Set 3 Homework, Analysis of Algorithms

Jay R Bolton

May 8, 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:

```
HeapIncreaseKey(A, i, key) :
  if key < A[i]
    error "new key is smaller than current key"
  while i > 1 and A[Parent(i)] < key
    A[i] = A[Parent(i)]
    i = Parent(i)
  A[i] = key
```

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)

```
ExtractMax(A)
  if A.heapsize < 1
    error "heap underflow"
  max = A[1]
  A[1] = A[A.heapsize]
  MaxHeapify(A, 1)
  return max
```

```
MaxHeapify(A, i)
  largest = i
  for c = di + 1 upto di + d
    if c ≤ A.heapsize and A[c] > A[largest]
      largest = c
  if largest ≠ i
    exchange A[i] with A[largest]
    MaxHeapify(A, largest)
```

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)**

2	3	4
5	8	9
12	14	16

It seems like there might be quite a few possibilites to make that matrix.

(b)

Chapter 7

1.