Solutions to Intro to Algorithms Midterm

Total: 122 points

Hint: Do not try to earn 122 points. Try to earn 100 points instead.

1 True or false. You ***must*** justify your answers. ***No justifications, no credits.*** (16%)

a) $O(n) = \{f(n) : \exists c > 0 \text{ such that } 0 \le f(n) \le cn \text{ for all } n > 0\}$

Hint: Compare it with book's definition on big-O.

$O(n) = \{f(n) : \exists c > 0\ n_0 > 0 \text{ such that } 0 \le f(n) \le cn \text{ for all } n \ge n_0\}$

Solution: *True*

Let $\hat{O}(n) =$ book's definition on big-O.

We show that $O(n) = \hat{O}(n)$

$O(n) \subseteq \hat{O}(n)$

This is trivial – simply pick $n_0 = 1$.

$\hat{O}(n) \subseteq O(n)$

Let $f(n) = \hat{O}(n)$

Then, $0 \le f(n) \le cn\ \forall\ n \ge n_0$, for some c and n_0

For $n < n_0$, it is possible that $f(n) > cn$.

However, we may choose a large enough constant c' to handle these cases.

For each $0 < i < n_0$, let $c_i = f(i)/i$

Let $c' = \max\{c, \max_{0<i<n_0} c_i\}$

Then, $0 \le f(n) \le c'n\ \forall\ n > 0$

b) $O(n^k) = O(n)^k$ for any integer k

Hint: Consider $k < 0, k = 0,$ and $k > 0$

Solution: *False*

1 $O(n^k) = O(n)^k$ for $k > 0$

2 $O(n^0) \ne O(n)^0$

$\because O(n^0) = O(1)$.

But, $O(n)^0 = \{f^0(n) : f(n) = O(n)\}$

$= \{g(n) : g(n) = 1 \text{ for all large enough } n\}$

3 $O(n^k) \ne O(n)^k$ for $k < 0$.

$\because O(n)^k = \Omega(n^k)$ for $k < 0$

c) Let $S(n) = S(n-1) + O(1)$
$T(n) = T\left(n/2 + \sqrt{n}\right) + n$
Then, $S(n) = \Theta(T(n))$

Solution: *False*
$S(n) = O(n)$ and $T(n) = \Theta(n) \Rightarrow S(n) = O(T(n))$
Proof of $S(n) = O(n)$

$$\begin{aligned} S(n) &= S(n-1) + O(1) \\ &= S(n-2) + O(1) + O(1) \\ &= \cdots \\ &= \sum_{i=1}^{n} O(1) = O\left(\sum_{i=1}^{n} 1\right) = O(n) \end{aligned}$$

[HW#2, 4a]
Proof of $T(n) = \Theta(n)$
Let $S(n) = S(n/2) + n$
$U(n) = U(2n/3) + n$
Then, by the master theorem, $S(n) = \Theta(n)$ and $U(n) = \Theta(n)$
Since $S(n) \le T(n) \le U(n)$ for n large enough, we have $T(n) = \Theta(n)$.

d) Heapsort is a good choice for sorting a linked list.

Solution: *False*
Since the parent and its children within a heap aren't located in consecutive memory, Heapsort relies strongly on random access and runs poorly on sequential access media.

2 For each algorithm below, give a recurrence that describes its running time and give a tight bound or upper bound of the running time. You need not justify your answers. Note that

- this problem asks for running time, rather than worst-case running time, and
- give a tight bound whenever possible. (12%)

a) Strassen's algorithm
Solution: [Chap04, p6]
$T(n) = 7T(n/2) + \Theta(n^2) \Rightarrow T(n) = \Theta\left(n^{\lg 7}\right)$

b) Randomized Quicksort

Solution: [Chap07, p18]

$$T(n) = \sum_{k=1}^{n} X_k \cdot (T(k-1) + T(n-k) + \Theta(n))$$

where $X_k = \mathrm{I}\{\text{the pivot is the } k\text{th smallest element}\}$

$T(n) = O(n^2)$, since the worst-case running time of quicksort is $\Theta(n^2)$.

c) Binary search

Solution: [HW#2, 6]

$$T(n) \le T\left(\left\lceil \frac{n-1}{2} \right\rceil\right) + \Theta(1)$$

or, a simpler recurrence

$$T(n) \le T\left(\frac{n}{2}\right) + \Theta(1)$$

Clearly, $T(n) = O(\lg n)$

3 Prove by the definition of big-O that $(2n+3) \times O(n) = O(n^2)$. (8%)

Hint: Be sure to specify the constants required by the big-O definition.

