Sample Solution to Problem Set 3

1. (5 + 5 = 10 points) Consider a set S of n distinct numbers given in unsorted order. Each of the following parts asks you to give an algorithm to determine two distinct numbers x and y that satisfy a stated condition. For each part, describe your algorithm and then briefly justify the correctness and the running time of the algorithm.

(a) In $O(n)$ time, determine $x, y \in S$ such that $|x - y|$ is maximum.

Answer: The two numbers that are farthest apart are the maximum and the minimum numbers in S. Therefore, find the maximum M and the minimum m of S. This takes $O(n)$ time. Return M and m.

(b) In $O(n \lg n)$ time, determine $x, y \in S$ such that $|x - y|$ is minimum.

Answer: The two numbers x and y such that $|x-y|$ is minimum must be consecutive numbers in the sorted order (otherwise, we can replace x by a number z that lies between x and y and thus obtain $|z-y| < |x-y|$, leading to a contradiction). So, we first sort the n numbers. This takes $O(n \lg n)$ time. Then, we scan through the sorted list, keeping track of the minimum difference between consecutive numbers, and return the two consecutive numbers that are closest. This scan takes $O(n)$ time. Therefore, total running time is $O(n \lg n)$.

2. (5 + 5 + 2 = 12 points) Stooge sort

Problem 7–3, pages 161-162.

Answer:

(a) Here is a formal proof of the correctness of STOOGE-SORT. (For getting credit, an informal but substantial argument would suffice.)

To prove that STOOGE-SORT(A, i, j) sorts correctly, we use induction on $\ell = j - i + 1$. (That is, ℓ is the number of elements in $A[i..j]$.) For the induction basis, we consider two cases: $\ell = 1$ or $\ell = 2$, i.e., $i = j$ or $i = j - 1$. For each of these cases, in Steps 1 and 2, $A[i]$ and $A[j]$ are exchanged if $A[i] > A[j]$. Also, in both cases the condition in Step 3 holds. Therefore, STOOGE-SORT(A, i, j) returns after having sorted A.

For the induction hypothesis, we assume that STOOGE-SORT sorts correctly for all arrays of length less than m, where $m > 2$. We now show that STOOGE-SORT sorts correctly for all inputs of length m. Since $m > 2$, the condition in Step 3 is not true; hence STOOGE-SORT does not return in Step 4. Moreover, for the discussion that follows, it does not matter whether the exchange of $A[i]$ and $A[j]$ took place in Step 2. So we now consider Steps 5 through 8. The variable k is set to $\lfloor (j - i + 1)/3 \rfloor$ in Step 5. Since $j - i + 1 \geq 3$, $k \geq 1$; therefore, $j - i - k + 1 = m - k < m$. Therefore, the length of the array $A[i..j - k]$ on which STOOGE-SORT is called in Step 6 is less than m. By the induction hypothesis, it follows that

$A[i..j-k]$ is in sorted order after Step 6. Hence, A has the following property: (P1) for all x in $[i..i+k-1]$ and y in $[i+k..j-k]$, $A[x] \leq A[y]$. In particular, we see that there are at least k elements in $A[i+k..j]$ that are greater than or equal to every element in $A[i..i+k-1]$. Next, STOOGE-SORT sorts $A[i+k,..,j]$. Again, the length of the array is less than m. So we invoke the induction hypothesis and claim that after Step 7, $A[i+k..j]$ is sorted. Since $A[i+k..j]$ is sorted, A has the following property: (P2) $A[j-k+1..j]$ is sorted correctly, and (P3) for all x in $[i+k..j-k]$ and y in $[j-k+1..j]$, $A[x] \leq A[y]$.

Since the elements in indices $[i..i+k-1]$ do not move, the following property (that follows from (P1)) still holds: (P4) there are at least k elements in $A[i+k..j]$ that are greater than or equal to every element in $A[i..i+k-1]$. Therefore, combining with property (P3) above, we have: (P5) for all x in $[i..j-k]$ and y in $[j-k+1..j]$, $A[x] \leq A[y]$. Thus, after step 6, properties (P2) and (P5) hold.

Finally, we apply STOOGE-SORT on $A[i..j-k]$. Again the length of the array, $j-k-i+1$ is less than m. So applying the induction hypothesis, we obtain that $A[i..j-k]$ is sorted correctly. Moreover, since the indices $[j-k+1..j]$ are unaffected, properties (P2) and (P5) still hold. We thus obtain that the entire array $A[i..j]$ is sorted correctly.

