ESO207A: Data Structures and Algorithms

Homework *1a: Divide and Conquer* HW Due Date: Aug 21, 2018

Review and some definitions. Assume f, g to be asymptotically non-negative functions. In brief, $f(n) = O(g(n))$ if asymptotically, $f(n) \le c \cdot g(n)$, for some constant c. $f(n) = \Theta(g(n))$ if asymptotically, $c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n)$, for some constants c_1, c_2; that is, up to constant factors, $f(n)$ and $g(n)$ are asymptotically similar. $f(n) = \Omega(g(n))$ if asymptotically, $f(n) \ge c \cdot g(n)$, for some constant c. (Informally, asymptotically and up to constant factors, f is at least g). $f(n) = o(g(n))$ if $\lim_{n\to\infty} \dfrac{f(n)}{g(n)} = 0$. That is, for every $c > 0$, asymptotically, $f(n) < c \cdot g(n)$, (i.e., f is an order lower than g).

Problem 1. Order Notation and Recurrence Equations (30)

(a) State whether each of the following statement is true or false, with a brief reason. (i) $n^2 = O(2^n)$, (ii) $n^2 = \Omega(n^3)$, (iii) $2^{2n} = \Theta(2^{n+c})$, for any constant c, (iv) $n^2 = o(n^2 \log n)$, (v) $\log_2 n = \Theta(\log_c n)$, for any fixed constant c. (10)

(i) True. Asymptotically with n, $n^2 \ll 2^n$ and so $n^2 = O(2^n)$; and $n^2 = o(2^n)$. (ii)False. In fact the opposite is true, $n^2 = o(n^3)$. (iii) False. $2^{2n} = (2^{n+c})^{2n/(n+c)}$. $2^{2n} = \Omega(2^{n+c})$ and also $2^{2n} = \omega(2^{n+c})$. (iv) True. $\frac{n^2}{n^2 \log n} = \frac{1}{\log n} \to 0$, as $n \to \infty$. (v) True. $\log_c n = \frac{\log_2 n}{\log_2 c} = \Theta(\log_2 n)$, since, c is a constant.

(b) Solve the following recurrence equations. Assume that $T(1) = \Theta(1)$ and $T(2) = \Theta(1)$ for all the functions below. Make and state appropriate simplifying assumptions if needed. $(4 \times 5 = 20)$.

1. $T(n) = T(2n/3) + n$.
   For simplicity, let n be a power of 3.

$$\begin{aligned} T(n) = n + T(2n/3) &= n + (2n/3) + T((2/3)^2 n) \\ &= n + (2n/3) + (2/3)^2 n + \ldots + T((2/3)^k n) \\ &= \Theta(n) + \Theta(1) \\ &= \Theta(n) \end{aligned}$$

   The sum $n + (2/3)n + \ldots$ is a geometric progression and its sum is upper bounded by $3n$ and lower bounded by n. Hence, the sum is $\Theta(n)$.

2. $T(n) = 4T(n/3) + n^2$.
   For simplicity, let n be a power of 3.

$$\begin{aligned} T(n) &= n^2 + 4T(n/3) \\ &= n^2 + 4(n/3)^2 + 4^2 T(n/3^2) \\ &= n^2 + 4(n/3)^2 + 4^2(n/3^2)^2 + \ldots + 4^k(n/3^k)^2 + \ldots + 4^{\log_3 n} T(1) \ . \end{aligned}$$

   The summation $\sum_{j=0}^{\log_3 n} 4^j (n/3^j)^2$ is a geometric series with common ratio $4/3^2 < 1$ and hence the sum is $\Theta($ first term $)$, which is $\Theta(n^2)$.

