15-451/651 Algorithms, Spring 2019

Homework #1 **Due: January 30, 2019**

This HW has some exercises, three regular problems, one programming problem, and one bonus problem. All problems on written HWs are to be done *individually*, no collaboration is allowed.

The exercises are to fortify your knowledge of the basics. Please solve them! You can talk to the course staff if you need help. However, you should not hand in your solutions. We will release sample solutions later so that you can check your work.

Solutions to the three written problems should be submitted as a single PDF file using `gradescope`, with the answer to each problem starting on a new page.

Submission instructions for the programming problem will be posted on the website and Piazza.

The bonus problem will be released soon; we'll post something on Piazza.

---

0. **(Exercises: Recurrences and Probability.)**

   Solve each recurrence below in Θ notation. As always, prove your answer. For all of these problems $T(1) = 1$.

   (a) Solve $T(n) = 3\,T(\lfloor n/2 \rfloor) + n$.

   (b) Now solve $T(n) = 3\,T(\lfloor n/2 \rfloor) + n \lg n$.

   (c) Finally, solve $T(n) = n^{2/3}\,T(\lfloor n^{1/3} \rfloor) + n$.
   (E.g., we might get this from a divide-and-conquer procedure that uses linear time to break the problem into $n^{2/3}$ pieces of size $n^{1/3}$ each. Hint: write out the recursion tree.)

(25 pts) 1. **(A Box of Sorts (or a Sort of Box)?)** Consider the following problem.

   INPUT: n^2 distinct numbers in some arbitrary order.

   OUTPUT: an $n \times n$ matrix containing the input numbers, and having all rows and all columns sorted in increasing order.

   EXAMPLE: $n = 3$, so $n^2 = 9$. Say the 9 numbers are the digits $1, \ldots, 9$. Possible outputs include:

```
1 4 7       1 4 5       1 3 4       1 2 3
2 5 8   or  2 6 7   or  2 5 8   or  4 5 6   or  ...
3 6 9       3 8 9       6 7 9       7 8 9
```

   It is clear that we can solve this problem in time $2n^2 \lg n$ by just sorting the input (remember that $\lg(n^2) = 2 \lg n$) and then outputting the first n elements as the first row, the next n elements as the second row, and so on. Your job in this problem is to prove a matching $\Omega(n^2 \log n)$ *lower bound* in the comparison-based model of computation.[1]

[1] By an "$\Omega(n^2 \log n)$ *lower bound*", we mean a lower bound of $cn^2 \log n$ for some constant $c > 0$ that is independent of n.

For simplicity, you can assume n is a power of 2. Recall $\lg x = \log_2 x$.

Some hints: Show that if you could solve this problem using $o(n^2 \log n)$ comparisons (in fact, in less than $n^2 \lg(n/e)$ comparisons), then you could use this to violate the $\lg(m!)$ lower bound for comparisons needed to sort m elements (which we prove in Lecture #2). You may want to use the fact that $m! > (m/e)^m$. Also, recall that you can merge two sorted arrays of size n using at most $2n - 1$ comparisons.

(25 pts) 2. **Sorting with Strange Primitives**

In both of the parts below an array A initially contains a permutation of $1, \dots n$. The goal in each case is to give an algorithm to sort the array by applying a linear number of the given primitives.

(a) A *prefix reversal* of A just reverses some prefix of the array. For example if $A = [1, 5, 3, 2, 4]$ this can be turned into $[3, 5, 1, 2, 4]$ with one prefix reversal (of the first 3 elements of A). Give an algorithm for sorting any initial permutation in at most $c_1 n$ prefix reversals. What is c_1 for your algorithm? (Try to make it as small as you can.)

(b) A *block swap* takes a pair of neighboring blocks of A of the same size and exchanges them. For example if $A = [8, 1, 3, 2, 4, 6, 5, 7]$ in one block swap we could get $[4, 6, 5, 7, 8, 1, 3, 2]$, or $[8, 3, 1, 2, 4, 6, 5, 7]$, or $[2, 4, 6, 8, 1, 3, 5, 7]$, but not $[7, 1, 3, 2, 4, 6, 5, 8]$. Give an algorithm for sorting any initial permutation in at most $c_2 n$ block swaps. What is c_2 for your algorithm? (Try to make it as small as you can.)

(25 pts) 3. **More Upper/Lower bounds (Stuck Here in the Middle with You.)**

Consider the problem of finding both the maximum element and the minimum element of an arbitrary set of n distinct numbers, where n is even.

It turns out that $\frac{3}{2}n - 2$ comparisons are required in the worst case to do this. That is, $\frac{3}{2}n - 2$ is a lower bound on the number of comparisions needed. There's also an algorithm that achieves this bound. That is $\frac{3}{2}n - 2$ is an upper bound on the number of comparisons needed. Since these bounds are equal, they are optimal.

(a) Prove the following theorem:

Theorem: Consider a deterministic comparison based-algorithm $\mathcal{A}$ which does the following: Given a set S of n numbers as input, $\mathcal{A}$ returns the largest and the smallest element of S. Prove that there is an input on which $\mathcal{A}$ must perform at least $\frac{3}{2}n - 2$ comparisons.

Hints:

Call an element *top* if it has been involved in at least one comparison and it has never lost a comparison.
Call an element *bottom* if it has been involved in at least one comparison and never won a comparison.
Call an element *free* if it has been involved in zero comparisons.

Call an element *middle* if it has won at least one comparion and lost at least one comparison.

Every element falls into exactly one of these categories, and this classification of elements evolves over time. Initially all elements are free. You know that at the end of a run of any correct algorithm, there are ______free ones, ______middle ones, ______top ones and ______bottom ones.

Define a potential function $\Phi()$ that is a linear function in the number of each type of element. You, the adversary, have to decide the outcome of each comparison to give to algorithm $\mathcal{A}$ when it asks you to compare two elements. By making the "right" choice of the outcome of a comparison, you should ensure that the potential decreases by at most 1 for each comparison. Using this fact, along with the initial value and final value of the potential, prove that algorithm $\mathcal{A}$ must do at least $\frac{3}{2}n-2$ comparisons.

(b) The proof above leads to a optimal deterministic algorithm (in terms of the number of comparisions). Describe one such algorithm.

(25 pts) 4. **Programming: Multi-Medians**

Write a program which takes as input a list of n *distinct* numbers $a_1, a_2, \ldots, a_n$, and outputs the elements of ranks $1, 2, 4 = 2^2, 8 = 2^3, 2^4, \cdots, 2^k$, where 2^k is the greatest power of 2 that is at most n.

Your algorithm should have $O(n)$ running time (deterministic worst-case or randomized expected). *Please include a comment at the start of your program explaining your algorithm, and why it runs in linear time.*

Details on how to submit, grading policy, etc., will be on the course website and piazza soon. Please do not use any built-in functions (or libraries) for sorting or selection.

INPUT: The first line contains n, which is at most 10^6. The second line consists of the numbers $a_1, a_2, \ldots, a_n$ separated by blanks. These numbers satisfy $-10^9 \leq a_i \leq 10^9$.

OUTPUT: The first line of the output is k. The second line consists of the required $k+1$ numbers, separated by spaces.

For example, if the input is:

```
5
8 3 1 2 6
```

then the output is:

```
2
1 2 6
```

Or if the input is:

```
10
12 5 1000000000 6 7 9 10 8 14 23
```

then the output is:

```
3
5 6 8 14
```