Set 3 Homework, Analysis of Algorithms

Jay R Bolton

May 9, 2012

- p 166: 6.5-6
- p 167: 6-1,6-2
- p 178: 7.2-1, 7.2-5
- p 180: 7.3-1
- p 284: 7.4-2
- p 185: 7-2, 7-4

Chapter 6

6.5-6 Do 'exchange' in 'Heap-Increase-Key' with one assignment.

The original:

$$\begin{array}{l} HeapIncreaseKey(A, i, key): \\ \quad if\ key < A[i] \\ \quad\quad error \text{ “new key is smaller than current key”} \\ \quad A[i] = key \\ \quad while\ i > 1\ and\ A[Parent(i)] < A[i] \\ \quad\quad exchange\ A[i]\ with\ A[Parent(i)] \\ \quad I = Parent(i) \end{array}$$

With three assignments:

$$\begin{array}{l} HeapIncreaseKey(A, i, key): \\ \quad if\ key < A[i] \\ \quad\quad error \text{ “new key is smaller than current key”} \\ \quad A[i] = key \\ \quad while\ i > 1\ and\ A[Parent(i)] < A[i] \\ \quad\quad tmp = A[i] \\ \quad\quad A[i] = A[Parent(i)] \\ \quad\quad A[Parent(i)] = tmp \\ \quad\quad i = Parent(i) \end{array}$$

With one assignment:

$$
\begin{aligned}
&HeapIncreaseKey(A, i, key):\\
&\quad if\ key < A[i]\\
&\quad\quad error\ \text{“new key is smaller than current key”}\\
&\quad while\ i > 1\ and\ A[Parent(i)] < key\\
&\quad\quad A[i] = A[Parent(i)]\\
&\quad\quad i = Parent(i)\\
&\quad A[i] = key
\end{aligned}
$$

That was a real fun little puzzle.

6-1 **(a)** No. The counterexample is $[N, 1, 2, 3]$. BMH produces $[N, 3, 2, 1]$ while BMH' produces $[N, 3, 1, 2]$. Both are heaps.

(b) Max-Heap-Insert requires $\Theta(lg\ n)$ time. In Build-Max-Heap', we are looping that function $n - 1$ times. Everything else is constant, so our bound is $\Theta(n\ lg\ n)$.

6-2 **(a)** Same way, but you'd have to store or pass d and the children would be calculated with $di + 1$ through $di + d$ where 'i' is the current index.

(b) The height would be $log_d(n)$.

(c)

$$
\begin{aligned}
&\text{ExtractMax}(A)\\
&\quad if\ A.\text{heapsize} < 1\\
&\quad\quad \text{error “}heap\ underflow\text{”}\\
&\quad max = A[1]\\
&\quad A[1] = A[A.\text{heapsize}]\\
&\quad \text{MaxHeapify}(A, 1)\\
&\quad \text{return } max
\end{aligned}
$$

$$
\begin{aligned}
&\text{MaxHeapify}(A, i)\\
&\quad largest = i\\
&\quad for\ c = di + 1\ upto\ di + d\\
&\quad\quad if\ c \leq A.\text{heapsize}\ and\ A[c] > A[largest]\\
&\quad\quad\quad largest = c\\
&\quad if\ largest \neq i\\
&\quad\quad exchange\ A[i]\ with\ A[largest]\\
&\quad\quad \text{MaxHeapify}(A, largest)
\end{aligned}
$$

ExtractMax remains unchanged, but MaxHeapify must now loop d times through all subtrees. Its complexity will be $\mathcal{O}(log_b n)$

(d, e) Both Insert and IncreaseKey can be implemented the same since neither depend on the selection of children.

6-3 **(a)**

$$
\begin{array}{cccc}
2 & 3 & 4 & 5\\
8 & 9 & 12 & 14\\
16 & \infty & \infty & \infty
\end{array}
$$

(b) Y[1,1] will be the least element in the matrix (least of the least of the columns and least of the least of the rows). If Y[1,1] is infinity/null, then there is no least element.

If Y[1,1] contains a non-null element then that means we have a least element. We have at least one element in that case, where m and n are 1.

Chapter 7

7.2-1 Prove: $T(n) = T(n-1) + \Theta(1)$ has complexity $\Theta(n^2)$.

$$\begin{aligned}
\text{Inductive Hypothesis: } T(n) &\leq c \cdot n^2 \\
\text{Also: } T(n) &\geq c \cdot n^2 \\
\text{Induction:} & \\
T(n) &\leq b(n-1)^2 + n^2 \text{ for some constant b} \\
&\leq bn^2 + n^2 \\
&= (b+1)n^2 \\
&\leq c \cdot n^2 \\
T(n) &\geq b(n-1)^2 + n^2 \text{ for some constant b} \\
&\geq n^2 \\
&= 1 \cdot n^2 \\
&= c \cdot n^2
\end{aligned}$$

7.2-5 For the minimum depth, our recurrence is $T(n) = 2T(n/2) + n$. The proportion will be one-half to one-half, which is our best case.

For the maximum depth, our recurrence is $T(n) = 2T(n-1) + n$. The proportion in this case is $\frac{n-1}{n}$ to $\frac{1}{n}$. This is the worst case.

The height of the minimum depth is $lg_2 n$ and the height of the maximum depth is n.

7.3-1 Because the worst case may be asymptomatic and the randomized version closer to average.

7.4-2 By induction with hypothesis: $T(n) \geq c \cdot n\ lg\ n$

The best-case recurrence is $T(n) = 2T(n/2) + n$, where each subproblem is partitioned evenly in two.

$$\begin{aligned}
T(n) &\geq 2(n/2)lg(n/2) + dn \\
&= n\ lg(n/2) + dn \\
&= n\ lg\ n - n\ lg\ 2 + dn \\
&= n\ lg\ n - n + dn \\
&\geq c \cdot n\ lg\ n
\end{aligned}$$

7-2

7-4