Homework 3
Com S 311

There are 3 problems and each problem is worth 40 points.

For problems 2 and 3, you must use the *divide and conquer* paradigm. Note that a typical design and conquer algorithm consists of 3 steps.

- Dividing the problem into sub problems
- Solving sub problems recursively.
- Combining solutions of sub problems.

Your algorithm must clearly and precisely describe each of these three steps. In addition, you must derive a recurrence relation for the runtime of the algorithm and solve the recurrence relation. (If you do not use divide and conquer, you will receive zero credit.)

1. Derive a solution to each of the following recurrence relations, using the technique of "unrolling" the recurrence as shown in sections 5.1 and 5.2 in the text[1] (in particular, see the subsections "Unrolling the Mergesort recurrence" on p. 212 and "The case of $q > 2$ subproblems" on p. 215). You must show your work, e.g. as on p. 216 of the text, but you are not required to provide a proof by induction of correctness (though you may wish to do so, in order to check your result).

   (a) $T(n) \leq 3T(n/2) + cn^2$, $T(2) \leq c$.
   Ans.

$$\begin{aligned} T(n) &\leq 3T(n/2) + cn^2 \\ &\leq 3[3T(n/4) + cn^2/4] + cn^2 \\ &= 3^2T(n/2^2) + \frac{3c}{4}n^2 + cn^2 \\ &\leq 3^2[3T(n/2^3) + cn^2/16] + \frac{3}{4}cn^2 + cn^2 \\ &= 3^3T(n/2^3) + \frac{3^2}{4^2}cn^2 + \frac{3}{4}cn^2 + cn^2 \end{aligned}$$

   In general,

$$T(n) \leq 3^kT(n/2^k) + [(\frac{3}{4})^{k-1}cn^2 + (\frac{3}{4})^{k-2}cn^2 + \cdots + (\frac{3}{4})^0cn^2]$$

[1] The required textbook for the course is *Algorithm Design* by Kleinberg and Tardos. There is a copy available on reserve at Parks library; ask at the circulation desk.

The recursion stops when $T(n/2^k) = 2$, thus when $k = \log_2 n - 1$. Substituting $k = \log n_2 - 1$ in the above equation, we obtain that

$$T(n) \leq 3^{\log_2 n-1}T(2) + [(\frac{3}{4})^{\log_2 n-2}cn^2 + (\frac{3}{4})^{\log_2 n-3}cn^2 + \cdots + (\frac{3}{4})^0 cn^2]$$

The term $3^{\log_2 n-1}T(2)$ is at most $c3^{\log_2 n-1}$ which is $O(n^{\log 3_2})$. The term

$$[(\frac{3}{4})^{\log_2 n-2}cn^2 + (\frac{3}{4})^{\log_2 n-3}cn^2 + \cdots + (\frac{3}{4})^0 cn^2]$$

is a geometric progression with a common ratio of $3/4$ and this sum is bounded by $O(n^2)$. Thus $T(n)$ is $O(n^2)$.

(b) $T(n) \leq 2T(n/2) + cn\log n$. $T(2) \leq c$.
Ans.

$$\begin{aligned} T(n) &\leq 2T(n/2) + cn\log n \\ &\leq 2[2T(n/2) + cn/2\log n/2] + cn\log n \\ &= 4T(n/4) + cn\log n/2 + cn\log n \\ &\leq 4[2T(n/8) + cn/4\log n/4] + cn\log n/2 + cn\log n \\ &= 8T(n/8) + cn\log n/4 + cn\log n/2 + cn\log n \end{aligned}$$

The kth term in the expansion is

$$2^k T(n/2^k) + cn\log n/2^{k-1} + cn\log n/2^{k-2} + \cdots + cn\log n$$

The recursion ends when $n/2^k$ equals 2, thus when $k = \log_2 n - 1$. Thus

$$T(n) \leq 2^{\log_2 n-1}T(2) + cn\log n/2^{\log_2 n-1} + cn\log n/2^{\log_2 n-2} + \cdots + cn\log n$$

$2^{\log_2 n-1}T(2)$ is $O(n)$.
Thus

$$T(n) = O(n) + cn[\log n + \log n/2 + \log n/4 + \cdots + \log n/2^{\log_2 n-1}$$

Consider

$$\log n + \log n/2 + \log n/4 + \cdots + \log n/2^{\log_2 n-1}$$

This has $\log n$ terms and this is at most

$$\log n + \log n + \log n + \cdots + \log n$$

which equals $\log^2 n$. Thus $T(n) = O(n\log^2 n)$.

2. An array $A = \{a_1, a_2, \cdots, a_n\}$ of integers is defined to be *k-sorted* if $A[i] \leq A[i+k]$ for every i in the range $[1, n-k]$. Give a divide and conquer algorithm that takes a k-sorted array of size n as input, and sorts it. Express the run-time as a function of n and k.

   Ans. For simplicity we assume that k is a power of 2 and n is even. Suppose A is k-sorted. Consider the following two arrays:

$$A[1], A[3], A[5], \cdots A[n-1]$$

$$A[2], A[4], A[6], \cdots A[n]$$

   Note that both of them are $k/2$ sorted arrays. Consider the following algorithm.