3. $T(n) = 3T(n/3) + n$.
   Let n be a power of 3.

$$\begin{aligned} T(n) &= n + 3T(n/3) \\ &= n + 3(n/3) + 3^2T(n/3^2) \\ &= n + 3(n/3) + 3^2(n/3^2) + \ldots + 3^k(n/3^k) + \ldots + 3^{\log_3 n}T(1) \\ &= \Theta(n \log n) \ . \end{aligned}$$

4. $T(n) = 2T(n/2) + n \log n$. For this problem argue an upper bound that is $o(n^2)$.
   Let n be a power of 2.

$$\begin{aligned} T(n) &= n \log n + 2T(n/2) \\ &= n \log(n) + 2(n/2)\log(n/2) + 2^2T(n/2^2) \\ &= n \log(n) + 2(n/2)\log(n/2) + \ldots + 2^k(n/2^k)\log(n/2^k) + \ldots + 2^{\log_2 n}T(1) \\ &\leq n \log(n) + 2(n/2)\log(n) + \ldots + 2^k(n/2^k)\log(n) + \ldots + 2^{\log_2 n}\log(n)T(1) \\ &= \log(n)\left(n + 2(n/2) + 2^2(n/2^2) + \ldots + 2^{\log_2 n}T(1)\right) \\ &= \log(n)O(n \log n) \\ &= O(n \log^2(n)) \ . \end{aligned}$$

Problem 2. Non-dominated points (35)

A two-dimensional point (x, y) is said to dominate another two-dimensional point (u, v) if $x \geq u$ and $y \geq v$. Given a set P of points, a point $p = (x, y)$ is said to be a non-dominated point of P (also called a maximal point) if no other point q in P dominates p. See Figure 1 for examples. Given a set of n points P placed in arbitrary order in an array, give a time efficient algorithm FIND_NON_DOM(P, n) to find the set of all non-dominated points in P. *Notes*:

1. For simplicity, assume that no two points have the same x-coordinate or the same y-coordinate.
2. A point p is represented as a structure with two attributes $p.x$ and $p.y$. The set of points P, is represented as an array $P[1, \ldots, n]$ of points in arbitrary order.
3. You can find the index of the median of the points in $P[k, \ldots, l]$ by the x coordinate by using an informal statement like "let $i =$ MEDIAN(P, k, l) by x-coordinate" Similarly, a statement like "let $i =$MEDIAN(P, k, l) by y coordinate" can be used to find the index of the median of the points in $P[k, \ldots, l]$ by the y coordinates. These functions run in time $O(k - l + 1)$. The median of n points is returned as the $\lfloor (n + 1)/2 \rfloor$ th ranked item.
4. Full marks will be provided for a correct solution that takes $O(n \log n)$ time.
5. A correct solution of $O(n^2)$ time will earn only 10 points in total.

Solution. The idea behind a divide and conquer solution is illustrated in Figure 2.

The top-level call is FIND_NON_DOM$(P, 1, n)$. Before calling this routine, assume that P is sorted in increasing order according to $P.x$. This advance sorting of P makes calls to median unnecessary. So point 3 of the problem may be ignored.

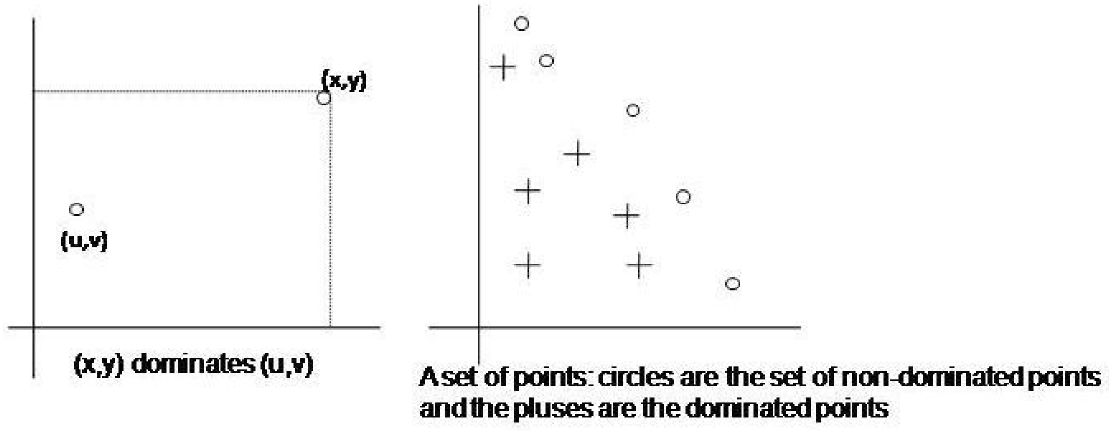
(x,y)
(u,v)
(x,y) dominates (u,v)
A set of points: circles are the set of non-dominated points
and the pluses are the dominated points

Figure 1: Dominated and non-dominated points in a point-set

The call to FIND_NON_DOM(P, k, l) returns a pair (N, r), where, N is an array of r points which are all (and only) the non-dominating points in the set $P[k, \ldots, l]$ *and are sorted* in increasing order of x-coordinate. Since N is a non-dominating set and it is sorted by x-coordinate in increasing order, it is automatically sorted in decreasing order of y-coordinate.

