CS 341 Midterm

2019-02-26

Instructions

- NO CALCULATORS OR OTHER AIDS ARE ALLOWED.
- You should have 19 pages (including the header and extra pages).
- Make sure you fill the information on the header page.
- Solutions will be marked for clarity, conciseness and correctness.
- If you need more space to complete an answer, you may continue on the two blank extra pages at the end.

Useful Facts and Formulas

1. Master Theorem
   Suppose that $a \geq 1$ and $b > 1$, $d \geq 0$. Consider the recurrence
   $$T(n) = a\,T\left(\frac{n}{b}\right) + \Theta(n^d)$$
   Then:
   $$T(n) \in \begin{cases} \Theta(n^{\log_b(a)}) & \text{if } a > b^d \\ \Theta(n^d \log n) & \text{if } a = b^d \\ \Theta(n^d) & \text{if } a < b^d. \end{cases}$$
2. $a^{\log_b c} = c^{\log_b a}$
3. $\frac{d}{dx} \log_c x = \frac{1}{x \ln c}$ for $c > 1$
4. $\sum_{i=1}^{n} \log(i) = \log n! \in \Theta(n \log(n))$

1. (28 marks) *Short Questions* For each question below, give your answer together with a brief explanation. Show computations if it is appropriate to do so.

(i) **(12 marks)** For each pair of functions $f(n)$ and $g(n)$, fill in the correct asymptotic notation among Θ o, ω in the statement $f(n) \in \sqcup\, g(n)$. Briefly justify your answers.

$f(n) = 2^n$ vs. $g(n) = \log 2^{(4^n)}$

Solution: Observe $g(n) = \log 2^{(4^n)} = 4^n \log 2 \in \Theta(4^n)$. Moreover, $f(n) = 2^n \in o(4^n)$. Therefore, $f(n) \in o(g(n))$.

$f(n) = 2015n^2 + n^3$ vs. $g(n) = 2016n^{\sqrt{5}} + n$

Solution: Observe $f(n) = 2015n^2 + n^3 \in \Theta(n^3) = \Theta(n^{\sqrt{9}})$. Moreover, $g(n) = 2016n^{\sqrt{5}} + n \in \Theta(n^{\sqrt{5}})$. Therefore, $f(n) \in \omega(g(n))$.

$f(n) = \sum_{i=0}^{\log_3 n} 3^i$ vs. $g(n) = 0.5n$

Solution: Note that $f(n)$ is a geometric summation

$$\sum_{i=0}^{x} r^i = \frac{r^{x+1} - 1}{r - 1}$$

where $r = 3$ and $x = \log_3 n$. Thus, we have the following.

$$f(n) = (3^{1+\log_3 n} - 1)/2 = (3 \cdot 3^{\log_3 n} - 1)/2 = (3n - 1)/2 \in \Theta(n)$$

Furthermore, $g(n) \in \Theta(n)$. So, $f(n) \in \Theta(g(n))$.

$f(n) = 4^{(n^2)}$ vs. $g(n) = 100^n$

Solution: First proof: $lim_{n\to\infty} \frac{4^{n^2}}{100^n} = lim_{n\to\infty} \frac{4^{n^2}}{4^{nlog_4 100}} = \infty$

Second proof: The n^2 term in the exponent will *dominate* the constant difference in the bases. Since $n^2 \in \omega(n)$, we have $f(n) \in \omega(g(n))$. We show for all $c > 0$, there exists $n_0 > 0$ such that $f(n) > cg(n)$ for all $n > n_0$ (the definition of ω). Fix arbitrary $c > 0$. Observe $4^{n^2} > 100^n$ iff $\log 4^{n^2} > \log(100^n c)$ iff $n^2 \log 4 > \log 100^n + \log c$ iff $n^2 \log 4 - n \log 100 - \log c > 0$. The left side is a quadratic equation that is positive for all $n > n_0$, where n_0 is the larger of its two roots.

