CS 598TMC, Fall 2023
Timothy Chan

Homework 1 (due Sep 22 Friday 10am)

Instructions: You may work individually or in groups of at most 3; submit one set of solutions per group to Gradescope. Always acknowledge any discussions you have with other people and any sources you have used (although most homework problems should be doable without using outside sources). In any case, *solutions must be written in your own words.*

1. [*25 pts*] Recall the range minimum query problem: store a sequence of n numbers $a_1, \ldots, a_n$ in a data structure so that given indices i, j, we can quickly compute the minimum of the contiguous subsequence $a_i, \ldots, a_j$. Suppose that in addition, we want to support the following update operation: given index i and number x, change a_i's value to x.

   Describe a data structure with $O(n)$ space, $O(\log n / \log\log n)$ query time, and $O(\log n)$ update time for this problem.

   [Note: a weaker $O(\log n)$ query time instead will get you up to 15 points.]

   [Hint: To get $O(\log n / \log\log n)$ query time, use a tree with a larger degree d (for what choice of d?); you will also need table lookup (like "Method 10" from class)...]

2. [*20 pts*] Consider the problem of maintaining an approximate median of a set S of n numbers, subject to insertions and deletions of elements in S. Here, an *approximate median* is defined as an element that has rank between $0.49n$ and $0.51n$.

   Design and analyze a (very simple) data structure that solves this problem with $O(1)$ amortized update time.

3. [*25 pts*] We have a set S of at most N numbers, and we want to place the elements of S into an array A with M slots, so that for any $x, y \in S$ with $x < y$, the slot for x is to the left of the slot for y. We also want to support insertions of new elements to S.

   One obvious solution is to place x in slot i if x is the i-th smallest element in S. This requires only $M = N$ slots. However, an insertion may require moving $O(N)$ elements to new slots.

   (a) [*7 pts*] Describe a simple solution using $M = 2^N$ slots so that each insertion can be done without moving any elements.

   (b) [*18 pts*] The solution in (a) uses exponentially many slots. We consider a different solution using $M = N^2$ slots, which is described recursively by the following pseudocode[1] (this has some vague similarity with the weight-balanced tree method from class):[2]

[1] I might be sloppy about floors/ceilings and boundary cases here, but hopefully you get the general idea...
[2] Correction: in line 2 of insert, S should be S_L (and similarly, in line 5 of insert, S should be S_R).

rebuild(S, N, A), where $|S| \le N$, and array A has N^2 slots:

1. if $N = 1$ then store the single element of S in the first slot and return
2. let m be the median of S
3. let $S_L = \{x \in S : x < m\}$ and $S_R = \{x \in S : x \ge m\}$
4. divide A into two subarrays A_L, A_R with $N^2/2$ slots each
5. rebuild$(S_L, N/\sqrt{2}, A_L)$
6. rebuild$(S_R, N/\sqrt{2}, A_R)$

insert(S, N, A, x):

1. if $x < m$ then
2. if $|S_L| + 1 \le N/\sqrt{2}$ then insert$(S, N/\sqrt{2}, A_L, x)$
3. else remove all elements of S from A, set $S = S \cup \{x\}$, and call rebuild(S,N,A)
4. else
5. if $|S_R| + 1 \le N/\sqrt{2}$ then insert$(S, N/\sqrt{2}, A_R, x)$
6. else remove all elements of S from A, set $S = S \cup \{x\}$, and call rebuild(S,N,A)

Show that with this method, each insertion moves only an amortized $O(\log N)$ number of elements. Use the following definition of potential:

$$\Phi = \sum_{(S,N,A)} (\max\{|S_L| - N/2, 0\} + \max\{|S_R| - N/2, 0\}),$$

where the sum is over all (S, N, A) that appear during the recursion.

4. [*30 pts*] Consider the dynamic predecessor search problem in the setting where there are only insertions, but no deletions. Balanced search trees solve the problem with $O(\log n)$ insertion and query time, but require $\Omega(n)$ extra space for the pointers. We will explore a completely different solution that uses only a single array *without any extra space*.

   Specifically, after n insertions, the elements are stored in $A[1, \ldots, n]$ in some (not necessarily globally sorted) order. Write $n = b_{\ell-1} \cdots b_1 b_0$ in binary. For each $i = 0, \ldots, \ell$, let n_i be the ℓ-bit number $n = b_{\ell-1} \cdots b_{\ell-i} 0 \cdots 0$. We maintain the invariant that for each i, the subarray $A[n_{i-1} + 1, \ldots, n_i]$ is sorted (whenever $n_i \ne n_{i+1}$).

   [For example: say $n = 11$ (i.e., 1011 in binary). Then $n_0 = 0$, $n_1 = 8$ (i.e., 1000 in binary), $n_2 = 8$, $n_3 = 10$ (i.e., 1010 in binary), and $n_5 = 11$.[3] The invariant says that $A[1, \ldots, 8]$ is sorted and $A[9, 10]$ is sorted. E.g., the array may look like $\langle 12, 15, 19, 30, 31, 40, 48, 54, 24, 35, 17 \rangle$.]

   The insertion procedure is simple (in line 3, recall that heapsort is an optimal sorting algorithm that does not require any extra space):

   insert(x):

   1. $n = n + 1$, $A[n] = x$
   2. find the largest integer k such that n is divisible by 2^k
   3. sort the subarray $A[n - 2^k + 1, \ldots, n]$ by heapsort

[3] Correction: n_5 should be n_4.

[For example: when inserting $x = 20$ in the above example, n becomes 12 (i.e., 1100 in binary), and $k = 2$. The new array is $\langle 12, 15, 19, 30, 31, 40, 48, 54, 17, 20, 24, 35\rangle$.]

(a) [*5 pts*] Argue that the above insertion procedure indeed preserves the invariant.

(b) [*10 pts*] Show how to answer a query in $O(\log^2 n)$ time.

(c) [*15 pts*] Prove that insertion takes $O(\log^2 n)$ amortized time.
[Hint: over a sequence of n insertions, how many times do we have $k = 0$? Or $k = 1$? Etc.?]