The pre-sorting of the point set P ensures that as points are dropped from P, the points that remain are sorted in order of x coordinate.

```
FIND_NON_DOM$(P, k, l)$ // Assume $P[1 \ldots n]$ is pre-sorted by $x$ coordinate.
1.  if  $l == k$
2.      Create a new array $N$ of size 1
3.      $N[1] = P[k]$
4.      return $(N, 1)$
5.  let $m = \lfloor (k + l)/2$ // due to pre-sorting
6.  $(N_1, r)$ = FIND_NON_DOM$(P, k, m)$
7.  $(N_2, u)$ = FIND_NON_DOM $(P, m + 1, l)$
8.  return  FIND_NON_DOM_MERGE$(N_1, r, N_2, u)$
```

The algorithm uses the procedure FIND_NON_DOM_MERGE(N_1, r, N_2, s) for merging two sets of non-dominating points, N_1 with r points and N_2 with s points. This merge routine assumes the following.

1. N_1 and N_2 are both sorted by x-coordinate. N_1 is the non-dominated point set of the first half of points $P[k \ldots m]$ and N_2 is the non-dominated set for the second half $P[m + 1 \ldots k]$.

2. $N_2.x \geq N_1.x$ that is, for every $p \in N_1$ and every $q \in N_2$, $q.x \geq p.x$.

Refer to Figure 2. All points in N_1 that have y value lower than or equal to the highest y-point in N_2 (which is the first point in the sorted order by x) are dominated by the highest y point of N_2. Conversely, each point in N_1 with y value larger than the highest y value point of N_2 is not

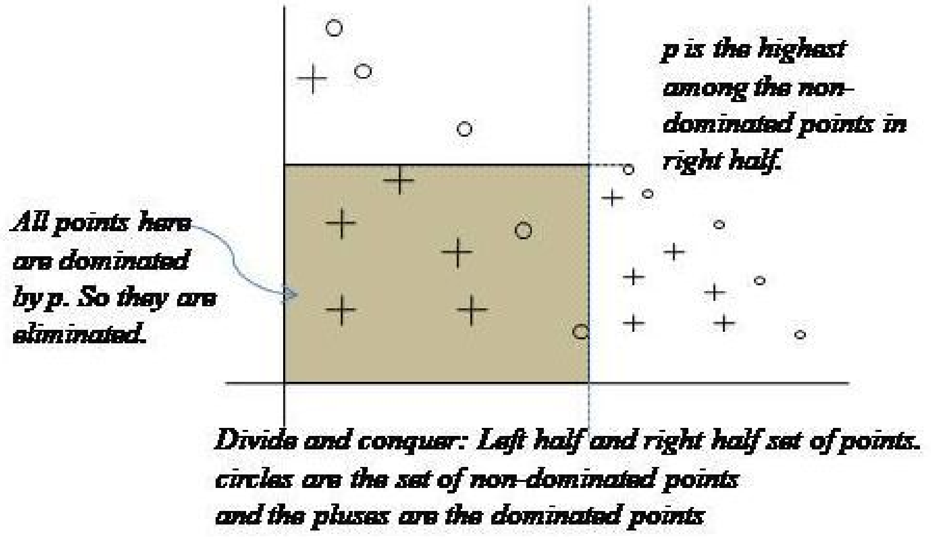
p is the highest among the non-dominated points in right half.
All points here are dominated by p. So they are eliminated.
Divide and conquer: Left half and right half set of points.
circles are the set of non-dominated points
and the pluses are the dominated points