(ii) *Pseudocode analysis* **(16 marks)**

Analyze the complexity of the following pseudocode, by filling in the table next to the pseudocode. Fill in each entry with a Θ-bound in simplified form. Provide short explanations for anything that is not obvious on the next page.

```
1.   S = 0
2.   i = 1
3.   while i < 2n + 1 do
4.       for j = 1 to i do
5.           S = S + j
6.       i = i + 2
7.   end while
8.   i = n
9.   while i > 1 do
10.      j = i
11.      while j > 1 do
12.          S = S + j
13.          j = ⌊ j /2 ⌋
14.      end while
15.      i = i - 1
16.  end while
17.  return(S)
```

lines	complexity
1–2	
4–5	
3–7	
8	
11–14	
9–16	
17	
1–17	

Solution:

lines	complexity
1–2	$\Theta(1)$
4–5	$\Theta(i)$
3–7	$\Theta(n^2)$
8	$\Theta(1)$
11–14	$\Theta(\log i)$
9–16	$\Theta(n \log n)$
17	$\Theta(1)$
1–17	$\Theta(n^2)$

For lines 3–7, note that

$$1 + 3 + \cdots + 2n - 1 = n^2.$$

For lines 11–14, j takes on the values $i, i/2, i/4, \ldots, 1$, so the number of iterations is $\Theta(\log i)$.

For lines 9–16, we have

$$\sum_{i=1}^{n} \Theta(\log i) = \Theta(n \log n)$$

by the formula on page 2.

For line 17, we compute $\max\{\Theta(1), \Theta(n \log n), \Theta(n^2)\} = \Theta(n^2)$.

(Extra space.)

2. (16 marks) *Recurrences.* Consider the recurrence:

$$T(n) = 2T(\lfloor n/9 \rfloor) + \sqrt{n} \qquad \text{if } n \geq 9$$

$$T(n) = 5 \qquad \text{if } n < 9$$

Prove $T(n) = O(\sqrt{n})$ by induction (i.e., guess-and-check or substitution method). Show what your c and n_0 are in your big-oh bound.

Solution: Claim: $T(n) \leq c\sqrt{n}$ for all $n \geq n_0$ c, n_0 will be determined inside the proof. Base Case: For T(1), T(2), ..., T(8) = $5 \leq c\sqrt{n}$ implies as long as $c \geq 5$ all the base cases will hold (and we can set $n_0 \geq 1$).

Inductive Hypothesis:

Assume $\forall k \leq n-1$, $T(k) \leq c\sqrt{k}$.

Prove $T(n) \leq c\sqrt{n}$.

$$T(n) \leq 2c\sqrt{n/9} + \sqrt{n} = (\frac{2c}{3} + 1)\sqrt{n} \leq c\sqrt{n}$$

$$c \geq 3$$

.

So if we let $c \geq \max\{3, 5\} = 5$ and $n_0 = 1$, the inductive proof follows.

3. (18 marks) *Divide and Conquer 1.* Consider a polynomial $P(x)$ of degree d, $P(x) = \sum_{i=0}^{d} p_i x^i$, where you can assume each p_i is a positive integer. You can assume that $P(x)$ is represented only by its coefficients given in an array $p_0, \ldots, p_d$. Given two polynomials $P(x)$ and $Q(x)$ of degree $n-1$, their product $R(x)$ is given as follows:

$$R(x) = (\sum_{i=0}^{n-1} p_i x^i)(\sum_{i=0}^{n-1} q_i x^i) = \sum_{i=0}^{2n-2} \Big(\sum_{j=\max(0,i-n+1)}^{i} p_j q_{i-j} \Big) x^i$$

For example if $P(x) = 2x^2 + 1$ and $Q(x) = x^2 + x + 1$, $R(x) = 2x^4 + 2x^3 + 3x^2 + x + 1$.

(a) (6 marks) Let $P(x) = 2x^3 + x^2 + 2$ and $Q(x) = x^3 + x$. What is $R(x) = P(x)Q(x)$? You can fill in the coefficients below.

$R(x) = \underline{\hspace{2em}}x^6 + \underline{\hspace{2em}}x^5 + \underline{\hspace{2em}}x^4 + \underline{\hspace{2em}}x^3 + \underline{\hspace{2em}}x^2 + \underline{\hspace{2em}}x^1 + \underline{\hspace{2em}}x^0$

Solution:

$$R(x) = 2x^6 + 1x^5 + 2x^4 + 3x^3 + 0x^2 + 2x^1 + 0x^0$$

(b) (12 marks) Given two polynomials $P(x)$ and $Q(x)$ of degree $n-1$, design a divide and conquer algorithm that is asymptotically faster than $O(n^2)$ to compute $R(x) = P(x)Q(x)$. You can assume that $P(x)$ and $Q(x)$ are given as two integer arrays of length n, where e.g. $P[i]$ is p_i, i.e., the ith coefficient of $P(x)$. The output of your algorithm should the the $2n-2$ coefficients of $R(x)$ given in an array. You can assume that multiplying two numbers is an $O(1)$ time operation. Give either the pseudocode or a high-level detailed description of your algorithm and briefly justify the correctness of your algorithm. Write down and solve the recurrence of the runtime of your algorithm.

