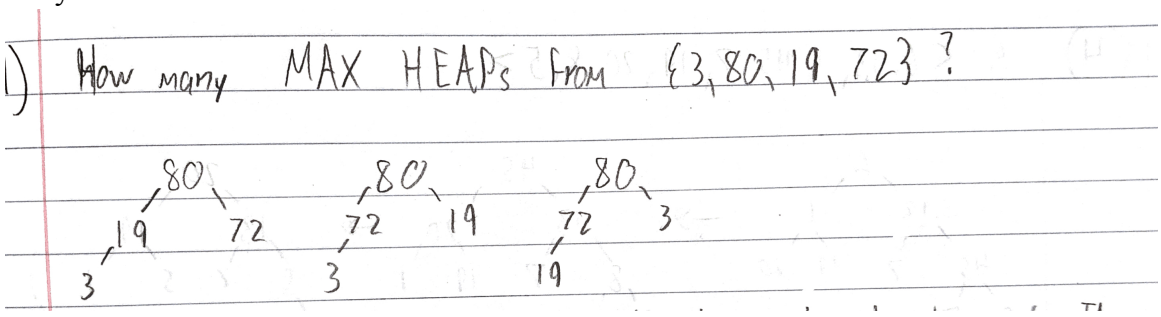
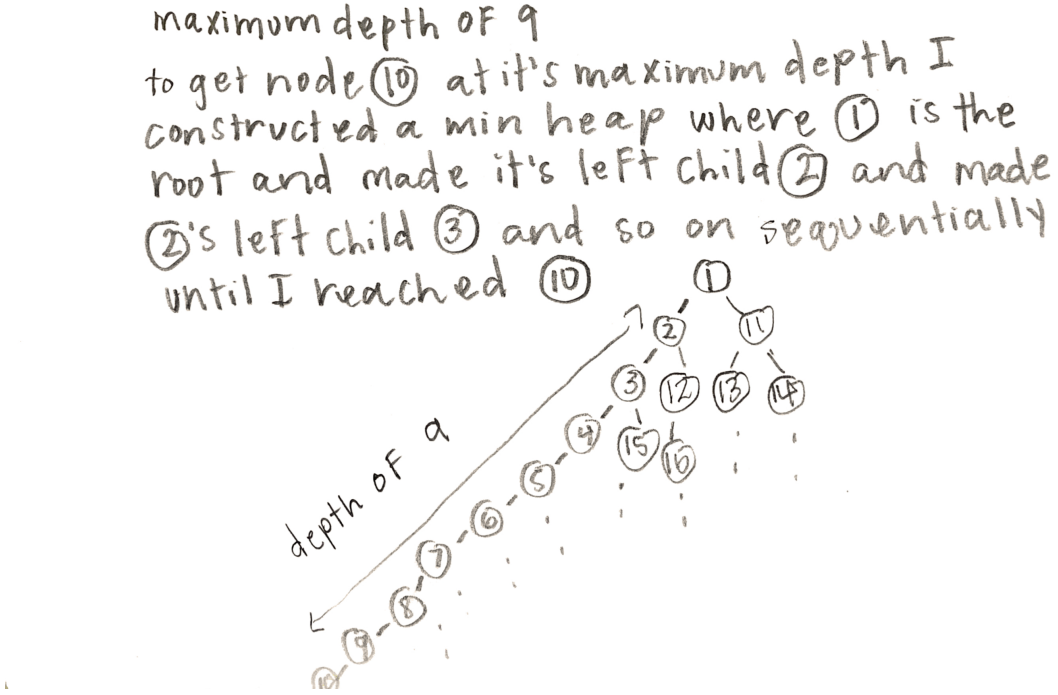
1. By Ben Albert



