1. Build the Heap

Please describe a linear time algorithm to build a heap from a given array of size n and prove its time complexity to be O(n).

Given array size of n, the height of a heap is $\log n$. From the book, we know that we need $O(\log n)$ of time to do Max-Heapify on a node, and it takes $O(n\log n)$ time on n nodes.

But this is the upper bound of time to build a heap. The time we need to do Max-Heapify differs by the height of the node. The height of a n-element heap is $\log n$, and at most $\lceil n/2^{h+1} \rceil$ nodes of any height h.

So the prove is that,

$$\sum_{h=0}^{\log n} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\log n} \left\lceil \frac{h}{2^h} \right\rceil\right)$$

$$\because \sum_{h=0}^{\infty} \frac{h}{2^h} = x = 0 + \frac{1}{2} + \frac{2}{4} + \frac{3}{8} + \cdots$$

$$-) \quad \frac{1}{2}x = \frac{1}{4} + \frac{2}{8} + \cdots$$

$$\frac{1}{2}x = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots$$

$$x = 1 + \frac{1}{2} + \frac{1}{4} + \cdots$$

$$\therefore \sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{\frac{1}{2}}{(1-\frac{1}{2})^2} = 2$$

$$\Rightarrow O\left(n \sum_{h=0}^{\log n} \left\lceil \frac{h}{2^h} \right\rceil\right) = O\left(n \sum_{h=0}^{\infty} \left\lceil \frac{h}{2^h} \right\rceil\right) = O(2n) = O(n)$$

So the time to build a heap has time complexity $O(n)$.

2. Selection problem

Given an unsorted array of size n, we can find the kth smallest element in O(n) time by partitioning them into ⌈n/5⌉ groups.

How about ⌈n/3⌉ or ⌈n/7⌉? Why or why not? Give your reasoning.

<case 1> partition into $\lceil n/7 \rceil$ groups

There are $n/7$ groups with at least $4(\frac{1}{2}(\frac{n}{7}))$ elements that are less than or equal to the median of the medians. Thus, the larger subset after partitioning has at most $n - 2n/7 = 5n/7$ elements. The running time is $T(n) = T(\frac{n}{7}) + T(\frac{5}{7}n) + O(n)$

Assuming $T(k) \le ck$ for $k < n$

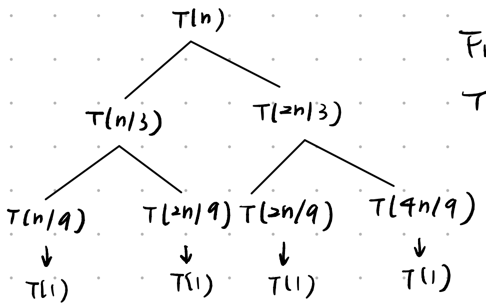
$$\begin{aligned} T(k) &\le cn/7 + 5cn/7 + c_1 n \\ &= 6cn/7 + c_1 n \\ &= cn + (c_1 - \tfrac{1}{7}c)n \\ &\le cn = O(n) \end{aligned}$$

$\Rightarrow$ The algorithm will work in linear time if they are divided into groups of 7.

<case 2> partition into $\lceil n/3 \rceil$ groups

There are $n/3$ groups with at least $2(\frac{1}{2}(\frac{n}{3}))$ elements that are less than or equal to the median of the medians. Thus, the largest subset after partitioning has at most $n - \frac{n}{3} = \frac{2}{3}n$ elements. The running time is $T(n) = T(\frac{n}{3}) + T(\frac{2}{3}n) + O(n)$

$$T(n) = T(\tfrac{n}{3}) + T(\tfrac{2}{3}n) + O(n)$$



From the recursion tree we know that

$T(n) \geq n \log_3 n \Rightarrow T(n) = \Omega(n \log n)$

$\Rightarrow$ The algorithm will not work in linear time if they are divided into groups of 3.