CSCI2100: Regular Exercise Set 5
Prepared by Yufei Tao

Problems marked with an asterisk may be difficult.

Problem 1. Let $S$ be a set of 9 integers $\{75, 23, 12, 87, 90, 44, 8, 32, 89\}$, stored in an array of length 9. Let us use quicksort to sort $S$. Recall that the algorithm randomly picks a pivot element, and then, recursively sorts two sets $S_1$ and $S_2$, respectively. Suppose that the pivot is 89. What are the contents of $S_1$ and $S_2$, respectively? The ordering of the elements in $S_1$ and $S_2$ does not matter.

Solution. $S_1 = \{75, 23, 12, 87, 44, 8, 32\}$ and $S_2 = \{90\}$.

Problem 2 (Sorting a Multi-Set). Let $A$ be an array of $n$ integers. Note that some of the integers may be identical. Design an algorithm to arrange these integers in non-descending order. For example, if $A$ stores the sequence of integers $(35, 12, 28, 12, 35, 7, 63, 35)$, you should output an array $(7, 12, 12, 28, 35, 35, 35, 63)$.

Solution. We will apply merge sort as a black box, namely, we do not need to change how the algorithm works at all. Let $S$ be a set of $n$ elements defined as follows: the $i$-th ($1 \leq i \leq n$) element of $S$ equals $(i, v)$ where $v = A[i]$. Create an array $B$ of length $n$, where $B[i]$ equals the $i$-th element in $S$. $B$ can be generated easily from $A$ in $O(n)$ time.

We apply merge sort to sort $B$, but compare two elements $e_1 = (i_1, v_1)$ and $e_2 = (i_2, v_2)$ in the following way:

- If $v_1 < v_2$, then rule $e_1 < e_2$
- If $v_1 > v_2$, then rule $e_1 > e_2$
- If $v_1 = v_2$:
  - If $i_1 < i_2$, then rule $e_1 < e_2$;
  - Otherwise, rule $e_1 > e_2$.

After $B$ has been sorted, we can easily generate the output array from $B$ in $O(n)$ time.

Problem 3. Let $S_1$ be a set of $n$ integers, and $S_2$ another set of $n$ integers. Each of $S_1$ and $S_2$ is stored in an array of length $n$. The arrays are not necessarily sorted. Design an algorithm to determine whether $S_1 \cap S_2$ is empty. Your algorithm must terminate in $O(n \log n)$ time.

Solution. Sort $S_1$ and $S_2$ together as a multi-set (using the algorithm of Problem 2) in $O(n \log n)$ time. Then, scan the sorted list, and check whether there are two identical integers coming from different sets; this can be done in $O(n)$ time.

Problem 4* (Inversions). Consider a set $S$ of $n$ integers that are stored in an array $A$ (not necessarily sorted). Let $e$ and $e'$ be two integers in $S$ such that $e$ is positioned before $e'$ in $A$. We call the pair $(e, e')$ an inversion in $S$ if $e > e'$. Design an algorithm to count the number of inversions in $S$. Your algorithm must terminate in $O(n \log n)$ time.
For example, if the array stores the sequence (10, 15, 7, 12), then your algorithm should return 3, because there are 3 inversions: (10, 7), (15, 7), and (15, 12).

**Solution.** If \( n = 1 \), simply return 0. If \( n \geq 2 \), we divide \( A \) into two halves: (i) the first half includes the first \( \lfloor n/2 \rfloor \) elements, and (ii) the second includes the rest. Let \( A_1 \) be the array corresponding to the first half, and \( A_2 \) be the array corresponding to the second. We count the number \( c_1 \) of inversions in \( A_1 \) recursively, and then count the number \( c_2 \) of inversions in \( A_2 \) recursively. We ensure that (i) when the execution returns from \( A_1 \), the array \( A_1 \) has been sorted, and (ii) the same is true for \( A_2 \).

We now count the number \( c_3 \) of such inversions \( (e, e') \) that \( e \in A_1 \) and \( e' \in A_2 \). This can be achieved in \( O(n) \) time utilizing the fact that both \( A_1 \) and \( A_2 \) have been sorted. Initially, set \( i \) and \( j \) to 1, and \( c_3 \) to 0. Next, repeat the following until either \( i > |A_1| \) or \( j > |A_2| \):

- If \( A_1[i] < A_2[j] \), then increase \( c_3 \) by \( j - 1 \), and increase \( i \) by 1;
- Otherwise (i.e., \( A_1[i] > A_2[j] \)), increase \( j \) by 1.

If at this moment \( j = |A_2| + 1 \), increase \( c_3 \) by \( (|A_1| - i + 1)|A_2| \). The total number of inversions equals \( c_1 + c_2 + c_3 \).

Before returning to the upper level of recursion, we merge \( A_1 \) and \( A_2 \) into one sorted list \( A' \), and copy the elements of \( A' \) into \( A \) (which thus becomes sorted). This takes \( O(n) \) time.

Let \( f(n) \) be the worst-case running time of our algorithm. It holds that \( f(1) = O(1) \), and \( f(n) = 2 \cdot f(\lfloor n/2 \rfloor) + O(n) \). By the master theorem, we have \( f(n) = O(n \log n) \).

**Problem 5* (Maxima).** In two-dimensional space, a point \( (x, y) \) dominates another point \( (x', y') \) if \( x > x' \) and \( y > y' \). Let \( S \) be a set of \( n \) points in two-dimensional space, such that no two points share the same x- or y-coordinate. A point \( p \in S \) is a maximal point of \( S \) if no point in \( S \) dominates \( p \). For example, suppose that \( S = \{(1, 1), (5, 2), (3, 5)\} \); then \( S \) has two maximal points: \((5, 2)\) and \((3, 5)\).

Suppose that \( S \) is given in an array of length \( n \), where the \( i \)-th (1 ≤ \( i \) ≤ \( n \)) element stores the x- and y-coordinates of the \( i \)-th point in \( S \) (i.e., each element of the array occupies 2 memory cells). For example, \( S = \{(1, 1), (5, 2), (3, 5)\} \) is given as the sequence of integers: \((1, 1, 5, 2, 3, 5)\). Design an algorithm to find all the maximal points of \( S \) in \( O(n \log n) \) time.

**Solution.** First, sort all the points of \( S \) by x-coordinate in \( O(n \log n) \) time. Then, process the points in descending order of x-coordinate as follows. Initially, set \( y_{max} \) to \( \infty \). For each \( i \in [1, n] \), let \( p_i = (x_i, y_i) \) be the \( i \)-th point in the (descending) sorted order. If \( y_i < y_{max} \), ignore \( p_i \) and move on to the next \( i \). Otherwise, report \( p_i \) as a maximal point, and set \( y_{max} \) to \( y_i \). The processing obviously takes only \( O(n) \) time, rendering the overall time complexity \( O(n \log n) \).