Problem Set 2

CSE 373 Fall 2015

Due October 9 2015

1 Notes on Grading.

- You can write "I don't know" for any question and receive 25% credit. You can take this option for any numbered problem, but not for part of a problem. For example, you can answer 3.1 and write "I don't know" for 3.2, but you can't write part of the solution for 3.2 and then write "I don't know" for the rest.
- You get a 10% bonus for typing your homework. You are encouraged to use LaTeX. You must type your entire homework to receive the bonus. The 10% bonus does not apply to problems answered with "I don't know."

2 Asymptotic Growth

Order the following functions, from the slowest growing to the fastest:
$(n)^{1/8}$, $(\log n)^{20}$, $(5)^n$, $(n)^{0.576}$, $\log\log n, (2)^{\sqrt{n}}$, $(n)^5$

Solution: $O(\log\log n) \leq O((\log n)^{20}) \leq O((n)^{1/8}) \leq O((n)^{0.576}) \leq O((n)^5) \leq O((2)^{\sqrt{n}}) \leq O((5)^n)$

Grading: points = length of longest correct chain given. Extra point (i.e. 8 points total) for all correct.

3 O and Ω

Let

$$A = \{n,\ n^2,\ n^3,\ 2^n\}$$

$$B = \{10^{100}n^2 + logn,\ 1000logn,\ 5n^{1.5}\}$$

and

$$C = \{n^3 + logn,\ (5)^n,\ n^{logn}\}$$

List every function $f(n) \in A$ that has the property that for all $g(n) \in B$ and $h(n) \in C$, $f(n) = \Omega(g(n))$ and $f(n) = O(h(n))$.

Solution: $f(n) = n^2, n^3$

Grading: 2 points for each element of A classified correctly. (8 points total)

4 Counting Sort

1. Given an input of thousands of characters, where each character is in the range [a,z], describe how you would use the counting sort for sorting this input, and why the run time will be $O(n)$.
2. Given a length n, write an algorithm, in pseudo code, which generates a list of numbers that would cause the counting sort to run in $O(n^2)$.

Solution: Withheld, since we might assign this problem later.

5 Partition

The following array has been partitioned. Which elements could have been the pivot value?

$$[31, 0, 25, 47, 53, 82, 79, 64, 98]$$

Solution: In a partition operation, all elements with values less than the pivot locate at one side of the pivot while all elements greater than the pivot locate at another side. In the array, 47, 53, 98 could have been the pivot.

Grading: 1/2 point for each number classified correctly, plus an extra half point for all correct. (5 points total)

6 Asymptotic analysis

Assume that $f(n)$ and $g(n)$ are greater than 1 and monotonically increasing functions, prove or find a counterexample to the following:

- If $f(n) = o(g(n))$, then $\log(f(n)) = o(\log(g(n)))$
- If $f(n) = O(g(n))$, then $\log(f(n)) = O(\log(g(n)))$
- If $f(n) = o(g(n))$, then $2^{(f(n))} = o(2^{(g(n))})$
- If $f(n) = O(g(n))$, then $2^{(f(n))} = O(2^{(g(n))})$

Grading: 10 points each. 3 points for getting truth/falsehood correct, 7 points for a valid counterexample or proof. 2 points for counter-examples that only meet half the required criteria. 2 points given for examples demonstrating the truth of a statement (but examples are not a proof).

Solutions:

- If $f(n) = o(g(n))$, then $\log(f(n)) = o(\log(g(n)))$

  Solution:

  Counterexample:

  Let $f(n) = n$ and $g(n) = n^2$

  $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ and $\lim_{n\to\infty} \frac{\log f(n)}{\log g(n)} = \frac{1}{2}$.

  Therefore we have $f(n) = o(g(n))$ but $\log(f(n)) \neq o(\log(g(n)))$. The statement is invalid.

- If $f(n) = O(g(n))$, then $\log(f(n)) = O(\log(g(n)))$

  Solution:

  Proof:

  Since $f(n) = O(g(n))$, which means $\exists c$ and n_0, such that $\forall n > n_0$, we have $f(n) \leq cg(n)$. As $f(n)$ and $g(n)$ are greater than 1 and monotonically increasing, we have

  $$\log f(n) \leq \log c + \log g(n), \forall n > n_0 \tag{1}$$

  Since c and n_0 are constants, there must be a constant c' such that

  $$c' \geq \frac{\log c}{\log g(n_0)} + 1 \tag{2}$$

which means $(c'-1)\log g(n_0) \geq \log c$.