Hint: Think of an approach we took for another problem in class.

Solution: The solution to this is exactly similar to the KO integer multiplication. We write $P(x)$ and $Q(x) as:$

$$P(x) = P_A(x)x^{n/2} + P_B(x), Q(x) = Q_C(x)x^{n/2} + Q_D(x)$$

. Here both P_A,P_B, Q_C, and Q_D are all polynomials of degree $n/2-1$. Then:

$$R(x) = P_A(x)Q_C(x)x^n + (P_A(x)Q_D(x) + P_B(x)Q_C(x))x^{n/2} + P_B(x)Q_D(x)$$

We can recover $P_A(x)Q_D(x) + P_B(x)Q_C(x)$ by:

$$(P_A(x) + P_B(x))(Q_C(x) + Q_D(x)) - (P_A(x)Q_C(x)) - P_B(x)Q_D(x))$$

So we can recursively compute: (1) $P_A(x)Q_A(x)$, (2) $(P_A(x)+P_B(x))(Q_C(x)+Q_D(x))$, and (3) $P_B(x)Q_D(x)$:

DCPM(P, Q: $n-1$ degree polynomials):

1. if $n \leq 1$: return $P[0]Q[0]$;
2. P$_AQ_A = DCPM(P[0,...,n/2-1], Q[0,...,n/2-1]$;
3. Tmp1 = P[0,...,n/2-1] + P[n/2,...,n-1];
4. Tmp2 = Q[0,...,n/2-1] + Q[n/2,...,n-1];
5. Tmp = DCPM(Tmp1, Tmp2);
6. P$_BQ_D = DCPM(P[n/2,...,n-1], Q[n/2,...,n-1]);$
7. Return P$_AQ_Ax^n + (Tmp - P_AQ_A - P_BQ_D)x^{n/2} + P_BQ_D$

Above, x^i is an "array shift" operation by n cells and + is a vector summation (so sums each cell of the array cell by cell), which can be done in $O(n)$ time.

Runtime: $T(n) = 3T(n/2) + O(n) = O(n^{\log_2(3)})$.

(Extra space)

4. (20 marks) Divide and Conquer 2: The L1-distance between two points (x, y) and (x', y') is defined as $|x - x'| + |y - y'|$. Given a set P of points in two dimensions, where each point is colored either "red" or "blue", we would like to compute the L1-closest red-blue pair, i.e., a pair of points p and q, where p is red and q is blue, such that the L1-distance between p and q is minimum across all such red-blue pairs. You may assume that coordinates are all distinct. There are three parts to this problem; see the following pages.

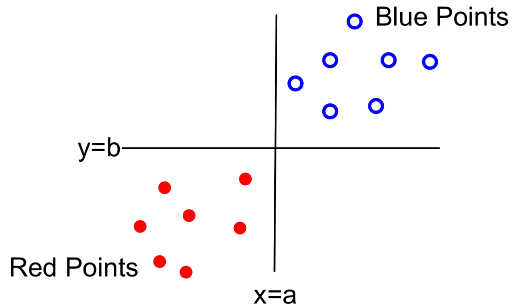

Figure 0.1: Sample input for part(a). Red points have their insides filled and blue points are drawn with empty insides.

(a) (5 marks) Consider the special case where all red points are in the region $\{(x, y)|x \leq a, y \leq b\}$ and all blue points are in the region $\{(x, y)|x \geq a, y \geq b\}$. An example input is given in Figure 0.1. Assume that the red and blue points are given in separate arrays R and B respectively. Give an $O(n)$-time algorithm that computes the L1-closest red-blue pair in this special case and briefly justify the correctness of your algorithm. **Your algorithm should not be a divide and conquer algorithm.** We will use this part in the combine step of a divide-and-conquer algorithm in part (c) of this question.