(b) Let $T(n)$ be the worts-case running time of STOOGE-SORT on an array of length n. Steps 1 through 5 take $\Theta(1)$ time. Steps 6 through 8 take $T(n-k)$ time. Since $k = \lfloor n/3 \rfloor$, $2n/3 \leq n-k \leq 2n/3+1$. Ignoring floors and ceilings, we can write the recurrence as

$$T(n) = 3T(2n/3) + \Theta(1).$$

We use the Master method. Since $b = 3, c = 3/2$, we get $E = \log_{3/2} 3$. We have $f(n) = \Theta(1)$; clearly, there exists an $\varepsilon > 0$ (for example, $\varepsilon = \log_{3/2} 3$) such that $f(n) = O(n^{E-\varepsilon})$. We thus invoke case 1 of the Master theorem to obtain $T(n) = \Theta(n^{\log_{3/2} 3}$.

(c) Since $T(n) = \Theta(n^{\log_{3/2} 3})$, and $\log_{3/2} 3 > 2$, STOOGE-SORT is worse than each of insertion sort, merge sort, heapsort, and quicksort. The professors do not deserve tenure.

3. (8 points) Generating a random uniform permutation

Exercise 5.3-4, page 105.

Answer: The probability that $A[i]$ winds up in position j of B is the probability that offset equals $j-i \bmod n$, if $j \neq i$, and n, if $j = i$. The probability that offset takes any particular value in the interval $[1 \ldots n]$ is $1/n$. Therefore, the probability that $A[i]$ winds up in $B[j]$ is $1/n$, for any given i and j.

Given an array $A[1..n]$, the given program only generates a permutation from the n cyclic permutations that do a "right circular shift" on A. To see this, we note that the only randomization that is happening is in the choice of offset. So we have only one of n permutations, the kth permutation mapping position i to $i+k$ (with appropriate wraparound), for all i. Each of these permutations has a probability $1/n$ of occuring. A uniformly random permutation generator, however, needs to generate all possible $n!$ permutations, each with probability $1/n!$.

4. (5 + 5 = 8 points) The effect of group size on SELECT

In the algorithm SELECT covered in class and described in Section 9.3 of the text, the input elements

are divided into groups of 5. Will the algorithm work in linear time if they are divided into groups of 3? How about groups of 7? Justify your answers.

(*Hint:* Obtain recurrence relations for both approaches, and solve them.)

Answer:

Groups of 3: SELECT is not linear in the worst-case, however, if we divide the input elements into groups of 3 instead. We get a good indication of that when we try to follow the same argument as we have done above. Since we can only claim that half of the $\lceil n/3 \rceil$ groups contribute 2 elements each, the best lower bound for any side of the partition that we are able to show is at most $2((1/2) \cdot (n/3))$, which is at most $n/3$. Thus the best upper bound for any side of the partition is at least $2n/3$. In fact, we can ensure this by choosing the input appropriately. Assume that n is a multiple of 6. We can select the input such that the number of groups with medians greater than the median-of-medians x is exactly $(1/2)(n/3) = n/6$. This implies that the number of elements greater than x is exactly $2n/6 + 1 = n/3 + 1$. Therefore, the number of elements less than x is exactly $2n/3 - 1$. We set the input rank such that it falls in the low side of the partition. If we do this for every level of the recursion, then this yields a recurrence of the form

$$T(n) \geq T(n/3) + T(2n/3 - 1) + \Theta(n),$$

which leads to $T(n) = \Omega(n \lg n)$. You can see this if you unravel the recursion tree obtained by the above recurrence and note that each of the $\Omega(\log n)$ levels of the recursion tree sum to $\Omega(n)$.

Groups of 7: If the input elements are divided into groups of 7, then we need to place new bounds on the size of low and high sides of the partition that are obtained in step 4 of SELECT on page 190 of the text. As before, at least half of the medians found in step 2 are greater than or equal to the median-of-medians x. Since there are $\approx n/7$ groups in our case, we find that at least half of the $n/7$ groups contribute 4 elements or more that are greater than x. Thus, the number of elements greater than x is at least $4n/14 = 2n/7$. Similarly, the number of elements that are less than x is at least (approximately) $2n/7$. It thus follows that the each of the two sides of the partition is of size at most $\approx 5n/7$. We thus get the following recurrence

$$T(n) \leq T(n/7) + T(5n/7) + O(n).$$

We can use the iteration method or the recursion tree approach to obtain that $T(n)$ in $O(n)$. The contributions at each level of the recursion tree form a geometric series with a ratio of $6/7$. Since $6/7 < 1$, this series converges, and we get $T(n) = O(n)$.