Therefore we have $\exists c', n_0$, $\forall n > n_0$,

$$\log f(n) \leq \log c + \log g(n) \leq c' \log g(n) \tag{3}$$

which means $\log f(n) = O(\log g(n))$. Therefore the statement is valid.

- If $f(n) = o(g(n))$, then $2^{(f(n))} = o(2^{(g(n))})$

  Solution:

  Proof:

  For $f(n) > 1$ and $g(n) > 1$, if $f(n) = o(g(n))$, we have: $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$. Here we aim to prove $\lim_{n\to\infty} \frac{2^{f(n)}}{2^{g(n)}} = 0$.

  Take an assumption called *Assumption 1*:

$$\lim_{n\to\infty} \frac{2^{f(n)}}{2^{g(n)}} = t > 0 \tag{4}$$

  We have

$$\log(\lim_{n\to\infty} \frac{2^{f(n)}}{2^{g(n)}}) = \log t \tag{5}$$

  and therefore

$$\lim_{n\to\infty} (\log \frac{2^{f(n)}}{2^{g(n)}}) = \log t \tag{6}$$

$$\lim_{n\to\infty} (f(n) - g(n)) = \log t = \lim_{n\to\infty} \log t \tag{7}$$

  based on which, we know

$$\lim_{n\to\infty} f(n) = \lim_{n\to\infty} (g(n) + \log t) \tag{8}$$

  and:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{g(n) + \log t}{g(n)} = 1 + \lim_{n\to\infty} \frac{\log t}{g(n)} \tag{9}$$

  Since $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, we have

$$\lim_{n\to\infty} \frac{\log t}{g(n)} = -1 \tag{10}$$

Because we have $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, $(f(n) \geq 1), (g(n) \geq 1)$, $f(n)$ and $g(n)$ are monotonically increasing, we know that:

$$0 = \lim_{n\to\infty} \frac{f(n)}{g(n)} \geq \lim_{n\to\infty} \frac{1}{g(n)} \geq 0 \tag{11}$$

which means

$$\lim_{n\to\infty} \frac{1}{g(n)} = 0 \tag{12}$$

and

$$\lim_{n\to\infty} g(n) = +\infty \tag{13}$$

From Equation 10 and 13 we can get that

$$\log t = -\lim_{n\to\infty} g(n) = -\infty \tag{14}$$

which means $t = 0$. It's a contradiction to *Assumption 1*. Therefore, the *Assumption 1* doesn't hold.

Hence we have:

$$\lim_{n\to\infty} \frac{2^{f(n)}}{2^{g(n)}} = t \leq 0 \tag{15}$$

While $2^{f(n)} > 0$ and $2^{g(n)} > 0$, we have $t \geq 0$. Therefore, we get:

$$\lim_{n\to\infty} \frac{2^{f(n)}}{2^{g(n)}} = t = 0 \tag{16}$$

which means $2^{f(n)} = o(2^{g(n)})$. The statement is valid.

- If $f(n) = O(g(n))$, then $2^{(f(n))} = O(2^{(g(n))})$

  Solution:

  Counterexample:

  Let $f(n) = 2\log n$ and $g(n) = \log n$.

  We have $2^{f(n)} = n^2$ and $2^{g(n)} = n$.

  It is obvious that $f(n) = O(g(n))$ but $2^{f(n)} \neq O(2^{g(n)})$. Therefore the statement is invalid.

7 Divide-and-conquer

A gray-scale image of size $(n \times n)$ is a $(n \times n)$ matrix of integers. Given that we have a very quick algorithm *rectangularCopy*() to copy a rectangular chunk of pixel of size from one location to another:

1. Design a divide-and-conquer algorithm to rotate an image 90^o clockwise. Assume that n is a power of 2. (You can use figures to help illustration).
2. If n is an arbitrary integer, how to modify the algorithm in question 1?
3. If the running time of *rectangularCopy*() on $(a \times a)$ matrix is $O(a^2)$, what's the running time of your algorithm?
4. If the running time of *rectangularCopy*() on $(a \times a)$ matrix is $O(a)$, what's the running time of your algorithm?

Grading:

1. 5 points for a correct an complete explanation of the algorithm (or pseudocode).
2. 5 points for generalizing to arbitrary sizes.
3. 5 points for each analysis. Must write down recurrence. Can use Master Theorem to solve it.

