Math 416

Worksheet 4. Sorting things

Problem 1. (Binary search)

(a) Write the Binary Search algorithm in pseudocode. (*Hint:* You will probably want to write a recursive algorithm, i.e., an algorithm that calls itself on an input that is in some sense smaller.)

(b) Discuss why only $O(\log n)$ steps are needed for Binary Search to determine whether b is in $L = [a_1, \dots, a_n]$.

Comparison sorting We consider the basic problem of taking a list of n numbers $a_0, \dots, a_{n-1}$ and returning a permutation $(a'_0, \dots, a'_{n-1})$ of the input sequence that is *sorted*: $a'_0 \leq a'_2 \leq \cdots \leq a'_{n-1}$. Today we will practice sorting with decks of cards. We will take the following ordering convention:

(1) Suits are most important and are ordered alphabetically: Clubs ($\clubsuit$), Diamonds ($\diamondsuit$), Hearts ($\heartsuit$), Spades $\spadesuit$.

(2) Card number is second most important, and are ordered from lowest to highest as $2, 3, \dots, K, A$.

Feel free to work all together at the table to run the sorting algorithms or to work in pairs.

Problem 2 (Merging two sorted list)**.** Consider the problem of taking two *sorted* piles $a_1, \dots, a_n$ and $b_1, \dots, b_n$ of cards and merging them into a sorted pile of cards of length $2n$. One way would be to stack a pile of cards on top of the other and then sort, but we can do better.

(a) Practice sorting some piles of cards by hand, and come up with an algorithm as you work.

(b) Describe in pseudocode an algorithm that will solve this problem in $O(n)$ time.

(*Hint:* Maintain pointers current(a) and current(b) pointing at the top card in each pile. While each list is nonempty, append the minimum of the cards pointed to by current(a) and current(b) to the output list, and advance the corresponding pointer.)

(c) Explain why your algorithm halts in $O(n)$ steps.

Problem 3 (Insertion Sort)**.** The idea behind Insertion Sort is this. Process the input data from left to right, starting with a_0. When you encounter a_i, swap it down into the subarray $a_0, \dots, a_{i-1}$ in such a way that the subarray remains sorted.

(a) Run Insertion Sort on several piles of cards.

(b) Write Insertion Sort in pseudocode.

(c) Identify loop invariants for loops in your pseudocode and use them to explain why Insertion Sort is correct. (*Hint:* You should have two loops.)

(d) Argue that Insertion Sort halts in $O(n^2)$ steps.

(e) Of all the permutations of the numbers $1, 2, \dots, n$, which one will Insertion Sort take the most swaps to sort? How many swaps will it require?

We can do better than Insertion Sort, asymptotically, using an idea from Problem 2.

Problem 4. (Mergesort)

(a) Carry out the following sorting algorithm with a deck of cards (it is very helpful to use partners here!).

Divide the input deck of cards $A[0 \cdots n]$ into two halves

$$A[0, \lceil n/2 \rceil] \text{ and } A[\lceil n/2 \rceil + 1, n].$$

Recursively sort each half, and merge the two sorted lists in linear time, as in Problem 2.

(b) Write the algorithm out in pseudocode.

(Call the algorithm from Problem 2 as a subroutine; you needn't rewrite it.)

Lower bounds for comparison sorting We will later see that Mergesort sorts n integers in only $O(n \log n)$ time. Asymptotically, this is the best we can hope for. In this section, we prove that; a rare **lower bound**.

The possible executions of a **comparison sort** (i.e., a sorting algorithm that can only make comparisons between input entries) on input list of a fixed length n can be organized into a **decision tree**, i.e., a binary tree in which every vertex corresponds to a state of the algorithm, and at each vertex we perform a comparison to see which child to move to. The leaves of the decision tree correspond to outputs of the algorithm.

Problem 5. Draw the decision tree for Insertion Sort on input *abc*.

(*Hint:* The first comparison the algorithm makes is between a and b. So the root should have two children, corresponding to the two possibilities $a < b$ and $b < a$.)

Problem 6. We consider the decision tree of a comparison sort.

(a) In terms of n, how many possible inputs of length n are there? What does that tell you about the number of leaves of the decision tree?

(b) Suppose that the algorithm halts and produces the output corresponding to leaf ℓ. How can you tell how many comparisons it made during its execution, by looking at the decision tree?

Problem 7. The following two assertions are proved on a problem set and later in this worksheet, respectively.

(1) If the height (= max distance from a root to any leaf) of a rooted binary tree is H, then it has at most 2^H-many leaves. (PS 2)

(2) $\log(n!)$ is $\Theta(n \log n)$. (Problem 10)

Assume for now that these two facts are true. Show how these two facts and Problem 6 can be used to prove the following Theorem.

Theorem. Any comparison sort must perform at least $\Omega(n \log n)$ comparisons to sort a list of n elements, in the worst case.

But we still have to prove that $\log(n!)$ is $\Theta(n \log n)$!

Estimating sums by integrals To analyze asymptotic running times of algorithms, we will often want to estimate expressions of the form $f(1) + f(2) + \cdots + f(n)$ (think iteration!).

Lemma. Suppose that $f\colon (0, n+1) \to \mathbb{R}$ is continuous and increasing. Then the sum $f(1) + f(2) + \cdots + f(n)$ can be bounded above and below as follows.

$$\int_0^n f(x)\,dx \le \sum_{i=1}^{n} f(i) \le \int_1^{n+1} f(x)\,dx.$$

Problem 8. Draw a compelling picture and give a short explanation using the phrase 'Riemann sum' to explain why the Lemma is true.

Problem 9. Apply the Lemma to $f(x) = x^p$ to conclude that $1^p + 2^p + \cdots + n^p$ is $\Theta(n^{p+1})$. Does this agree with what you know about the case $p = 1$?

Problem 10. Apply the Lemma to $f(x) = \log x$. Conclude that $\log(n!)$ is $\Theta(n \log n)$.

(*Hint:* I'm sure you haven't forgotten that $x \log x - x$ is an antiderivative of $\log x$.)