CSE13S notes
Week of 1/30/23

Assignment 3: Sortin time

- Sorting is the act of putting things into a defined order
- Dictionaries are sorted in what is called lexicographical order
    - Lexicographical = alphabetical order
- Numbers can be sorted in their natural order, or reverse order (low to high or high to low)
- There are both toal and partial orderings, but we will only concern ourselves with total orderings (for now)

Why sort?

- Sorting adds information to our data
- For examples, we now can make assertions about before, after, lesser, greater, etc.
- Here are a few examples:
    - We can search most efficiently in sorted data
    - We can merge sorted list efficiently
    - We can detect duplicates efficiently
    - We can find the median efficiently

First idea:

- Enumerate all of the possible orderings of n objects
    - There are n! such orderings
- Pick the one that is in order
- On second thoughts, that was a bad idea

Second idea:

- Pick smallest element from a list of n elements (you must go through all n elements)
    - Put it in slot 1, you can ignore that slot now
    - Swap that element in the first slot with the selected element
- This is selection sort

Third idea:

- Suppose we have array with 1 element
    - This array is sorted
- Given a second element, it must go either before or after the first element
    - The 2 element array is sorted
- This is slightly better than selection sort
- You can make selection sort better by using binary search
    - Since list being inserted into is already sorted
    - However, this is still somewhat slow and requires a lot of comparisons

Fourth idea

- Given unsorted array:
- If element 1 is greater than element 2, swap them.
- Continue doing for each pair of element in the list until sorted
- Bubble sort

Is that the best we can do:

- No, we will learn in later classes where we can prove that we can sort in time proportional to n log n
- Suppose array is examined in disjoint pairs: a[0], a[1], then a[2], a[3]
    - We have n/2 such pairs, but we have to look at both so we can call it n
    - We put the elements of every pair in order (length 2), and call it a run
- Now we take a run, each of length 2, and merge them into runs of length 4
    - Again, this will take us time proportional to n
- How man times can we double 1 before we exceed n? Thats easy, log n
- Thus, our sort finishes in time n log n, this is mergesort

Comparitive sorting algorithms

- Quicksort
    - Fastest average algorithm for sorting, especially for large number of elements
- Heap sort:
    - A heap is a binary tree
    - A single node is a heap
    - It is a heap if the parent is a heap and the trees rooted at both children are heaps
    - A parents value if greater than that of either child
    - There is no order among children
- Using comparisons, these are the fastest algorithms
- But, if we make some assumptions about the encoding then you can use a radix sort
    - It runs in time proportional to the number of digits in the key imes the number of records
    - It was invented to mechanically sort punched cards

Summary

- Sorting is a fundamental operation, so doing it efficiently has a huge impact on computing
- Analysis of algorithm complexity is an important topic, and you will be a lot of it in you advanced classes
    - A better algorithm makes a lot more difference than a faster computer

- You will encounter many instances in you career where you need to employ a sorting algorithm, and the best one will depend on the circumstances and the data structures employed.

Computational complexity (simplified)

- BIG O = on the order of
    - O(log(n)) = binary search
    - O(n) = find minimum
    - O(n*log(n)) = merge sort
    - O(n^2) = Bubble sort
    - O(n^3) = Matrix Multiply
    - O(2^n) = Enumerate subsets
    - O(n!) = Enumerate permutations
- When we say O(n^2) we're not being very precise
- There's some point in time (or number) where O(g(n) becomes true

Estimating complexity

- See slides

Constant "*c*"

- You can thing of *c* as the overhead an algorithm requires
    - This is why for ~50 items or less, bubble sort is smarter than quicksort
    - Many factors contribute to the overhead, like having to copy the entire array for mergesort, if statements, etc.
- Time space trade of
    - Look up memoizing

Summary

- Computer scientists talk about the complexity of algorithms all the time
- Complexity can measure time, space, or both
    - Time is "how many steps it takes" and
    - Space is "how much memory it uses"
- The algorithm that you choose has the largest impact on performance of you program
- For small problems, it may be better to use a worse algorithm if the better algorithm that a larger constant
    - But you need to take the time and examine it carefully

Who's Counting?

- We talk about the size of a problem
    - How long is the string
    - How large is the number?

Bit Vectors and sets

- Units of info
    - Bit = 1 bit, can represent 0 & 1
    - Nibble = 4 bits, an represent hex digits

Logical shifts

- Left: Same as arithmetic shift left, see below
- Right: zeroes are shited in of the left

Arithmetic shift

- Left: Zeroes are shifted in on the right (multiplication by 2 for each shift)
- Right: sign bits are shifted in on the left (division by 2 for each shift)

- Bitwise operations in C
    - & - AND
    - | - OR
    - ~ - NOT
    - ^ - XOR
    - >> - Left shift
    - << - right shift
- In C, we do NOT know if shift is logical or arithmetic, it is implementation defined
    - ONLY shift unsigned integers

Sets

- Well-defined unordered* collections that are characterized by the elements they contain
    - Sets can be ordered, says Long (or rather, that we will order them)
- Set intersection
    - AND operator (returns bit only if both sets contain that bit)
- Set Union
    - OR operator
- Set Difference
    - NAND or complement + and
- Set complement
    - NOT or complement
    -