- **1. Solution**:

  Pseudocode of the algorithm is shown as following:

Algorithm 1 Rotate Square Image 90° Clockwise

```
1: function RotateSq90Clockwise($A, n$)    ▷ Where A - input matrix, n - width of A(and power of 2)
2:     if $n \leq 1$ then
3:         return $A$
4:     else
5:         $A_1 \leftarrow$ RotateSq90Clockwise($A[(0 : \frac{n}{2} - 1), (0 : \frac{n}{2} - 1)]$, $\frac{n}{2}$)
6:         $A_2 \leftarrow$ RotateSq90Clockwise($A[(0 : \frac{n}{2} - 1), (\frac{n}{2} : n - 1)]$,$\frac{n}{2}$)
7:         $A_3 \leftarrow$ RotateSq90Clockwise($A[(\frac{n}{2} : n - 1), (\frac{n}{2} : n - 1)]$,$\frac{n}{2}$)
8:         $A_4 \leftarrow$ RotateSq90Clockwise($A[(\frac{n}{2} : n - 1), (0 : \frac{n}{2} - 1)]$,$\frac{n}{2}$)
9:         $A[(0 : \frac{n}{2} - 1), (0 : \frac{n}{2} - 1)] \leftarrow$ rectangularCopy($A_4$)
10:        $A[(0 : \frac{n}{2} - 1), (\frac{n}{2} : n - 1)] \leftarrow$ rectangularCopy($A_1$)
11:        $A[(\frac{n}{2} : n - 1), (\frac{n}{2} : n - 1)] \leftarrow$ rectangularCopy($A_2$)
12:        $A[(\frac{n}{2} : n - 1), (0 : \frac{n}{2} - 1)] \leftarrow$ rectangularCopy($A_3$)
13:        return $A$
14:    end if
15: end function
```

- **2. Solution**:

  (There are multiple possible approaches for this question. Any valid method will be considered correct in grading.)

  Note that in the last question, we assume n is the power of 2 so that we don't need to worry about the width of a matrix $\frac{n}{2}$ not being integer in the recurrence. When n is not a power of 2, the only difficulty we have is to make sure that the size of matrix is an integer. To enable this, we define a function which rotate arbitrary size image as following:

Algorithm 2 Rotate Rectangular Image 90° Clockwise

```
1: function ROTATEREC90CLOCKWISE($A, m, n$)    ▷ Where A - input matrix, m - height(#row) of A, n - width(#column) of A
2:     if $(m \leq 1) \wedge (n \leq 1)$ then
3:         return $A$
4:     else
5:         $A_1 \leftarrow$ RotateRec90Clockwise($A[(0 : \lceil \frac{m}{2} \rceil - 1), (0 : \lceil \frac{n}{2} \rceil - 1)], \lceil \frac{m}{2} \rceil, \lceil \frac{n}{2} \rceil$)
6:         $A_2 \leftarrow$ RotateRec90Clockwise($A[(0 : \lceil \frac{m}{2} \rceil - 1), (\lceil \frac{n}{2} \rceil : n - 1)], \lceil \frac{m}{2} \rceil, \lfloor \frac{n}{2} \rfloor$)
7:         $A_3 \leftarrow$ RotateRec90Clockwise($A[(\lceil \frac{m}{2} \rceil : m - 1), (\lceil \frac{n}{2} \rceil : n - 1)], \lfloor \frac{m}{2} \rfloor, \lfloor \frac{n}{2} \rfloor$)
8:         $A_4 \leftarrow$ RotateRec90Clockwise($A[(\lceil \frac{m}{2} \rceil : m - 1), (0 : \lceil \frac{n}{2} \rceil - 1)], \lfloor \frac{m}{2} \rfloor, \lceil \frac{n}{2} \rceil$)
9:         Create a matrix $B$ of size $n \times m$
10:        $B[(0 : \lceil \frac{n}{2} \rceil - 1), (0 : \lfloor \frac{m}{2} \rfloor - 1)] \leftarrow$ rectangularCopy($A_4$)
11:        $B[(0 : \lceil \frac{n}{2} \rceil - 1), (\lfloor \frac{m}{2} \rfloor : m - 1)] \leftarrow$ rectangularCopy($A_1$)
12:        $B[(\lceil \frac{n}{2} \rceil : n - 1), (\lfloor \frac{m}{2} \rfloor : m - 1)] \leftarrow$ rectangularCopy($A_2$)
13:        $B[(\lceil \frac{n}{2} \rceil : n - 1), (0 : \lfloor \frac{m}{2} \rfloor - 1)] \leftarrow$ rectangularCopy($A_3$)
14:        return $B$
15:    end if
16: end function
```

For the case where n is not power of 2, function RotateRec90Clockwise(A, n, n) rotates the image 90° clockwise.

