ALGORITHMS MIDTERM SOLUTION

1 a) [Problem 3-4 h), HW#1]

True

See HW#1 solution

b) [Chap06 Lecture note, pp7,13~14]

True

$T(n) \le O(n) \Rightarrow T(n) \le f(n)$ for some $f(n) = O(n)$

$\Rightarrow T(n) \le f(n)$ where $f(n) \le cn$ for n sufficiently large

$\Rightarrow T(n) \le cn$ for n sufficiently large

$\Rightarrow T(n) = O(n)$

$T(n) = O(n) \Rightarrow T(n) \le cn$ for n sufficiently large

$\Rightarrow T(n) \le f(n)$ $\because$ let $f(n) = cn = O(n)$

$\Rightarrow T(n) \le O(n)$

c) [Chap04 Lecture note, p13]

True

$T_1(n)$ and $T_2(n)$ can be rewritten as

$$T_1(n) = \begin{cases} \Theta(1) & n = 1 \\ 2T_1(n/2) + \Theta(n) & n > 1 \end{cases}$$

$$T_2(n) = \begin{cases} \Theta(1) & n \le 9999 \\ 2T_2(n/2) + \Theta(n) & n > 9999 \end{cases}$$

because $n \le 9999 \Rightarrow n^2$ is a finite integer.

Since the boundary cases of the recurrences are immaterial, $T_1(n)$ and $T_2(n)$ have the same order, i.e. $T_1(n) = \Theta(T_2(n))$. In fact, they are both in $\Theta(n \lg n)$.

d) [Past exam, 2010 midterm 1b]

True

Let

$A =$ "The running time of **HEAPESORT** is $O(n \lg n)$. ", and

$B =$ "The worst-case running time of **HEAPSORT** is $O(n \lg n)$."

Then

$A \Rightarrow B$

$\because$ the worst case is one of the cases

$B \Rightarrow A$

$\because$ running time (of any case) $\le$ running time of worst case $= O(n \log n)$

e) [Chap08 Lecture note, pp17~19]

False

Let $n = 65536$

The proposed algorithm takes a time in

$$\underbrace{\Theta(n+n)}_{\text{the pass over } d_0} + \underbrace{\Theta(n+n/2)}_{\text{the pass over } d_1} + \underbrace{\Theta(n+2)}_{\text{the pass over } d_2} = \Theta\left(4\tfrac{1}{2}n+2\right)$$

It is right to treat the signed bits differently. However, they needn't be treated ***alone***. A better way is to partition each signed integer as a 2-digit number $d_1 d_0$, where d_1 and d_0 are 16-bit digits (since $\lg 65536 = 16$), and then treat d_1 differently, as it contains the signed bits.

The running time is

$$\underbrace{\Theta(n+n)}_{\text{the pass over } d_0} + \underbrace{\Theta(n+n)}_{\text{the pass over } d_1} = \Theta(4n)$$

f) False

The upper bound is correct, but the lower bound isn't.

For the upper bound, we have

1) (running time of worst-case of size $n \geq n_0$) $\leq cn^2$
2) (running time of any case of size $n \geq n_0$)

$\leq$ (running time of worst-case of size $n \geq n_0$)

Clearly, $1 + 2 \Rightarrow$ (running time of any case of size $n \geq n_0$) $\leq cn^2$

Thus, if the running time of an instance of size $> cn^2$, then $n < n_0$.

Since there are only a finite number of instances for *each* $n < n_0$, there can thus be only a finite number of instances of size $n < n_0$ on which insertion sort takes a time $> cn^2$.

For the lower bound, we have

3) (running time of worst-case of size $n \geq n_1$) $\geq dn^2$
4) (running time of any case of size $n \geq n_1$)

$\leq$ (running time of worst-case of size $n \geq n_1$)

But, $3 + 4 \nRightarrow$ (running time of any case of size $n \geq n_1$) $\geq dn^2$

In fact, there are infinitely many instances of size $n \geq n_1$ on which insertion sort takes a time $< dn^2$. For example, it takes $\Theta(n)$ time to sort arbitrarily large instances in which the elements are already sorted.

2 [Problem 3-3, HW#1]

a) $2^n = \omega((3/2)^n)$

Proof

Let $c > 0$

Then, $2^n > c(3/2)^n \Rightarrow (4/3)^n > c \Rightarrow n > \log_{4/3} c \Rightarrow n \geq \lfloor \log_{4/3} c \rfloor + 1$

So, pick $n_0 = \lfloor \log_{4/3} c \rfloor + 1$

b) $\lg n! < n^2 < (\lg n)!$ where $f(n) < g(n)$ means $f(n) = o(g(n))$

Proof

$\lg n! = \Theta(n \lg n) \Rightarrow \lg n! = o(n^2)$

Also,

$\lg n! = \Theta(n \lg n) \Rightarrow \lg(\lg n)! = \Theta(\lg n \lg \lg n)$ $\because$ substitute $\lg n$ for n

$\lg n^2 = 2 \lg n = o(\lg n \lg \lg n)$ $\because 2 = o(\lg \lg n)$

From these we have

$\lg n^2 = o(\lg(\lg n)!) \Rightarrow n^2 = o((\lg n)!)$

3 [Ex 4.2-4, The solution is posted publicly.]