Figure 2: The set of points is divided into two almost equal parts by the median according to x coordinate. The set of non-dominated points in each set are computed. All the circled points, i.e., non-dominated points of the left half, lying in the shaded region on or below $p.y$ line, are eliminated. The non-dominating set consists of all points from the left-half non-dominating set above the shaded region and all the points in the non-dominating set of the second half.

dominated by it, and therefore cannot be dominated by any other point of N_2, since they all have smaller y values. On the other hand, no point in N_2 can be dominated by a point in N_1 since their x values are larger.

```
FIND_NON_DOM_MERGE($N_1, r, N_2, s$)
1.  // first find the points $p$ in $N_1$ with higher $y$-coordinate than the first point of $N_2$.
2.      $t = 0$
3.      while  $t < r$ and $N_1[t+1].y > N_2[1].y$
4.          $t = t + 1$
5.      Create a new array $N[1, \dots, t+s]$.
6.      copy first $t$ elements of $N_1$ into the respective first $t$ positions of $N$.
7.      Subsequently, copy all $s$ elements of $N_2$ into positions $t+1, \dots t+s$ of $N$.
8.  return  $(N, s+t)$
```

The time complexity of FIND_NON_DOM_MERGE (N_1, r, N_2, s) is $O(r + s)$. Going back to the function FIND_NON_DOM(P, k, l), in lines 5 and 6, the recursive calls FIND_NON_DOM(P, k, m) and FIND_NON_DOM$(P, m+1, l)$ takes time $T(\lceil (l-k+1)/2 \rceil)$ and $T(\lfloor (l-k+1)/2 \rfloor)$ by recursion. The non-dominating set from the first recursive call is N_1 with r elements and so $r \leq \lceil (l-k+1)/2 \rceil$. Similarly, the non-dominating set from the second recursive call is N_2 with s elements, where, $s \leq \lfloor (l-k+1)/2 \rfloor$. Thus, $r + s \leq (k - l + 1)$.

The time taken by FIND_NON_DOM_MERGE (N_1, r, N_2, u) is $O(r + s)$, which is $O(l - k + 1)$. Thus, letting $l - k + 1 = n$, we get the recurrence relation,

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n)$$

whose solution is $T(n) = O(n \log n)$.

Alternative Solution. This solution does not use the divide and conquer method. Let $P[1, \ldots, n]$ be sorted in increasing order by x coordinates, with ties broken by y coordinate in increasing order. Now scan P backwards from the point with highest x. Let $y_m = P[n].x$. Maintain the loop invariant that any point p in the backwards scan of P must have $p.y > y_m$ to be included in the non-dominating set. The initialization is obviously true.

To see the maintenance, suppose the invariant is true for n downwards to $k \geq 2$. Now consider $k-1$. Then, $y_m = \max_{t=k}^{n} P[t].y$. If $P[k-1].y \leq y_m$, then, $P[k-1]$ is dominated by at least one point in $P[k \ldots n]$ and we discard $P[k-1]$. Otherwise, $P[k-1]$ is included into N, which is filled in the backwards order (increasing order of y, and decreasing order of x).

```
FIND_NON_DOM(P, n)      //assumes $n \geq 1$
1.  sort $P[1, \ldots, n]$ in increasing order of $x$, ties broken in increasing order of $y$.
2.  Create a new array $N[1, \ldots, n]$
3.  $y_m = -\infty$
4.  $t = 0$
5.  for  $k = n$ downto 1
6.      if  $P[k].y > y_m$
7.          $t = t + 1$
8.          $N[t] = P[k]$
9.          $y_m = P[k].y$
```

The time required is dominated by the sorting in line 1, which requires $\Theta(n \log n)$ time.

Problem 3. (35)

An important court trial is going on that has N witnesses. However, not all witnesses are honest; a witness is either *true* or *false*. A *true* witness always speaks the truth, whereas a *false* witness may lie and cannot be relied upon. To test the witnesses' credibility, the judge speaks to them in pairs (e.g., $W1$ and $W2$). Both are asked the same questions about the case and the answers given by each one is presented to the other. Each of them (e.g., Wi) is now asked whether the other person (i.e. $W2$) is telling the truth or not (and vice-versa). The possibilities are as follows:

	W1 says	W2 says	Conclusion
Case 1.	W2 is true	W1 is true	Both are true, or both are false witnesses
Case 2.	W2 is true	W1 is false	At least one is a false witness
Case 3.	W2 is false	W1 is true	At least one is a false witness
Case 4.	W2 is false	W1 is a false	At least one is a false witness

a. Show that if there are more than $N/2$ false witnesses, the judge cannot necessarily pick out the true witnesses, irrespective of the pairwise test strategy used. Assume that the false witnesses can conspire to fool the judge.

