Assignment 3: Sample solutions and comments

1. (a) In general the cost of a divide-and-conquer algorithm can be expressed as the sum of three terms: the *split cost* (the cost of splitting the problem into subproblems), the *recursive cost* (the cost of recursively solving the subproblems) and the *combine cost* (the cost of combining the solutions of the subproblems into a solution of the full problem). With this in mind, we can write the recurrence as $T(n) = S(n)+T(\lfloor n/2 \rfloor)+T(\lceil n/2 \rceil)+M(u_L, u_R)$, where $S(n)$ denotes the split cost and u_L (respectively, u_R) denotes the number of un-dominated points in the left (respectively, right) subproblem.

   Very frequently we will drop the floors and ceilings from recurrences like this, when it is clear that this will not change the asymptotic behaviour of the solution. This is essentially an observation that it suffices to solve the recurrence for input sizes of a special form (in this case, a power of two) since the cost of the algorithm increases monotonically with input size. Note that we would expect that $M(\cdot, \cdot)$ also grows monotonically, so for the purpose of upper bounding the cost it would suffice to replace u_L and u_R by $n/2$ as well. (Later we will see that it makes sense to analyse the cost of algorithms for problems like this in terms of *both* the input size and the output size.)

   Note that, assuming the inputs do not come pre-sorted by x-coordinate, we have $S(n) = \Theta(n)$, by using a deterministic linear-time median-finding algorithm.

   (b) Most people recognized that, since all of the points in the left subset S_L have smaller x-coordinates than all those in the right subset S_R, (i) the points of S_R that are un-dominated in S are precisely those that are un-dominated in S_R, and (ii) the points of S_L that are un-dominated in S are precisely those that are un-dominated in S_L and have y-coordinate greater that the largest y-coordinate among all points in S_R.

   This means that the list $output(S)$ representing the un-dominated points in S can be formed by appending the list $output(S_R)$ of un-dominated points of S_R to the list $output(S_L)$ of un-dominated points of S_L that has been truncated at last point whose y-coordinate exceeds the y-coordinate of the first point on $output(S_R)$. This can clearly be done in time proportional to the total length of $output(S_L)$, using standard list operations, which is certainly $O(u_L + u_R)$.

Expressed as a function of input size alone we have $T(n) \leq 2T(n/2) + O(n)$ which has the solution $T(n) = O(n \lg n)$.

(c) A simple (but by no means unique) reduction that achieves the desired result is to simply take the input $\langle x_1, x_2, \ldots, x_n \rangle$ that you wish to sort, and respond with the same permutation that corresponds to the *sequence* of un-dominated points associated with the input $\langle (x_1, -x_1), (x_2, -x_2), \ldots, (x_n, -x_n) \rangle$.

Although this implies that the un-dominated points problem is at least as hard as sorting, in some sense, one needs to be careful about concluding that $O(n \lg n)$ operations are required, since pairwise comparisons between inputs might not be the only operation that makes sense.

2. (a) We can imagine constructing S by a sequence of n uniform random draws from the interval $(0, 1]$. For any fixed $x \in (0, 1]$, we let X_i denote the indicator random variable with value 1, if the i-th draw is less than or equal to x, and 0 otherwise. Then the random variable $X = \sum_{i=1}^{n} X_i$ is just the number of keys in S that are less than or equal to x (i.e. the S-rank of x).

Since each key is chosen independently, each X_i is just a Bernoulli trial with success probability x, and X (the number of successes in n trials, is binomially distributed (see, CLRS, p. 1113- for details). Thus,

$$E[X] = E[\sum X_i] = \sum E[X_i] = nx,$$

(b) It follows immediately from part (a) that the expected number of elements in S that are less than or equal to q (which is the same as the expected rank of q in S) is nq. Thus, we expect to find the element of S that is closest in value to query q in location $\lfloor nq \rfloor$ or $\lceil nq \rceil$.