Solution

Let $f(n) = O(n)$

We have to show that $(2n+3) \cdot f(n) = O(n^2)$

$$\begin{aligned}(2n+3) \cdot f(n) &\le (2n+3) \cdot cn \qquad \forall\, n \ge n_0 \text{ for some } n_0 \\ &= 2cn^2 + 3cn \\ &= 3cn^2 - (cn^2 - 3cn) \\ &\le 3cn^2\end{aligned}$$

as long as

$cn^2 - 3cn \ge 0 \Rightarrow cn - 3c \ge 0 \Rightarrow n \ge 3$

Thus,

$(2n+3) \cdot f(n) \le 3cn^2 \quad \forall\, n \ge \max(n_0, 3)$

as desired.

4 Given

$T(n) = 8T(n/2) + O(n^2)$

Use constructive induction to show that $T(n) = O(n^3)$. (8%)

Hint: Try $T(n) \leq dn^3 - d'n^2$.

Solution: [Chap04, p33]

We shall prove that $T(n) \leq dn^3 - d'n^2$

$$\begin{aligned} T(n) &= 8T(n/2) + O(n^2) \\ &\leq 8T(n/2) + cn^2 \\ &\leq 8(d(n/2)^3 - d'(n/2)^2) + cn^2 \\ &= dn^3 - 2d'n^2 + cn^2 \\ &= dn^3 - d'n^2 - (d'n^2 - cn^2) \\ &\leq dn^3 - d'n^2 \end{aligned}$$

as long as

$d'n^2 - cn^2 \geq 0 \Rightarrow d' \geq c$

5 Consider the mergesort recurrence

$$T(n) = T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + T\left(\left\lceil\frac{n}{2}\right\rceil\right) + O(n)$$

Prove that $T(n) = O(n \lg n)$ by *domain transformation*.

That is, define $S(n) = T(n + \alpha)$, where α is a constant, chosen to make $S(n)$ satisfy a simpler recurrence. (8%)

Hint:

$$T(n) = T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + T\left(\left\lceil\frac{n}{2}\right\rceil\right) + O(n) \leq 2T\left(\left\lceil\frac{n}{2}\right\rceil\right) + O(n) \leq 2T\left(\frac{n}{2} + 1\right) + O(n)$$

Solution: [HW#2, 1]

Let $S(n) = T(n + \alpha)$

and suppose $S(n) \leq 2S(n/2) + O(n)$

Then

$T(n + \alpha) = S(n) \leq 2S(n/2) + O(n) = 2T(n/2 + \alpha) + O(n) \quad \cdots (1)$

and

$T(n + \alpha) \leq 2T\left(\frac{n+\alpha}{2} + 1\right) + O(n) \quad \cdots (2)$

From (1) and (2), we have

$n/2 + \alpha = \frac{n+\alpha}{2} + 1 \Rightarrow \alpha = 2$

Thus, $S(n) = T(n + 2)$

By the master theorem, $S(n) = O(n \lg n)$

It follows that

$$T(n) = S(n-2)$$
$$= O((n-2)\lg(n-2))$$
$$= O(n-2)O(\lg(n-2)) = O(n)O(\lg n) = O(n \lg n)$$

6 Consider a variant of mergesort that divides an array of n elements into $\sqrt{n}$ subarrays, each having $\sqrt{n}$ elements. The $\sqrt{n}$ sorted subarrays are then merged simultaneously with the help of a min-priority queue.

```
MERGESORT(A[1..n])
1   for i = 1 to √n do                               // √n subarrays
        MERGESORT(A[(i − 1)√n + 1..i√n])             // sort the ith subarray
2   MERGE(A[1..n])
```

```
MERGE(A[1..n])     // A[1..n] contains √n sorted subarrays
1   Let B[1..n] be a new array
2   Build a min-priority queue Q on the √n smallest elements, one from each
    sorted subarray
3   for k = 1 to n do
        B[k] = EXTRACT-MIN(Q)
        // Suppose the element just extracted comes from the ith subarray
        if the ith subarray is not empty then
            INSERT(Q, the next element of the ith subarray)
4   Copy B[1..n] back to A[1..n]
```

a) Show that the running time of MERGE$(A[1..n])$ is $O(n \lg \sqrt{n})$. (4%)

Solution

Step 2 takes a time in $O(\lg \sqrt{n})$.