Suppose the every bad witness does the following. Given a good witness's account, it says it is false, and given a bad witness's account, it says it is true. Now the good witnesses will report truthfully, and so it is impossible for the judge to distinguish between the set of good witnesses and the set of bad witnesses (who are consistent by conspiracy).

b. Assuming that more than $N/2$ witnesses are true, the judge now wants to identify one true witness. Show that $\lfloor N/2 \rfloor$ pairwise tests are sufficient to reduce this problem to that of approximately half its size.

Solution. Arbitrarily number the witnesses from 1 to N. Consider the following algorithm.

```
procedure ReduceWitnesses($M, N$)
// $M$ is a list of $N$ witnesses (array)
1.  Let $L = \phi$.
2.  Arbitrarily pair the witnesses. // if $N$ is odd, there may be one unpaired witness
3.  for each pair that is of type case 1
4.          choose an arbitrary witness from the pair
5.          add it to $L$
6.  if $L$ has even length and there is an unpaired witness $x$ // for odd $N$
7.      add it to $L$
8.  return $(L, |L|)$
```

The number of tests is $\lfloor N/2 \rfloor$, as we are pairing witness 1 with 2, 3 with 4, ..., $2i-1$ with $2i$, for $i = 1, 2, \ldots, \lfloor N/2 \rfloor$. If N is odd, then the final Nth witness is unpaired.

Pairs belonging to cases 2 and 3 have one good and one bad witness pairing. Since the number of good witnesses are overall in majority, removing all such (one good, one bad) pairings still leaves the good witnesses in majority. Pairs belonging to case 4 are both bad, and so removing such pairs still leaves the number of good witnesses in majority (only increases the fraction of good witnesses holding the number constant). Finally, each pair belonging to case 1 is either both good or both bad, so choosing one witness from one pair means, for a good pair, we choose a good witness, and from a bad pair, we choose a bad witness.

Assume N is even. After removing all case 2, case 3 and case 4 pairs, the good witnesses are in majority. Then, choosing one witness from each case 1 pair, reduces the problem size to $N/2$ or smaller and the number of good witnesses remains in majority.

Assume N is odd. Remove all case 2,3 and 4 pairs. There would be an unpaired witness. Suppose L is empty, that is there are no case 1 pairs. Then the unpaired witness is good since good witnesses are in majority. If L has odd length, then the number of good pairs are more than the number of bad pairs, so good items are in majority in L.

Now suppose L has even length. Let x be the unpaired witness. In the worst case, L may have equal number of good and bad witnesses, in which x must be good and we can add x to L. Otherwise, L has two more good witnesses than bad, in which case, even if x is bad, it can be added to L and still the number of good witnesses are in majority.

c. Show that the true witnesses can be identified with $\Theta(N)$ pairwise tests, assuming that more than $N/2$ witnesses are true. Argue and solve the recurrence that describes the number of tests.

Recursively call the procedure ReduceWitnesses. Initally, when called on witness array of size N with majority of good witnesses, it gives a witness array of size $\lceil N/2 \rceil$ with the property that there is a majority of good witnesses. We can call this recursively, until the size reduces to 1, in which case we have a good witness.

```
procedure FindOneGoodWitness$(M, N)$
// $M$ is a list of $N$ witnesses (array)
1.    if $N == 1$ return $M$     // $M$ has only one good witness
2.    $(L, l) = ReduceWitnesses(M, N)$
3.    return FindOneGoodWitness $(L, l)$
```

The number of witness pairings in each step of *ReduceWitness*(M, N) is at most $N/2$. Let $T(N)$ be the number of witness pairings in the procedure *FindOneGoodWitness*(M, N). Then, $T(N) \leq T(N/2) + O(N)$. The solution of this recurrence equation is $T(N) = O(N)$.