BuildMaxHeap analysis

(a) BuildMaxHeap(H[1 .. n])
// Put H[1..n] into MaxHeap order
for j := n/2 downto 1
  MaxHeapify(H, j) Θ(height of j)

(b) Less efficient
Initialize empty binary heap H'
for j := 1 to n do
  Θ(lg(H.size)) Insert(H', H[j])
return H'

Total time for B is

$$\sum_{j=1}^{n} \Theta(\lg j) = \Theta\left(\sum_{j=1}^{n} \lg j\right)$$

$$\sum_{j=1}^{n} \lg j \le \sum_{j=1}^{n} \lg n = n \lg n$$

$$\sum_{j=1}^{n} \lg j \ge \sum_{j=\frac{n}{2}}^{n} \lg j \ge \sum_{j=\frac{n}{2}}^{n} \lg\left(\frac{n}{2}\right)$$

$$= \sum_{j=\frac{n}{2}}^{n} (\lg n - 1) = \frac{n}{2} \lg n - \frac{n}{2} = \Omega(n \lg n)$$

∴ Time for (B) is $\Theta(n \lg n)$.

Analysis of (A):
assume (for the U.B.) that all leaves on one level.

Time for (A) is

$$\sum_{\text{height}=1}^{\lg n} \underbrace{\Theta(\text{height})}_{\substack{\text{time for} \\ \text{1 call to maxHeapify} \\ \text{on node of height}}} \left(\frac{n}{2^{\text{height}}}\right)$$

$$= \Theta\left(\sum_{h=1}^{\lg n} h \cdot \frac{n}{2^h}\right) = \Theta\left(n \sum_{h=1}^{\lg n} \frac{h}{2^h}\right)$$

$$\le \Theta\left(n \underbrace{\sum_{h=1}^{\infty} \frac{h}{2^h}}_{?\,=2}\right) = O(n)$$

Let $S := \sum_{h=1}^{\infty} \frac{h}{2^h}$

$$2S = \sum_{h=1}^{\infty} \frac{h}{2^{h-1}} \quad [j := h-1]$$

$$= \sum_{j=0}^{\infty} \frac{j+1}{2^j}$$

$$= \sum_{j=0}^{\infty} \frac{j}{2^j} + \underbrace{\sum_{j=0}^{\infty} \frac{1}{2^j}}_{\frac{1}{1-\frac{1}{2}} = 2} \quad (r = \tfrac{1}{2})$$

$$= \sum_{j=1}^{\infty} \frac{j}{2^j} + 2$$

$$2S = S + 2$$

$$\boxed{S = 2}$$

Lower Bound: is $\Omega(n)$ because algo loop iterates $\frac{n}{2}$ times.

Quicksort

QS(A[p .. q])
if q > p:
  {Choose a pivot value x (some entry of A)}
  m := Partition(A, x)
  // A[m] = x
  // A[j] ≤ x for p ≤ j ≤ m
  // A[j] ≥ x for m ≤ j ≤ q

// A: [≤x | x | ≥x] (p, m, q)

QS(A[p .. (m-1)])
QS(A[(m+1) .. q])

Worst case: pivot is an extreme value
time:

$$T(n) = \underbrace{\Theta(n)}_{\substack{\text{time for} \\ \text{partition}}} + T(n-1)$$

$$T(n) = \Theta(n^2) \quad \text{(worst case)}$$

[Averaging over all inputs of a given size n.
Assume some probability distribution μ_n on inputs of size $\le n$
Average time $T(n)$

$$= \sum_{\substack{\text{inputs } x \\ \text{of size } n}} \underbrace{T(x)}_{\substack{\text{running} \\ \text{time on } x}} \Pr_{\mu_n}[x \text{ is the input}]$$

If μ_n is uniform dist, then

$$T_{\text{avg}}(n) = \Theta(n \lg n) \quad \text{[Book]}$$

But μ_n may be non-uniform, and not even known.

RQS(A[p..q])

if p < q:

Choose pivot x uniformly at random from A[p], ..., A[q]

m = Partition on x

RQS(A[p..(m-1)])

RQS(A[(m+1)..q])

E(n) := expected time to run RQS on an input of size n (n = q-p+1)

[assume no duplicate items]

$= \sum_{\text{all random choices } C}$ T(given C) · Pr[C]

$= \sum_{\text{all random choices made at the top level}}$ (T(top level) + expected time of the recursive calls) Pr[choice

$= \sum_{l=0}^{n-1} Pr[l]\left(\overbrace{\Theta(n)}^{\text{time at top level}} + E(l) + E(n-1-l)\right)$

{l = length of left subtist} $= \sum_{l=0}^{n-1} \frac{1}{n}(an + E(l) + E(n-$

$= \sum_{l=0}^{n-1} \frac{2}{n}(an + E(l)) = E(n)$

$= 2a + \frac{2}{n}\sum_{l=0}^{n-1} E(l) = E(n)$

Claim: $E(n) = O(n \lg n)$

Proof by substitution method

Assume $E(l) \le c\, l \lg l$ for $l < n$.

$E(n) = 2a + \frac{2}{n}\sum_{l=0}^{n-1} E(l)$

$\le 2a + \frac{2}{n}\sum_{l=0}^{n-1} c\, l \lg l$

$= 2a + \frac{2c}{n \ln 2}\sum_{l=1}^{n-1} l \ln l$

$\le 2a + \frac{2c}{n \ln 2}\int_1^n \underbrace{x \ln x}_{\text{monotone ascending in } x}\, dx$

$= 2a + \frac{2c}{n \ln 2}\left(\frac{x^2 \ln x}{\text{const.}} + \text{lower order terms}\right)$

$\le 2a + \frac{2c}{n \ln 2}\left(n^2 \ln n + \text{lower order terms}\right)$

$= 2a + 2c\, n \lg n + o(n \lg n)$

$= O(n \lg n)$

Matches the lower bound asymptotically.