Solution: Observe that by definition of L1-distance, the distance between a red point p and a blue point q is $|p.x - q.x| + |p.y - q.y| = q.x - p.x + q.y - p.y = (q.x + q.y) - (p.x + p.y)$. Therefore, we have to find the blue point q with the minimum $q.x + q.y$ and the red point with the maximum $p.x + p.y$ to minimize the L1-distance.

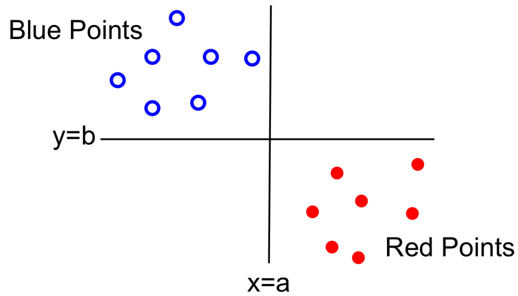

Figure 0.2: Sample input for part(b). Red points have their insides filled and blue points are drawn with empty insides.

(b) (5 marks) Now consider the special case where all red points are in the region $\{(x, y)|x \geq a, y \leq b\}$ and all blue points are in the region $\{(x, y)|x \leq a, y \geq b\}$. An example input is given in Figure 0.2. Similar to part (a), assume that the red and blue points are given in separate arrays R and B respectively. **Solve this special-case problem by a reduction.** Specifically, reduce your problem to the special-case problem given in part (a) and use your algorithm from part (a). Argue very briefly why your reduction is correct. Your entire reduction algorithm should take $O(n)$ time.

Solution: For each point x, blue or red, taking 2a-x reflects it over the x=a vertical line:

1. A blue point: x: $a + |x - a| = a + (a - x) = 2a - x$
2. A red point: x: $a - |x - a| = a - (x - a) = 2a - x$

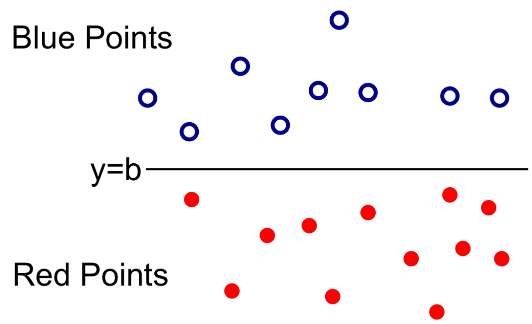

Figure 0.3: Sample input for part(c). Red points have their insides filled and blue points are drawn with empty insides.

(c) (10 marks) Next consider the case where all red points are below the horizontal line $y = b$ and all blue points are above the line $y = b$. Similar to part (a), assume that the red and blue points are given in separate arrays R and B respectively. Give an $O(n \log n)$-time **divide-and-conquer algorithm** that computes the L1-closest red-blue pair in this case (consider using your subroutines from (a) and (b) in your algorithm). For simplicity, your algorithm can return only the distance of closest pair and not the actual pair. Give the pseudocode of your algorithm and briefly justify its correctness. Write down and solve its runtime recurrence. You can get full marks even if you did not complete parts (a) or (b) by assuming you are given the subroutines for those parts.

Solution:

Let $P = R \cup B$
Sort(P by x-axis).
DC-L1-CP(P of n points):
if $n \leq 2$: return dist($P[1], P[2]$) if they are red-blue; or $+\infty$ otherwise.;
let x_m be $P[n/2]$.
closestL = DC-L1-CP(P[1,...,n/2]);
closestR = DC-L1-CP(P[n/2+1,...,n]);
$P_{left-blue}$ = select blue points from $P[1, ..., n/2]$
$P_{left-red}$ = select red points from $P[1, ..., n/2]$
$P_{right-blue}$ = select blue points from $P[n/2 + 1, ..., n]$
$P_{right-red}$ = select red points from $P[n/2 + 1, ..., n]$
closestSpan1 = subroutineFromA($P_{left-red}$,$P_{right-blue}$);
closestSpan2 = subroutineFromB($P_{right-red}$,$P_{left-blue}$);
return min{closestL, closestR, closestSpan1, closestSpan2}

Runtime: $T(n) = 2T(n/2) + O(n) = O(\log n)$.