Step 4 takes a time in $O(n)$.

Analysis of step 3

There are $O(n)$ EXTRACT-MIN and INSERT operations, each taking $O(\lg|Q|)$ running time. Since $|Q| \le \sqrt{n}$, step 3 takes a time in

$$O(n) \times O(\lg \sqrt{n}) = O(n \lg \sqrt{n})$$

Thus, the total time is

$$O(\lg \sqrt{n}) + O(n) + O(n \lg \sqrt{n}) = O(n \lg \sqrt{n})$$

b) Let $T(n)$ be the running time of MERGESORT($A[1..n]$), then
$T(n) = \sqrt{n}T(\sqrt{n}) + O(n \lg \sqrt{n})$
Give an asymptotic upper bound for $T(n)$. (8%)
Hint: Range transformation $S(n) = T(n)/n$ and change of variable

Solution: [HW#2, 5]
We shall solve this recurrence in two steps.
Step 1: Range transformation
Let $S(n) = T(n)/n$
Then,
$T(n) = \sqrt{n}T(\sqrt{n}) + O(n \lg \sqrt{n}) \Rightarrow T(n)/n = T(\sqrt{n})/\sqrt{n} + O(\lg \sqrt{n})$
$\Rightarrow S(n) = S(\sqrt{n}) + O(\lg \sqrt{n})$
Step 2: Change of variable
Next, let $U(m) = S(2^m)$ (i.e. rename $m = \lg n$)
Then,
$U(m) = S(2^m) = S(2^{m/2}) + O(\lg 2^{m/2}) = U(m/2) + O(m)$
By the master theorem, $U(m) = O(m)$

Finally, we have
$T(n) = nS(n) = nU(\lg n) = n \times O(\lg n) = O(n \lg n)$

7 Consider the insertion sort

INSERTION-SORT(A, n)
for $i = 2$ **to** n
  $key = A[i]$
  $j = i - 1$
  while $j > 0$ and $A[j] > key$ // key (i.e. $A[i]$) is compared to $A[j]$
    $A[j+1] = A[j]$
    $j = j - 1$
  $A[j+1] = key$

a) Draw the decision tree for insertion sort on 3 elements. (8%)
Solution: [Chap08, p2]

b) In terms of the number of comparisons, for what value of n is insertion sort an optimal comparison sort in the worst case? (6%)
Hint: First, find the number of comparisons taken by insertion sort in the worst case. Then, compare it with the theoretical lower bound.

Solution: [Chap02, p7; Chap08, pp6~7]

Let $T(n) =$ the number of comparisons taken by INSERTION-SORT(A, n) in the worst case. Then,

$$T(n) = \sum_{i=2}^{n} (i-1) = \frac{n(n-1)}{2}$$

On the other hand, the minimal number of comparisons needed is $\lceil \lg n! \rceil$.

We have

$T(1) = 0 = \lceil \lg 1! \rceil$

$T(2) = 1 = \lceil \lg 2! \rceil$

$T(3) = 3 = \lceil \lg 3! \rceil$

$T(4) = 6 > \lceil \lg 4! \rceil = 5$

Thereafter, $T(n) > \lceil \lg n! \rceil$, since $T(n)$ grows faster than $\lceil \lg n! \rceil$.

Thus, insertion sort is optimal for $n = 1,2,3$.

8 (Continuing 7)

Assume that the n elements are distinct and each of the $n!$ possible inputs is equally likely.

a) Define the indicator random variable

$X_{ij} = \mathrm{I}\{A[i] \text{ is compared to } A[j]\}, \quad 2 \le i \le n, 1 \le j \le i-1$

where $A[i]$ and $A[j]$ denote the values held in variables key and $A[j]$, respectively, at the time $A[j] > key$ is evaluated.

What is the value of $\mathrm{E}[X_{ij}]$? (4%)

Solution

First of all, the n elements are distinct and each of the $n!$ possible inputs is equally likely $\Rightarrow$ any element in the subarray $A[1..i]$ is equally likely to be the kth smallest element in that subarray, $1 \le k \le i$.