- **3. Solution**:

  Assuming $f(a)$ is the running time of *rectangularCopy*() on a $(a \times a)$ matrix, the recurrence is as following:

$$\begin{aligned} T(n) &= 4T(\frac{n}{2}) + 4f(\frac{n}{2}) + c' \\ &= 4^2T(\frac{n}{2^2}) + 4^2f(\frac{n}{2^2}) + 4f(\frac{n}{2}) + c' + c' \\ &... \\ &= 4^iT(\frac{n}{2^i}) + 4^if(\frac{n}{2^i}) + 4^{i-1}T(\frac{n}{2^{i-1}}) + ... + 4f(\frac{n}{2}) + ic' \\ &= 4^iT(\frac{n}{2^i}) + \sum_{j=1}^{i} 4^j f(\frac{n}{2^j}) + ic' \end{aligned}$$

  The basis case is $T(1) \leq c$, in which $n/2^i = 1$. We have $2^i = n$ and $i = \log n$. Since we know that $f(a) = O(a^2)$, it means $\exists c_0$ s.t. $f(a) \leq c_0(a^2)$. Furthermore we have:

$$4^j f(\frac{n}{2^j}) \leq 4^j(c_0(\frac{n}{2^j})^2) = c_0(4^j(\frac{n}{2^j})^2) = c_0n^2$$

Therefore we have

$$\begin{aligned}
T(n) &= 4^i T(\frac{n}{2^i}) + \sum_{j=1}^{i} 4^j f(\frac{n}{2^j}) + ic' \\
&\le cn^2 + \sum_{j=1}^{i} c_0 n^2 + ic' \\
&= cn^2 + \log n(c_0 n^2) + c' \log n \\
&= O(n^2 \log n)
\end{aligned}$$

- **4. Solution**:

  Similar to last question, we have

  $$T(n) = 4^i T(\frac{n}{2^i}) + \sum_{j=1}^{i} 4^j f(\frac{n}{2^j}) + ic'$$

  The basis case is $T(1) \le c$, in which $n/2^i = 1$. We have $2^i = n$ and $i = \log n$. Since $f(a) = O(a)$, it means $\exists c_0$ s.t. $f(a) \le c_0(a)$, and we have

  $$4^j f(\frac{n}{2^j}) \le 4^j(c_0(\frac{n}{2^j})) = c_0(4^j(\frac{n}{2^j})) = c_0(2^j n)$$

  Therefore we have

  $$\begin{aligned}
T(n) &= 4^i T(\frac{n}{2^i}) + \sum_{j=1}^{i} 4^j f(\frac{n}{2^j}) + ic' \\
&\le cn^2 + c_0 \sum_{j=1}^{i} (2^j n) + ic' \\
&= cn^2 + c_0(2^{i+1} - 2)n + ic' \\
&= cn^2 + c_0(2n - 2)n + c' \log n \\
&= (c + 2c_0)n^2 - 2c_0 n + c' \log n \\
&= O(n^2)
\end{aligned}$$

8 Applications of findKthSmallest

1. Show how to find all the k_1- through k_2th-smallest elements of an array A of n distinct elements in $O(n)$ time.

2. Show how to determine in linear time whether any element of an array A of n elements occurs at least $n/2$ times.

3. Given n distinct elements $X = \{x_1, \ldots, x_n\}$ with weights $w_1, \ldots, w_n$ such that $\sum_i w_i = 1$, the weighted median of X is the element x_k such that

$$\sum_{i:x_i<x_k} w_i < \frac{1}{2}$$

and

$$\sum_{i:x_i>x_k} w_i \leq \frac{1}{2}$$

Show how to compute the weighted median in linear time.

1. 5 points for this easy algorithm. Must be linear time for full credit. 1 point for an $\omega(n)$ algorithm.

2. 5 points. Hashing not allowed. 2 points for the insight about an element that occurs $n/2$ times being the median, and 3 points for the rest of the algorithm.

3. 10 points. 5 points for realizing the algorithm needs to be generalized to more than just weighted median in order to be do-able as a divide and conquer. 5 points for the code. 1 point for an $\omega(n)$ algorithm.

- **1. Solution**:
  - 1. $a_1 \leftarrow$ findKthSmallest(A, k_1)
  - 2. $a_2 \leftarrow$ findKthSmallest(A, k_2)
  - 3. Scan the array A, return all elements x s.t. $a_1 \leq x \leq a_2$.