(Extra space)

5. (18 marks) *Greedy Algorithms.* It is desired to perform a series of n musical pieces, denoted $M_1, vdots, M_n$, in some order. Each piece M_i has a length of L_i and a deadline of t_i. It is also desired that the performance of each M_i be completed before M_i's deadline t_i or as soon after t_i as possible. In particular any M_i that finishes at time $f_i > t_i$ incurs a penalty $p_i = f_i - t_i$. If M_i finishes at time $f_i \leq t_i$ then the penalty $p_i = 0$. The question now is to determine the ordering that incurs the smallest penalty, where the penalty of the ordering is defined to be

$$\max\{p_i : 1 \leq i \leq n\}.$$

For example, suppose that M_1 has length 4 and deadline 8; M_2 has length 3 and deadline 7; and M_3 has length 5 and deadline 9. Consider the ordering M_3, M_2, M_1. Then M_3 finishes at time $5 \leq 9$, so $p_3 = 0$. M_2 finishes at time $8 > 7$, so $p_2 = 1$. M_1 finishes at time $12 > 8$, so $p_1 = 4$. The penalty of this ordering is $\max\{0, 1, 4\} = 4$.

(a) (4 marks) Consider two possible greedy algorithms for this problem: (1) $\text{Greedy}_{\text{edf}}$ that schedules the ***e***arliest ***d***eadline piece ***f***irst; (2) $\text{Greedy}_{\text{sf}}$, that schedules the ***s***hortest piece ***f***irst. Give a counter example input to show that one of these algorithms is incorrect. Clearly demonstrate the output of both algorithms and which one is incorrect.

Solution: Consider $L_1 = 2$, $t_1 = 5$, $L_2 = 3$, $t_2 = 3$.

$\text{Greedy}_{\text{edf}}$ schedules M_2 then M_1. M_2 finishes at time $3 \leq 3$, so $p_2 = 0$. M_1 finishes at time $5 \leq 5$, so $p_1 = 0$. The maximum penalty is 0.

$\text{Greedy}_{\text{sf}}$ schedules M_1 then M_2. M_1 finishes at time $2 \leq 5$, so $p_1 = 0$. M_2 finishes at time $5 > 3$, so $p_2 = 5 - 3 = 2$. The maximum penalty is 2.

For this problem instance, $\text{Greedy}_{\text{eff}}$ yields a better solution, so $\text{Greedy}_{\text{sf}}$ cannot be a correct greedy algorithm.

(b) (14 marks) Give a detailed pseudocode description of the other algorithm (i.e., the correct algorithm) and prove that it is correct (i.e., it always finds the optimal solution).

Hint: Consider the possibility that two pieces are "out-of-order" in an optimal solution.

Solution: The algorithm to compute the penalty using $\text{Greedy}_{\text{edf}}$ is as follows:

1. sort the items by deadline in increasing order
2. F = 0
3. p = 0
4. for i = 1 to n do
5.   F = F + L_i
6.   if $F > t_i$ then $P = \max\{F - t_i, P\}$
7. return(P)

Proof of correctness: Suppose an optimal solution $\mathcal{O}$ schedules some M_i before M_j, where $i > j$ (i.e., $t_i \geq t_j$). We can assume $j = i + 1$. Consider the solution $\mathcal{O}'$ obtained by swapping M_i and M_{i+1}. We only need to consider the penalties associated with M_i and M_{i+1} in $\mathcal{O}$ and $\mathcal{O}'$. Let F denote the time at which M_i starts in $\mathcal{O}$. Then in $\mathcal{O}$, we have

$$p_i = \max\{F + L_i - t_i, 0\}$$

and

$$p_{i+1} = \max\{F + L_i + L_{i+1} - t_{i+1}, 0\}.$$

In $\mathcal{O}'$, we have

$$p'_{i+1} = \max\{F + L_{i+1} - t_{i+1}, 0\}$$

and

$$p'_i = \max\{F + L_i + L_j - t_i, 0\}.$$

It is easy to see that $p_{i+1} \geq p'_{i+1}$ because $L_i \geq 0$, and $p_{i+1} \geq p'_i$ because $t_i \geq t_{i+1}$. Therefore

$$\max\{p_i, p_{i+1}\} \geq p_{i+1} \geq \max\{p'_i, p'_{i+1}\}$$

and it follows that the penalty of $\mathcal{O}$ is $\geq$ the penalty of $\mathcal{O}'$. Since $\mathcal{O}$ is optimal, $\mathcal{O}'$ must also be optimal. By a sequence of swaps of this type, we can transform $\mathcal{O}$ into the greedy solution, maintaining optimality at every step.

(Extra space.)

(Extra space.)

(Extra space.)