Also, $A[i]$ is compared to $A[j]$ $\Rightarrow A[i]$ is the kth smallest element in the subarray $A[1..i]$ for some k, $1 \le k \le j+1 \le i$

It follows that

$$\mathrm{E}[X_{ij}] = \Pr\{A[i] \text{ is compared to } A[j]\} = \frac{j+1}{i}$$

b) Let X be a random variable denoting the total number of times $A[j] > key$ is compared in the course of executing INSERTION-SORT(A, n).

Show that $\mathrm{E}[X] = n^2/2 + \Theta(n)$. (6%)

Hint: $X = \sum_{i=2}^{n} \sum_{j=1}^{i-1} X_{ij}$

Solution

Since $X = \sum_{i=2}^{n} \sum_{j=1}^{i-1} X_{ij}$, we have

$$\mathrm{E}[X] = \mathrm{E}\left[\sum_{i=2}^{n} \sum_{j=1}^{i-1} X_{ij}\right] = \sum_{i=2}^{n} \sum_{j=1}^{i-1} \mathrm{E}[X_{ij}] = \sum_{i=2}^{n} \sum_{j=1}^{i-1} \frac{j+1}{i}$$

$$= \sum_{i=2}^{n} \frac{\sum_{j=2}^{i} j}{i} = \sum_{i=2}^{n} \frac{i(i+1)/2 - 1}{i} = \frac{1}{2}\sum_{i=3}^{n+1} i - \sum_{i=2}^{n} \frac{1}{i}$$

$$= \frac{n^2 + 3n - 4}{2} - \sum_{i=2}^{n} \frac{1}{i} = \frac{n^2}{2} + \Theta(n)$$

9 Consider the following MSD radix sort on $A[1..n]$ in which each number $A[i]$ has d digits $x_d \dots x_2 x_1$. The value of each digit x_i satisfies $0 \le x_i \le k$ for some constant k.

MSD_RADIX_SORT$(A[1..n])$
for $i = d$ **downto** 1
for each pile having the same digit x_{i+1}, sort digit x_i by counting sort

a) What is the running time of MSD_RADIX_SORT$(A[1..n])$? (6%)
Hint: Compare it with LSD radix sort

Solution: [Chap08, p16]
When sorting digit x_i, suppose the counting sort is called j times on arrays of length $n_1, n_2, \dots, n_j$ such that $\sum_{l=1}^{j} n_l = n$.
Then, the total time spent on sorting digit x_i is

$$\sum_{l=1}^{j} \Theta(n_l + k) = \Theta\left(\sum_{l=1}^{j} (n_l + k)\right) = \Theta\left(\sum_{l=1}^{j} n_l + k\right) = \Theta(n + k)$$

The running time of MSD radix sort is $\Theta(d(n+k))$.
Note: MSD radix sort and LSD radix sort have the same time complexity, except that MSD radix sort needs $\Theta(d)$ stack spaces.

b) Suppose each $A[i]$ is a 32-bit unsigned integer, what is the best value of d and the resulting running time? (4%)
Hint: Compare it with LSD radix sort

Solution: [Chap08, p17]

Like LSD radix sort, the best way is to treat each digit as a $\lg n$-bit number. Thus, the best value of $\mathrm{d} = \lceil 32/\lg n \rceil$.
And, the resulting running time is

$$\Theta\left(\frac{32}{\lg n}\left(n + 2^{\lg n}\right)\right) = \Theta\left(\frac{32}{\lg n}(n + n)\right) = \Theta(n/\lg n)$$

10 Prove the following theorem.

THEOREM Finding the minimum and maximum of n elements needs ***at least*** $\lceil 3n/2 \rceil - 2$ comparisons in the worst case. (10%)

Solution: [Chap09, pp5~10]

11 Recall that, in the worst case, quicksort makes extremely unbalanced partitions ($0 : n-1$ split) that cause the depth of the recursion tree to be about n. But, in the best case, it makes balanced partitions ($(n-1)/2 : (n-1)/2$ split) that reduce the depth of the recursion tree to $\lg n$. Thus, to speed up quicksort, the depth of the recursion tree shall be limited.

