COSC3101.03 Fall 1997-98

Solutions to Mid-Term Test

$\boxed{1}$

a. **True.** $f(n) \le cg(n)$ implies $lgf(n) \le lgc + lgg(n) \le (lgc+1)lgg(n)$, since $lgg(n) \ge 1$.

b. **False.** For example, $2n+2 = O(n+2)$, but $2^{2n+2} = \Theta(4^n) \ne O(2^{n+2})$.

c. **True.** $n^5\ lg\ n = O(n^6 / lg\ n)$.

d. **True.** Compare the logarithm of each term. The largest term is $2^{n\ lg\ n} = n^n$.

e. **True.** Both are $\Theta(1)$. See chapter 3 of [CLR].

f. **True.** For a similar recurrence, see pages 59-61 of [CLR].

g. **False.** It is $\Theta(n^2)$.

h. **True.** See your lecture notes or section 10.2 of [CLR].

i. **False.** It is $n-1$.

j. **False.** It is $\Theta(n\ lg\ n)$. See your lecture notes or section 35.4 of [CLR].

$\boxed{2}$

(a) Here the recurrence is of the form $T(n) = aT(n/b) + \Theta(n^d)$. Apply the Master Theorem: since $\log_b a = \log_4 8 = 3/2 > d = 1/2$, the solution is $T(n) = \Theta(n^{\log_b a}) = \Theta(n\sqrt{n})$.

(b) Similar to part (a), but now $\log_b a = d = 3/2$. Thus, $T(n) = \Theta(n^d\ lg\ n) = \Theta(n\sqrt{n}\ lg\ n)$.

(c) $f(n)$ is not a polynomial, so we cannot apply the Master Theorem. Instead, we start with the general solution formula:

$$T(n) = \Theta(\ n^{\log_b a}\ [1 + \sum_{i=1}^{k} f(b^i)/a^i\ ]\ ) \quad , \quad \text{where } k = \lceil \log_b n \rceil = \Theta(lg\ n)$$

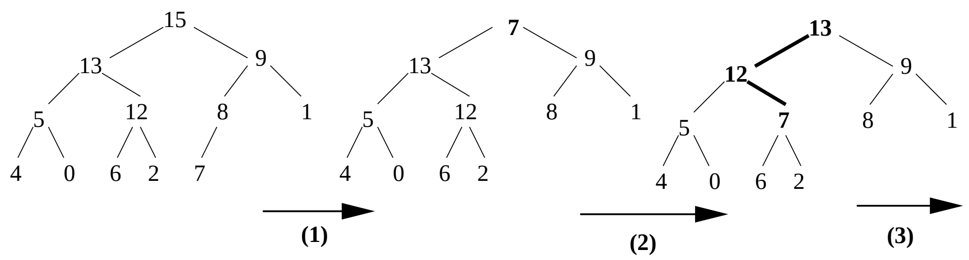
$$= \Theta(\ n\sqrt{n}\ [1 + \sum_{i=1}^{k} \frac{8^i/lg4^i}{8^i}\ ]\ ) \ = \ \Theta(\ n\sqrt{n}\ [1 + \sum_{i=1}^{k} 1/2i\ ]\ )$$

$$= \Theta(n\sqrt{n}[1 + H_k/2]\ ) \ = \ \Theta(n\sqrt{n}\ lg\ k)$$

$$= \Theta(n\sqrt{n}\ lg\ lg\ n)\ .$$

$\boxed{3}$ This question concerns **max-heaps**.

(a) See the procedure on page 150 of [CLR]. The figure below gives the illustration:
(1) record max $\leftarrow$ 15 from the root; move the key of the last node, 7, to the root; and decrement heap-size by 1.
(2) *Heapify*$(A, 1)$. This will push 7 down the largest-child path as highlighted.
(3) return the deleted maximum key 15.



(b) Here is the algorithm performed on a **max-heap** A:

```
procedure Heap-Increase-Key $(A, i, k)$
    if $A[i] \geq k$ then return
    while $i > 1$ and $A[\lfloor i/2 \rfloor] < k$ do $A[i] \leftarrow A[\lfloor i/2 \rfloor]$; $i \leftarrow \lfloor i/2 \rfloor$ end
    $A[i] \leftarrow k$
end
```

This is essentially *Up-Heap* as explained in class, and it takes $O(lg\ n)$ time on a heap of size n.

$\boxed{4}$ (See also Exercise 10.3-8, page 192 of [CLR].)

Observation: The median (i.e., 5th smallest) of the 10 elements is:

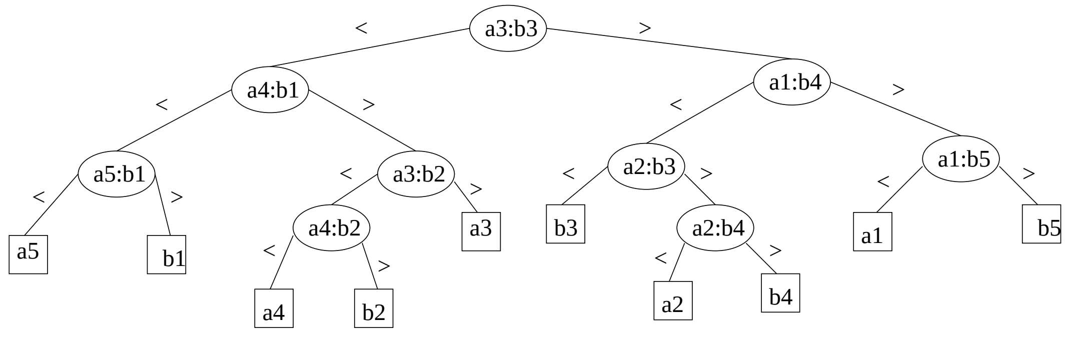
(i) a_i, if and only if $b_{5-i} < a_i < b_{6-i}$,

(ii) b_i, if and only if $a_{5-i} < b_i < a_{6-i}$.

(We assumed the notation $a_0 = b_0 = -\infty$ and $a_6 = b_6 = +\infty$.) The proof is left to the reader. Check these conditions by looking at each leaf of the decision tree and comparisons made at its ancestors.

The decision tree below iteratively compares the two medians of the sorted a-list and the b-list in constant time and discards half of the irrelevant elements. For instance, the first comparison is between a_3 and b_3. If, say, $a_3 < b_3$, then a_1 and a_2 are smaller than 5th smallest (and hence can be eliminated), since they are less than at least the 6 elements a_3 through a_5 and b_3 through b_5. With a similar reasoning we see that the 3 elements b_3 through b_5 are larger than the 5th smallest. So, we eliminate 5 elements, and the 5th smallest overall is the 3rd smallest among the remaining 5 elements. Repeat the same idea.

Since any one of the 10 elements could possibly be the final outcome, any such binary decision tree must have at least 10 leaves. Therefore, it will have height $h \geq \lceil lg\ 10 \rceil = 4$. Thus, the decision tree below which makes at most 4 comparisons, is worst-case optimal.



a3:b3
a4:b1
a1:b4
a5:b1
a3:b2
a2:b3
a1:b5
a4:b2
a2:b4
a5
b1
a4
b2
a3
b3
a2
b4
a1
b5
<
>