- SortKSortedArray(A, k)
- $B_1 = [A[1], A[3], \cdots A[n-1]]$.
- $B_2 = [A[2], A[4], A[6], \cdots A[n]]$
- If $k$is two, then B_1 and B_2 are sorted arrays. Use the merge algorithm to merge B_1 and B_2 into a single sorted array and return the resulting sorted array.
- $C = SortKSortedArray(B_1, k/2)$
- $D = SortKSortedArray(B_2, k/2)$
- Return $Merge(C, D)$.

Observe that B and C are arrays of size $n/2$ and they are $k/2$ sorted, Note that C and D are sorted arrays of size $n/2$, thus the time taken to merger them is $O(n)$. Thus we have the following recurrence

$$T(n,k) = 2T(n/2, k/2) + cn$$

$$\begin{aligned}
T(n,k) &= 2T(n/2, k/2) + cn \\
&= 2[2T(n/4, k/4) + cn/2] + cn \\
&= 4T(n/4, k/4) + 2cn \\
&= 4[2T(n/8, k/8) + cn/4] + 2cn \\
&= 8T(n/8, k/8) + 3cn \\
&= \cdot \\
&= \cdot \\
&= 2^{\ell}T(n/2^{\ell}, k/2^{\ell}) + \ell cn
\end{aligned}$$

The recursion ends when $k/2^{\ell}$ equals 2, thus when$\ell = \log_2 k - 1$.

Thus

$$\begin{aligned}
T(n,k) &= 2^{\log_2 k - 1}T(n/2^{\log_2 k - 1}, 2) + (\log_2 k - 1)cn \\
&\leq kT(2n/k, 2) + cn \log k
\end{aligned}$$

Note that $T(2n/k, 2)$ is $O(2n/k)$. Thus $T(n,k)$ is $O(2n + n \log k)$ which is $O(n \log k)$.

3. Let S be a set of two-dimensional points. Assume that all x-coordinates are distinct and all y-coordinates are distinct. A point $\langle x, y \rangle \in S$ is *purple* if there exists a point $\langle p, q \rangle$ in S such that $x < p$ and $y < q$. Give a divide and conquer algorithm that gets a set of points as input and outputs all purple points.

   Ans. We will sort S based on x co-ordinate. Let m be the median of the x co-ordinates. Let S_1 be the set of all points whose x co-ordinate is at most mid and S_2 be the set of all points whose x co-ordinate is more than mid. Solve the problem recursively. Let P_1 be the set of purple points of S_1 and P_2 be the set of all purple points of S_2. Note that every point in S_1 has its x co-ordinate smaller than every point in S_2. Fix a point $\langle a, b \rangle$ from S_1, It is possible that there is no point $\langle p, q \rangle$ in S_1 such that $x < p$ and $y < q$ thus $\langle a, b \rangle$ does not belong to P_1. However, if there is a point from S_2 whose y co-ordinate is bigger than b, then $\langle a, b \rangle$ becomes purple. We need to detect such points during the merge phase. Thus for every point $\langle a, b\rangle$ from S_1 we need to check if there is a point from S_2 whose y-coordinate is larger than b. Let Y_2 be the sorted list of points from S_2 based on y co-ordinate. Given $\langle a, b\rangle$ we can detect if there is a point in Y_2 whose y co-ordinate is larger than b in $O(\log n)$ time. Using these ideas, we arrive at the following algorithm.

- PurplePoints(S).
- Sort S based on x co-ordinate. Let z be the median of x co-ordinates.,
- If S has only two points, then determine purple points by comparing the x and y coordinates of both the points and return.
- S_1 all points whose x co-ordinate is at most z,
- S_2 all points whose x co-ordinate is larger than z.
- $L_1 = PurplePoints(S_1)$
- $L_2 = PurplePoints(S_2)$.
- Y_2 = sorted list of points from S_2 based-on y co-ordinate.
- $L_3 = emptylist$.
- For every $\langle a, b\rangle \in S_1$ perform a binary search and determine if there is a point $\langle p, q\rangle$ in Y_2 such that $q > b$. If so, add $\langle a, b\rangle$ to L_3.
- Return the union of L_1, L_2 and L_3.

Note that sorting S takes $O(n \log n)$ time. Sorting S_2 takes $O(n \log n)$ time. Time taken by the binary search step takes $O(n \log n)$ time. Thus the recurrence is

$$T(n) = 2T(n/2) + cn \log n$$

whose solution is $O(n \log^2 n)$.

Remark. The run time can be reduced to $O(n \log n)$ as in closest-pair of points problem. Instead of sorting during recursive steps, we maintain two sorted lists of points X sorted based on x-coordinate and Y based on y-co-ordinate. Now the merge step is the following:

Y_1 sorted list of points from S_1 based on y-co-ordinate. Y_2 Sorted list of points from S_2 based on y-co-ordinate. For every $\langle a, b\rangle \in S_1$ perform a binary search and determine if there is a point $\langle p, q\rangle$ in Y_2 such that $q > b$. This can be done is $O(n)$ time using an algorithm that is similar to merge. Details are not provided, please think about this.