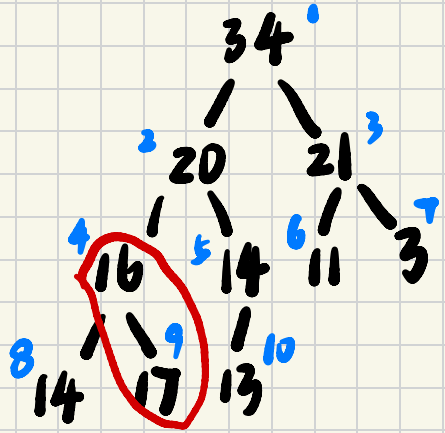
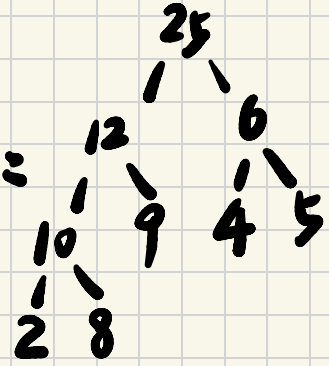
Question 4.1



Thus, it's Not a max-heap.

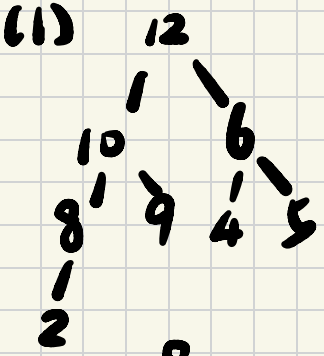
Question 4.2

After Buid-Max-Heap =

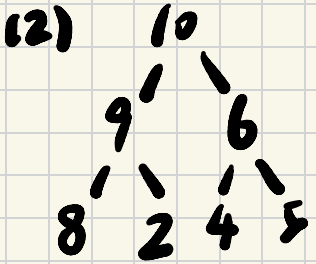


Then

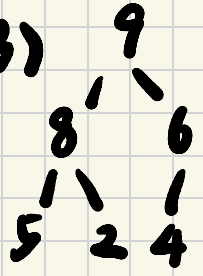
(1)



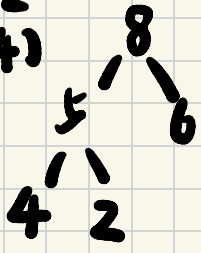
(2)



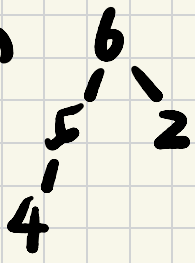
(3)



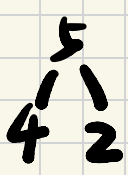
(4)



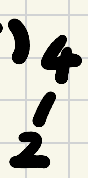
(5)



(6)



(7)



(8) 4

Question 4.3

(1) MAX-HEAPIFY(A, i)

```
1: While i ≤ A.heap-size
2:     l = Left(i)
3:     r = Right(i)
4:     if l ≤ A.heap-size and A[l] > A[i] then
5:         largest = l
6:     else
7:         largest = i
8:     if r ≤ A.heap-size and A[r] > A[largest] then
9:         largest = r
10:    if largest ≠ i then
11.        exchange A[i] with A[largest]
12.        i = largest
13.    else
14.        break
```

(2) Loop invarient: Every time it enters into a new while-loop, the subtrees Left(i) and Right(i) are max-heaps, but max-heap property might be violated in root of subtree at i.

Initialization: Before the first while loop, the subtree rooted at i might violate the max-heap property, but subtrees Left(i) and Right(i) are max-heaps because Max-Heapify is called only when the node's children are both max-heaps.

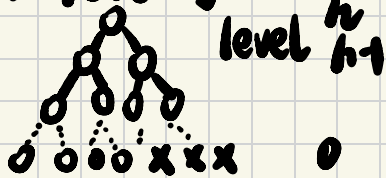
Maintence: Assume when it enters into k th while-loop, the loop invarient holds. If A[i] is larger than both children during (k-1) th while-loop, we won't have k th while-loop.

If A[i] is smaller than at least one child, which we define as A[largest], then we exchange A[i] with A[largest]. The subtree of the other child is still max-heap. And we let i = largest and enter (k+1)th while-loop. The subtrees of both node i's children are max-heap

Termination: The loop terminates when no node is smaller than its children or i > A.heap-size, ensuring the whole heap is a max-heap.

Question 4.4

(1) Consider that the maximum number of elements in a subtree when the left subtree has the last level full and the right subtree,



The total number of nodes of a tree like this: $\sum_{i=0}^{h} 2^i - \frac{1}{2} \times 2^h$

$= 2^{h+1} - 1 - 2^{h-1} = 3 \cdot 2^{h-1} - 1$

Left child of the root of n-node heap have most nodes: $\sum_{i=0}^{h-1} 2^i - 1 = 2^h - 1$

$\frac{2}{3} n = \frac{2}{3}(3 \cdot 2^{h-1} - 1) 2^h - \frac{2}{3} \geq 2^h - 1$

The proposition holds.

(2) $a=1$ $b=\frac{3}{2}$ $f(n) = \Theta(1)$

$n^{\log_b a} = n^0 = 1$

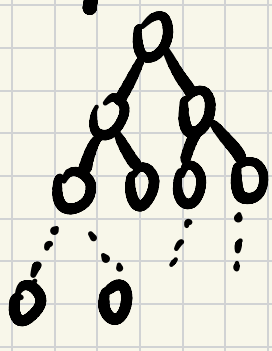
let $k=0$ $f(n) = \Theta(n^0 \lg^0 n)$

watershed and driving function grows asymptotically nearly at the same rate.

Then. $T(n) = \Theta(\lg n)$

Question 4.5

At the begining of BUILD-MAX-HEAP, every node is smaller than its children.



Every node in k-th level will take $\Omega(1)$ runtime and another k-1 iterations one by one.

Then, the runtime of MAX-HEAPIFY for every node is $\Theta(\log n)$, and total time is $\Omega(\frac{n}{2} \cdot \log n)$.

Then for HEAPSORT, the total runtime is $\Theta(n-1) \cdot \Theta(\log n)$

Thus, $\Omega(\frac{n}{2} \cdot \log n) + \Theta(n-1) \cdot \Theta(\log n)$

$= \Omega(n \log n)$