Proof for The Complexity of Building A Heap

Hu Ding

Department of Computer Science and Engineering
Michigan State University
huding@msu.edu

Theorem 1. *Let A be an array having n integers, then the running time of BuildMaxHeap(A) is* $\Theta(n)$.

Proof. We denote the total running time as T. We know that the algorithm of BuildMaxHeap calls the function *MaxHeapify(A, i)* for $i = n/2$ down to 1. So we can directly obtain $T = \Omega(n)$.

It is also easy to know that the running time of MaxHeapify(A, i) is bounded by $O(h_i)$, where h_i denotes the height of the sub-tree rooted at the node i. Thus, we have

$$T = O(\sum_{i=n/2}^{1} h_i). \tag{1}$$

For convenience, we denote $k = \log n$ (the base is 2). So we have one node with $h_i = k$ (i.e., the root), two nodes with $h_i = k-1$, four nodes with $h_i = k-2$, and so on. In total, we rewrite (1) as:

$$T = O(\sum_{j=0}^{k-1} 2^j \times (k-j)). \tag{2}$$

In order to calculate $\sum_{j=0}^{k-1} 2^j \times (k-j)$, we express the sum as

$$A = k + 2 * (k-1) + 2^2 * (k-2) + \cdots + 2^{k-2} * 2 + 2^{k-1} * 1, \tag{3}$$

$$2A = 2 * k + 2^2 * (k-1) + \cdots + 2^{k-1} * 2 + 2^k * 1. \tag{4}$$

(4) is directly obtained by doubling both sides of (3). Then we have

$$A = (4) - (3) = -k + 2 + 2^2 + \cdots + 2^{k-1} + 2^k = \Theta(n). \tag{5}$$

Back to (2), we have $T = O(A) = O(n)$. Since we have $T = \Omega(n)$ before, we know $T = \Theta(n)$. □

Remark 1. We have another solution for proving $T = O(n)$. It is easy to know $h_i \leq \log \frac{n}{i} + 1$. Then

$$\begin{aligned} A &\leq \sum_{i=n/2}^{1} (\log \frac{n}{i} + 1) \\ &= \frac{n}{2} \log n - \sum_{i=1}^{n/2} \log i + n/2. \end{aligned} \tag{6}$$

For $\sum_{i=1}^{n/2} \log i$, we can use integration method, i.e.,

$$\sum_{i=1}^{n/2} \log i \geq \int_{x=1}^{n/2} \log x dx = \frac{n}{2} \log \frac{n}{2} - \frac{n/2 - 1}{\ln 2}. \tag{7}$$

Plug (7) into the right hand of (6), we have $A = O(n)$. Thus, $T = O(n)$.