- **2. Solution**:
  - 1. $a_{midF} \leftarrow$ findKthSmallest$(A, \lfloor \frac{n}{2} \rfloor)$
  - 2. $a_{midC} \leftarrow$ findKthSmallest$(A, \lfloor \frac{n}{2} \rfloor + 1)$
  - 3. Scan the array A, count the number(n_F) of elements x s.t. $x = a_{midF}$.
  - 4. Scan the array A, count the number(n_C) of elements x s.t. $x = a_{midC}$.
  - 5. If the number n_F(or n_C) is larger than $\frac{n}{2}$, then there is an element (a_{midF} or a_{midC}) occurs at least $\frac{n}{2}$ times. Otherwise there is no such element.

- **3. Solution**:

  (1). We define a function $(A', W') = \text{findAllElementBetweenK1nK2}(A, W, k_1, k_2)$. This function returns those elements in A between the k_1th smallest and k_2th smallest elements(A'), as well as the associated weights(W'):

  - 1. $a_1 \leftarrow \text{findKthSmallest}(A, k_1)$
  - 2. $a_2 \leftarrow \text{findKthSmallest}(A, k_2)$
  - 3. Scan the array A, return all elements x s.t. $a_1 \le x \le a_2$ as A'. Meanwhile return all associated weights as W'.

  Given that the size of A is n, the complexity of findAllElementBetweenK1nK2() is $O(n)$.

  (2). We define a function $(S) = \text{sumLargerThanKthElementsWeights}(A, W, k)$. This function find out in A a set of elements larger than the kth smallest element, and sum up all associated weights:

  - 1. $a_k \leftarrow \text{findKthSmallest}(A, k)$
  - 2. Scan the array A, find all elements x s.t. $x \le a_k$, and sum up all associated weights in W. Return the summation S.

  The complexity of sumLargerThanKthElementsWeights() is $O(n)$ if the size of A is n.

(3). An algorithm of finding the arbitrary weights split is as following:

Algorithm 3 Find Weights Split

```
1: function FindWeightsSplit($A, W, L, S_l, S_h, S_a$)    ▷ Where $A$ - input matrix, $W$ - weights of $A$, $L$ - size of $A$, $S_l$ - lower-side weights summation, $S_h$ - higher-side weights summation, $S_a$ - all weights summation
2:     $k \leftarrow \frac{L}{2}$
3:     $x^k \leftarrow$findKthSmallest($A$, $k$)
4:     $w^k \leftarrow$weight of $x^k$
5:     $S_h^k \leftarrow$sumLargerThanKthElementsWeights($A, W, k$)
6:     $S_l^k \leftarrow (S_a - S_h^k - w^k)$
7:     if $S_l^k < S_l \wedge S_r^k \leq S_r$ then
8:         return $x^k$
9:     else if $S_l^k < S_l \wedge S_r^k > S_r$ then
10:        $S_l' \leftarrow S_l - S_l^k$
11:        $S_h' \leftarrow S_h$
12:        $(A', W') \leftarrow$findAllElementBetweenK1nK2($X, W, \frac{L}{2} + 1, L$)
13:        $L' \leftarrow$ length of $A'$
14:        $S_a' \leftarrow S_h^k$
15:        $x \leftarrow$FindWeightsSplit($A', W', L', S_l', S_h', S_a'$)
16:        return $x$
17:    else if $S_l^k \geq S_l \wedge S_r^k \leq S_r$ then
18:        $S_h' \leftarrow S_h - S_h^k$
19:        $S_l' \leftarrow S_l$
20:        $(A', W') \leftarrow$findAllElementBetweenK1nK2($X, W, 1, \frac{L}{2} - 1$)
21:        $L' \leftarrow$ length of $A'$
22:        $S_a' \leftarrow S_l^k$
23:        $x \leftarrow$FindWeightsSplit($A', W', L', S_l', S_h', S_a'$)
24:        return $x$
25:    end if
26: end function
```

FindWeightsSplit($A, W, L, \frac{1}{2}, \frac{1}{2}, 1$) returns the weighted median as requested. Given that findAllElementBetweenK1nK2() and sumLargerThanKthElementsWeights() are linear time(of complexity $O(n)$), the complexity of FindWeightsSplit() is $T(n)$:

$$\begin{aligned}
T(n) &\leq T(\frac{n}{2}) + c_1 n + c_2 n \\
&\leq T(\frac{n}{2}) + c_3 n \\
&\leq T(\frac{n}{4}) + c_3(n + \frac{n}{2}) \\
&... \\
&\leq T(\frac{n}{2^i}) + c_3 \sum_{j=0}^{i-1} \frac{n}{2^j} \\
&\leq T(\frac{n}{2^i}) + c_3(1 - \frac{1}{2^i})n
\end{aligned}$$

The basis case is $\frac{n}{2^i} = 1$(or earlier), therefore,

$$T(n) \leq c + c_3(n-1) = O(n)$$