5. (10 points) Locating the Central Post Office in Manhattan

We are given the coordinates $(x_1, y_1), \ldots, (x_n, y_n)$, for each of n houses in Manhattan. The distance between any two houses (a, b) and (c, d) is given by $|a - c| + |b - d|$. This is referred to as the *Manhattan distance*, since the streets in Manhattan are organized in a grid.

We would like to determine the location for a central post office in the city such that the sum of the distances of the n houses from the central post office is minimized. Design and analyze an efficient algorithm for this problem. The more efficient your algorithm is in terms of its worst-case running time, the more points you will get.

Answer: We first show that an optimal location for the central post office is (x^*, y^*), where x^* is a median of $x_1, \ldots, x_n$ and y^* is a median of $y_1, \ldots, y_n$. Without loss of generality, we may renumber

the x_i's and y_j's in sorted order. Note that this is for our argument only. We do not actually sort the numbers. Let $m = \lceil n/2 \rceil$. Then, x_m (resp., y_m is a median of $x_1, \ldots, x_n$ (resp., $y_1, \ldots, y_n$). We now show that

$$\sum_{i=1}^{n} |x_i - x_m| \leq \sum_{i=1}^{n} |x_i - x|,$$

for any x. We first consider the case when $x < x_m$. In this case, we have

$$\begin{aligned} \sum_{i=1}^{n} (|x_i - x_m| - |x_i - x|) &\leq \sum_{i=1}^{m-1} |x_m - x| - \sum_{i=m}^{n} |x_m - x| \\ &\leq 0. \end{aligned}$$

Similarly, if $x > x_m$, we have

$$\begin{aligned} \sum_{i=1}^{n} (|x_i - x_m| - |x_i - x|) &\leq -\sum_{i=1}^{m} |x_m - x| + \sum_{i=m+1}^{n} |x_m - x| \\ &\leq 0. \end{aligned}$$

By the same argument, we have

$$\sum_{i=1}^{n} |y_i - y_m| \leq \sum_{i=1}^{n} |y_i - y|,$$

for any y. This shows that (x_m, y_m) is an optimal location for the post office.

We now analyze the running time. Both the coordinates x_m and y_m can be found using the linear time selection algorithms. Therefore, the optimal post office location can be found in linear time.

6. (10 bonus points) Selection from two sorted lists

Design and analyze an efficient algorithm to select the median from a set of $m + n$ keys given in the form of two sorted lists, one of length m and the other of length n. For convenience, you may assume that the $m + n$ keys are all distinct. (*Hint:* A running time of $O(\log(\min\{m, n\}))$ is achievable.)

Answer: We present an algorithm for selecting an element of arbitrary rank k from the two sorted lists. The median can then be computed by setting $k = \lfloor (m + n)/2 \rfloor$ or $k = \lceil (m + n)/2 \rceil$. Let the two sorted lists be $A[1..m]$ and $B[1..n]$. Without loss of generality, assume that $n \leq m$. The worst-case running time of our algorithm will turn out to be $O(\log n)$.

The basic idea is to reduce the size of array B by half in each iteration by performing only a consant number of operations. Thus, the number of iterations will be $O(\log n)$ and so will the worst-case running time be. In the following pseudocode, A and B are the two given arrays, p and r are two indices of array B, $p \leq r$, and k is the rank that we are seeking. The algorithm returns the element of rank k from the two sorted lists A and $B[p..r]$. In order to find the median, we simply call TwoLists$(A, B, 1, n, \lfloor (m + n)/2 \rfloor)$.

TwoLists(A, B, p, r, k)

1. **if** $k < r - p + 1$ then // No point in searching $B[p + k..r]$

2. then $r = p + k - 1$
3. **if** $p = r$ then // B has only one element
4. **if** $A[k-1] > B[p]$ then **return** $A[k-1]$
5. **else if** $A[k-1] < B[p]$ then **return** $\min\{A[k], B[p]\}$
6. **else** // B has more than one element
7. $\ell = \lfloor (r - p + 1)/2 \rfloor$ // ℓ is half the size of B
8. **if** $A[k - \ell] > B[p + \ell - 1]$ // compare the middle element of B with appropriate element of A
9. then **return** TWOLISTS$(A, B, p + \ell, r, k - \ell)$ // discard first half of B
10. **else return** TWOLISTS$(A, B, p, p + \ell - 1, k)$ // discard second half of B

The worst-case running time of the algorithm is given by the recurrence $T(n) = T(n/2) + \Theta(1)$, which yields $T(n) = \Theta(\log n)$ (like binary search).