(c) Since the random variable X is binomially distributed, it has variance $np(1-p) \leq n/4$. It follows, by Chebyshev's inequality that, for any $s > 0$, $\Pr\{|l_q - e_q| \geq s\sqrt{n}\} \leq 1/(4s^2)$. (This was *given* in the assignment.) If we let $h_j = \Pr\{\lceil |l_q - e_q|/\sqrt{n} \rceil = j\}$ then it follows that

$$\sum_{j>i} h_j \leq 1/(4i^2).$$

Thus,

$$\begin{aligned} E[|l_q - e_q|/\sqrt{n}] &\le E[\lceil |l_q - e_q|/\sqrt{n} \rceil] \\ &= \sum_{j \ge 1} j h_j \\ &= \sum_{i \ge 1} \sum_{j \ge i} h_j \\ &= \sum_{i \ge 1} [h_i + \sum_{j > i} h_j] \\ &\le 1 + \sum_{i \ge 1} 1/(4i^2) < 1 + 1/2. \end{aligned}$$

(d) The pseudocode finds an interval of size $\sqrt{n}$ containing $D[l_q]$ by checking successive intervals of size $\sqrt{n}$, starting with an interval incident to $D[e_q]$, until the desired interval is found. It is clear that the total amount of work done is proportional to the number of iterations of the while loop which is just $\lceil |l_q - e_q|/\sqrt{n} \rceil$. By part (c), this has expected value less than 2.

(e) Since $T(n)$, the expected cost of finding the element closest to an arbitrary query q in an array of size n, grows monotonically with n, it follows that

$$T(n) \le T(2^{2^{\lceil \lg \lg n \rceil}}).$$

But, for some constant c, we have $T(2^{2^1}) = T(4) \le c$ and $T(2^{2^k}) \le c + T(2^{2^{k-1}})$, for $k > 1$. Thus, it follows, by induction on k, that $T(2^{2^k}) \le ck$, and hence $T(n) = O(1 + \lg \lg n)$.

3. (a) The while loop repeats at most n times: each iteration decreases $j - i$ by 1, and the loop terminates when $j - i$ (which is n-1 to start) becomes negative. Since the loop body involves $\Theta(1)$ work, the total cost is $\Theta(n)$.

(b) If the procedure returns TRUE it must be because both $A[i] + A[j] \le 0$ and $A[i] + A[j] \ge 0$; thus it is correct in this case. Suppose that $A[i_0] + A[j_0] = 0$, where $i_0 \le j_0$. We need to show that the procedure must return TRUE. Otherwise, since we have $i \le i_0$ and $j \ge j_0$ to start and $i > j$ to finish, at some point in the computation we must have $i = i_0$ and $j > j_0$, or $i < i_0$ and $j = j_0$. In the first case, $A[i] + A[j] > 0$, and so j will be decremented until $j = j_0$. In the second case, i will be incremented until $i = i_0$.

It is also possible to probe correctness, slightly more formally, by arguing that the following condition is an invariant of the while loop:

$$\begin{aligned} &\text{for all } p, q, \text{ where } 1 \le p < i \text{ and } j < q \le n \\ &\qquad A[p] + A[j] < 0 \text{ and } A[i] + A[q] > 0 \text{ and } A[p] + A[q] \ne 0 \end{aligned}$$

This holds trivially at the start (when $i = 1$ and $j = n$) and is easily seen to be be preserved by each loop iteration. If the loop is exited when $i = j + 1$ then the invariant is sufficient to establish that $A[p] + A[q] \neq 0$, for $1 \leq p \leq j$ and $j \leq q \leq n$.

(c) (i) replace "0" by "t". Nothing in the procedure depends on the target being 0.
(ii) replace "$i \leq j$" by "$i < j$". This eliminates the case where $i = j$.

(d) Assume A is sorted.
repeat, for each element $A[k]$ in A,
    use the generalized procedure from (c) to test if some pair
        of elements in A add up to the target value $-A[k]$.
    if so return TRUE
return FALSE

The cost is $O(n^2)$, since it involves n repetitions of the linear-time procedure in (c).