The introspective sort (introsort) is a variant of quicksort that

- puts an $O(\lg n)$ limit on the depth of the recursion tree, and
- switches to heapsort when the depth limit is reached.

The following pseudocode also resorts to insertion sort when the array size is smaller than some threshold value.

INTROSORT($A[p..r], depth\_limit$)
if $(1 < r - p + 1 \le threshold)$ INSERTION_SORT($A[p..r]$)
else if $(depth\_limit == 0)$ HEAPSORT($A[p..r]$)
else
    $q =$ PARTITION($A[p..r]$)
    INTROSORT($A[p..q-1], depth\_limit - 1$)
    INTROSORT($A[q+1..r], depth\_limit - 1$)

a) What is the best value for the constant $threshold$? (2%)

1) 4   2) 16   3) 128   4) 256

Hint: HW#1

Solution: 2)

16 is a reasonable choice, based on our experience in profiling insertion sort and merge sort in Chap02.

Note: Introsort is used by C++ STL sort function template. The threshold value is 16 in SGI C++ STL, and 32 in VC++.

b) Consider the call

INTROSORT$(A[1..n], c\lg n)$

What is the best value for the constant c? (2%)

1) 0.5 2) 1.0 3) 1.5 4) 5.0

Hint: Don't call heapsort and insertion sort too frequently.

Solution: 3)

If the limit is too small, the algorithm will call heapsort too frequently. If the limit is too large, the algorithm will call insertion sort too frequently.

Note: $c = 1.5$ in VC++, and $c = 2.0$ as suggested by David Musser, the inventor of introsort.

Musser, David, Introspective Sorting and Selection Algorithms. *Software: Practice and Experience*, **27** (8), 983–993, 1997.

c) Prove that the call in b) takes $O(n\lg n)$ time in the worst case. (6%)

Hint: Count the running time of all calls to HEAPSORT and the running time of all calls to PARTITION.

Solution

INTROSORT is called $O(\lg n)$ times. Each call spends $\Theta(1)$ time, excluding INSERTION_SORT, HEAPSORT, or PARTITION time. Thus, the total time for INTROSORT is $O(\lg n)$.

PARTITION is called $O(\lg n)$ times. Each call spends $O(n)$ time. Thus, the total time for PARTITION is $O(n\lg n)$.

Suppose HEAPSORT is called j times on arrays of length $n_1, n_2, \dots, n_j$. Then, the total time for HEAPSORT is

$$\sum_{i=1}^{j} O(n_i \lg n_i) = \sum_{i=1}^{j} O(n_i \lg n) \quad \because n_i \le n$$

$$= \lg n \cdot \sum_{i=1}^{j} O(n_i)$$

$$= \lg n \cdot O\left(\sum_{i=1}^{j} n_i\right) \quad \because \sum_{i=1}^{j} n_i \le n$$

$$= \lg n \cdot O(n) = O(n\lg n)$$

Suppose INSERTION_SORT is called k times. Then, $k = O(n)$, since each call sorts at least one element. Each call takes $O(1)$ time, since it sorts an array of length $\leq threshold$. Thus, the total time for INSERTION_SORT is $O(n)$.

In conclusion, the running time of the call INTROSORT$(A[1..n], c\lg n)$ is

$O(\lg n) + O(n\lg n) + O(n\lg n) + O(n) = O(n\lg n)$

It follows that the worst-case running time is $O(n\lg n)$.

d) Prove that the call in b) takes $\Omega(n\lg n)$ time in the worst case. (4%)

Solution 1

Since introsort is a comparison sort and any comparison sort takes at least $\Omega(n\lg n)$ time in the worst case to sort n elements.

Solution 2

In the best case of balanced partitions, the depth of the recursion tree is at least $\Omega(\lg n)$. To see this, let k be the depth of the best-case recursion tree. Then,

$n/2^k \leq threshold \Rightarrow n/threshold \leq 2^k \Rightarrow \lg(n/threshold) \leq k$

Since $threshold$ is a constant, it follows that $k = \Omega(\lg n)$.

Also, the calls to PARTITION in each level of the recursion tree take $\Omega(n)$ time in total. Thus, the total time for PARTITION is $\Omega(n\lg n)$.

Finally,

the worst case running time

$\geq$ the best case running time

$\geq$ the total time for PARTITION in the base case

$=$ $\Omega(n\lg n)$