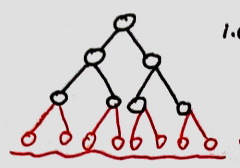
Q4.2

2021年3月8日 星期一　　下午10:18

(ii) w.l.o.g., we only consider the cases where the binary tree looks like :



← newly added k elements

i.e. Every layer of Tree is fully filled. AND the added nodes fill the next layer.

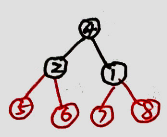
If not, that's OK. we can increase the number of added nodes to get the case looks like above.

In the worst case, we need to append $k+2k=3k$ nodes. But as long as it's a multiple of k, that doesn't matter.

☆ First Part :

⇒ Again, consider the worst case where all values of newly appended tree nodes are larger than the current node at root.

e.g.



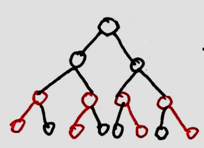
← third last layer

← second last layer

← last layer

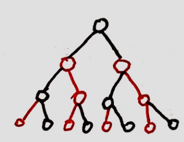
For the last layer & second last layer, we need to swap $\frac{k}{2}$ times.

⇒

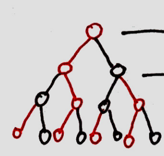


⇒ For the second and third last layers, we need to swap $\frac{k}{4}$ ~~tag~~ times.

⇒ Now we get :



⇒ Apply this method for the remaining layers, finally we will get below :



→ first layer

→ second layer

⋮

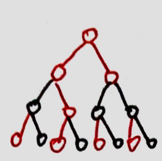
⇒ All number of swap times are $N_1 = \frac{k}{2} + \frac{k}{4} + \cdots + \frac{k}{2^b}$ $(b = \lceil \log n \rceil)$

☆ Second Part :

⇒ Now, consider backwards :

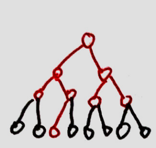
Starting from second layer, we see that at most 2 nodes need to be sift down one layer,

i.e. We need $\frac{k}{2^{b-1}}$ swaps to get :



⇒ Again, for the third layer, we need at most 4 swaps. i.e. $\frac{k}{2^{b-2}}$ swaps.

⇒ Apply this method for the remaining layers, we get :



⇒ Sorting Finished

Total number of swaps :

$$N_2 = \frac{k}{2} + \frac{k}{4} + \cdots + \frac{k}{2^{b-1}} \quad b = \lceil \log n \rceil$$

Combine the two parts :

Total number of steps : $N_1 + N_2 = \left(\frac{k}{2} + \frac{k}{4} + \cdots + \frac{k}{2^b}\right) \times 2 - \frac{k}{2^b}$

$$\Rightarrow N_1 + N_2 < 2k\left(\frac{1}{2} + \frac{1}{4} + \cdots\right)$$

$$= 2k \cdot \frac{1}{2} \cdot \left(1 + \frac{1}{2} + \cdots\right)$$

$$= 2k$$

Add k steps for inserting k nodes, Total number of steps is $3k$, which is in $O(k)$.

Note : If n is very large, i.e. $n \gg k$, inserting k elements is close to insert a small amount of elements, we need $c \cdot \log n$ steps, (c is small) which is in $O(\log n)$.

So, this algorithm is in $O(k + \log n)$.