Use the master theorem to solve the recurrence

$T(n) = kT(n/3) + \Theta(n^2)$

Case 3: $n^2 = \Omega\left(n^{\log_3 k + \varepsilon}\right) \Rightarrow T(n) = \Theta(n^2)$

In this case, $\log_3 k < 2 \Rightarrow k < 9$

And, $T(n) = o(n^{\lg 7})$

Case 2: $n^2 = \Theta\left(n^{\log_3 k}\right) \Rightarrow T(n) = \Theta(n^2 \lg n)$

In this case, $\log_3 k = 2 \Rightarrow k = 9$

And, $T(n) = o(n^{\lg 7})$

Case 1: $n^2 = O(n^{\log_3 k - \varepsilon}) \Rightarrow T(n) = \Theta(n^{\log_3 k})$

In this case, $\log_3 k > 2 \Rightarrow k > 9$

And, $T(n) = o(n^{\lg 7})$ if $\log_3 k < \lg 7 \Rightarrow k < 3^{\lg 7}$

Since $\lg 7 \approx 2.8073549 \Rightarrow 3^{\lg 7}$ is not an integer

Thus, $k < 3^{\lg 7} \Rightarrow k \leq \lfloor 3^{\lg 7} \rfloor$

In conclusion, the divide-and-conquer algorithm beats Strassen's algorithm if $1 \leq k \leq \lfloor 3^{\lg 7} \rfloor$. (In fact, $\lfloor 3^{\lg 7} \rfloor = 21$)

4 [Ex 4.3-7, HW#2]

5 [Ch07 Lecture note, pp14~17]

6 [Ch09 Lecture note, pp4~5]

7 a) Observe that

- The height of the left subheap is $k-1$.
- The height of the right subheap is $k-1$ or $k-2$.
- After building the two subheaps, it takes at most k sift-down steps to build a heap of height k.

Therefore,

$T'(k) \le 2T'(k-1) + k$

We show that $T'(k) = O(2^k)$ by constructive induction.

Wanted: $T'(k) \le c2^k - dk$

Inductive step

$$
\begin{aligned}
T'(k) &\le 2T'(k-1) + k \\
&\le 2(c2^{k-1} - d(k-1)) + k \\
&= c2^k - dk - (dk - k - 2d) \\
&\le c2^k - dk
\end{aligned}
$$

as long as

$dk - k - 2d \ge 0 \Rightarrow (d-1)k \ge 2d \Rightarrow k \ge 2d/(d-1)$ if $d > 1$

So, pick $k_0 = \lceil 2d/(d-1) \rceil$.

Alternative solution

Let

$a_k = 2a_{k-1} + k$ $\cdots (1)$

$a_{k-1} = 2a_{k-2} + k - 1$ $\cdots (2)$

$a_k - 3a_{k-1} + 2a_{k-2} - 1 = 0$ $\cdots (1) - (2) = (3)$

$a_{k-1} - 3a_{k-2} + 2a_{k-3} - 1 = 0$ $\cdots (4)$

$a_k - 4a_{k-1} + 5a_{k-2} - 2a_{k-3} = 0$ $\cdots (3) - (4) = (5)$

The characteristic equation of (5) is

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

Thus, the homogeneous solution of (5) is

$a_k = c_1 2^k + c_2 1^k + c_3 k 1^k = c_1 2^k + c_2 + c_3 k = \Theta(2^k)$

Since $T'(k) \le a_k$, it follows that $T'(k) = O(2^k)$

b) [Chap06 Lecture note, p2]

The height of an n-node heap is $\lfloor \lg n \rfloor$.

Thus,

$T(n) = T'(\lfloor \lg n \rfloor) = O\left(2^{\lfloor \lg n \rfloor}\right) = O\left(2^{\lg n}\right) = O(n)$

8 a) They are $\binom{2n}{n}$ possible pairs of sorted lists. Any comparison-based merging algorithm has to determine the placement of the elements of one list among the elements of the other. Each pair of lists induces a unique placement. Thus, the algorithm must distinguish all possible placements and so its decision tree must have at least $\binom{2n}{n}$ leaves.

b) Let

$h =$ height of the decision tree $=$ number of comparisons in the worst case

Then,

$$\binom{2n}{n} \leq \#\text{ of leaves} \leq 2^h \Rightarrow h \geq \lg\binom{2n}{n}$$

Since

$$\sum_{i=0}^{2n}\binom{2n}{i} = 2^{2n}$$

and $\binom{2n}{n}$ is the maximum among the $2n+1$ terms, we have

$$\binom{2n}{n} \geq 2^{2n}/(2n+1)$$

Thus,

$$\begin{aligned} h &\geq \lg\binom{2n}{n} \\ &\geq \lg(2^{2n}/(2n+1)) \\ &= \lg 2^{2n} - \lg(2n+1) \\ &= 2n - o(n) \end{aligned}$$

9 a) Let's say that the PARTITION procedure partitions the array into

$\leq$ pivot	pivot	$>$ pivot

Here is one way to sort the n integers:

for $i = 1$ **to** 9 **do**

 ▷ At this point, we have

$= 1$	$\cdots$	$= i - 1$	$> i - 1$

 use i as the pivot to partition the $> i - 1$ region

 ▷ At this point, we have

$= 1$	$\cdots$	$= i - 1$	$= i$	$> i$

Observe that at the end of the loop, $i = 10$ implies that the $> i - 1$ region contains integers > 9, i.e. integers $= 10$.

Clearly, the algorithm runs in time

$$\Theta(n) + \sum_{i=2}^{9} O(n) = \Theta(n) + O\left(\sum_{i=2}^{9} n\right) = \Theta(n) + O(8n) = \Theta(n)$$

[Alternative partition order]

$i = 9,8,7,6,5,4,3,2,1$ or $i = 2,4,6,8,1,3,5,7,9$, etc.

b) The lower-bound is for ***general-purpose*** comparison-sorting algorithms, i.e. those that make no assumption on the input.