42, 4, 51]:

The lower-bound on runtime is (n2). Selection sort is dumb, and always will have to find the smallest element on each pass.

Example 5: Insertion Sort Insertion sort runs in O(n2) time complexity. It works on an array of size n, building the final sorted array one element at a time by removing, one at a time, elements in the unsorted portion of the array and placing them in their correct position in the sorted array.
For example, with array [2, 8, 1, 4, 3]:

Insertion sort is in many cases better than bubble and selection Sort. Whereas both bubble and selection sort in normal cases, without any optimizations, run in the best case n2, insertion sort will run in n. But the upper bound on runtime is nevertheless n2.

The lower-bound on runtime is (n2). Selection sort is dumb, and always will have to find the smallest element on each pass.

Example 6: Merge Sort Merge sort is very different from the previous three sorting algorithms. It is most easily implemented using recursion instead of iteration

If worried your students might be uncomfortable with recursion, heres a sample program showing both an iterative and recursive approach to summing integers.

And second, it runs in $nlog(n)$ time making it significantly faster than the other algorithms. Its power comes from its divide and conquer approach. Think of it like this: whereas the other sorting algorithms take n comparisons n times as they traverse an array, merge sort only has to do n comparisons log(n) times since the size of the array is halved each recursive call (think back to the phone book from binary search!)
Let's try an example with array, [3,5,2,6,4,1]:

Below is the code for Merge sort.

```
int mergesort(int x, int y[])
{
    if (x == 0)
        return y[0];
    else
        return mergesort(x - 1, y) + y[x];
}
```

Algorithms Summary

Table 1: Runtime Summaries

Algorithm Name	Basic Concept	O	Ω
Selection Sort	Find the {smallest} unsorted element in an array and swap it with the first unsorted element of that array.	n^2	n^2
Bubble Sort	Swap adjacent pairs of elements if they are out of order, effectively bubbling larger elements to the right and smaller ones to the left.	n^2	n
Insertion Sort	Proceed once through the array from left-to-right, shifting elements as necessary to insert each element into its correct place.	n^2	n
Merge Sort	Split the full array into subarrays, then merge those subarrays back together in the correct order	n log n	n log n
Linear Search	Iterate across the array from left-to-right, trying to find the target element.	n	1
Binary Search	Given a sorted array, divide and conquer by systematically eliminating half of the remaining elements in the search for the target element.	log n	1

Recursion

Recursion is a divide-and-conquer approach to problem solving that leverages solutions to smaller problems (for which solutions are axiomatic or much easier to determine) to help in determining the solution to a larger one.
Draw Sam's machine/diagram on this page!

Each recursive procedure has two pieces:
1. Base case (where recursion stops, the simple solution is given)
2. Recursive case (where the problem is made a little bit smaller but another functioncall is made and another frame is placed onto the stack)

Example of recursion: Code out factorial in blank space below

Recursion is great! However, it has downsides, especially given how memory-intensive it can be to employ. And while a recursive algorithm is not always required, it frequently looks much more beautiful and (though recursion is not itself a simple concept), a recursive implementation usually looks much simpler once coded.