2. By Duyen Tran

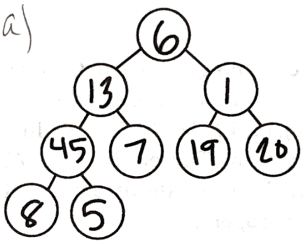


3. By Karamel Quitayen

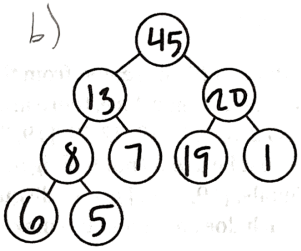
The minimum value for an element in a binary max-heap would have to be a leaf – a node having no children. The algorithm would have to check all the leaves and for a heap of size n, the number of leaves would be $n/2$. Therefore, the tight bound would be $O\left(\frac{n}{2}\right) = O(n)$.

4. By Karamel Quitayen

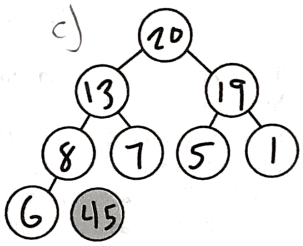
a)



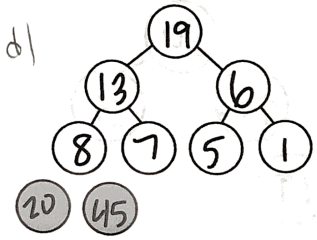
b)



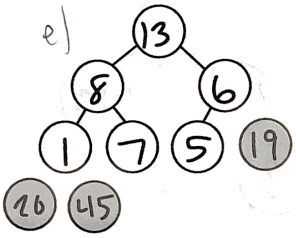
c)



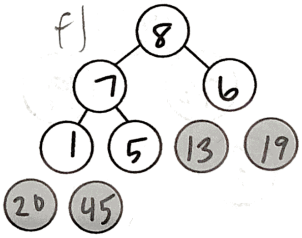
d)



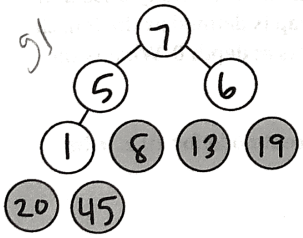
e)



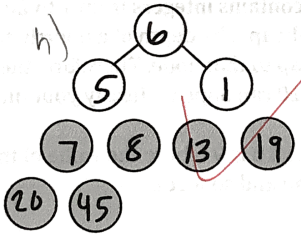
f)



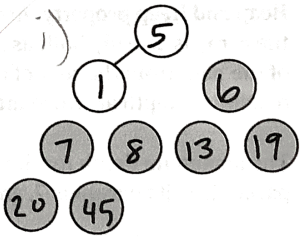
g)



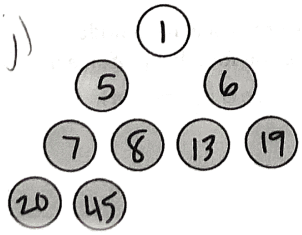
h)



i)



j)



k)

A	1	5	6	7	8	13	19	20	45

5. By Bonnie Liu

a. root : A[1]
Level1 : A[2] to A[d+1]
Level2 : A[d+2] to A[d^2+d+1]
:

From previous step: the parent (i), the child(i,j) is d(i−1)+j+1 when 1≤j≤d.

b.
For d-ary heap, each node has d children, then the height of a d-ary heap with n node is $\log_d n$.

c.
It will be almost the same as the one in binary heaps. Instead of just compare the parent node with its two children, parent node have to compare with its all d children. Because we need recursively compare each node with its d children to the depth of tree. The time is $O(d \log_d n)$
T(n) for Extract-Max : $T(n) = T(1) + T(\text{Max-Heapify}) = O(d \log_d n)$

d. The running time for insert in a d-ary max-heap is the same as in binary heap. The different is the running for Heap-increase-key. For the d-ary, time in upward path for node i in len th $O(\log_d n)$ in an n element heap. The method will be traverses full height of tree.
so, T(n) is $O(\log_d n)$

c and d: Big-O notation is fine.