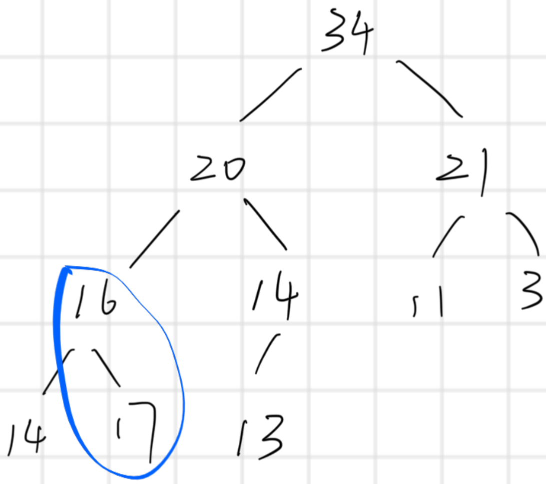
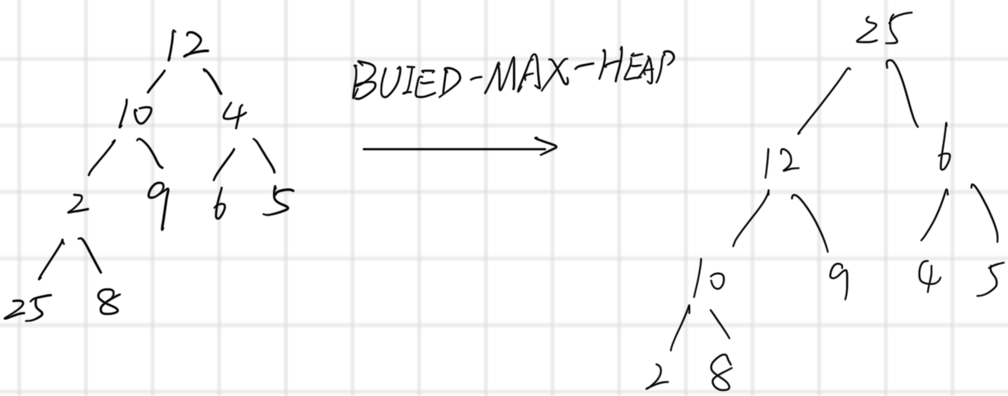
Exercise 4 12310401 王予恒

4.1

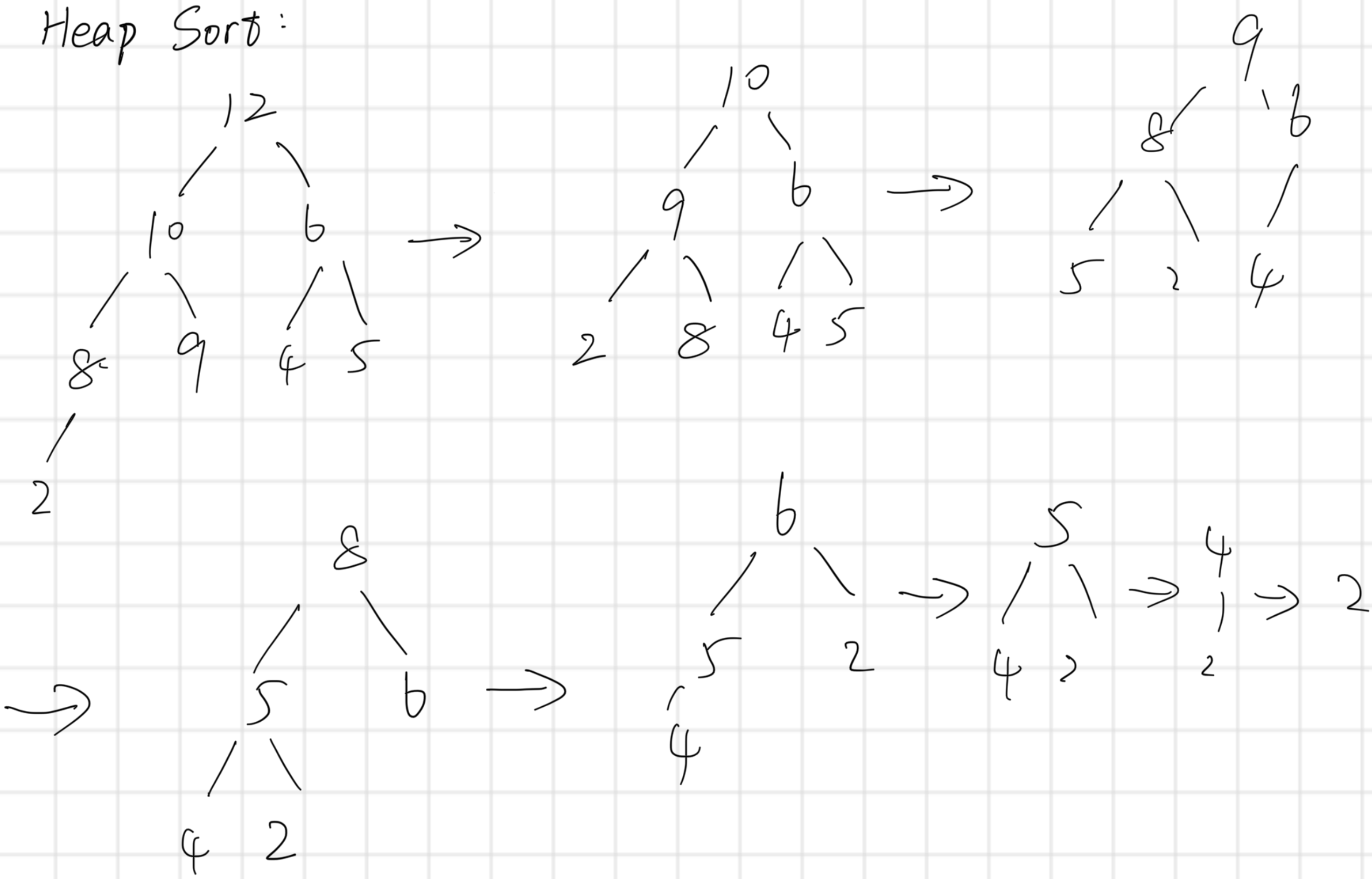


Not a Max-Heap

4.2



Heap Sort:



4.3 1. MAX-HEAPIPY(A, i):

```
while i <= A.size
    left = 2*i
    right = 2*i+1
    largest = i
    if left <= n and A[left] > A[largest]:
        largest = left
    if right <= n and A[right] > A[largest]:
        largest = right
    if largest != i:
        swap A[i] and A[largest].
        i = largest
    else:
        break.
```

2. Loop Invariant: At the start of each iteration of the while loop: the subtree rooted at i is a max-heap, except that the element A[i] may not satify the max-heap.

Initialization: Because we do the Build-Max-Heap before the Max-Heapify, and just change the root i, Thus, the subtree didn't change the max-heap properties for no element has changed.

Maintanance: In each iteration of the loop: we compare A[i] with its children A[left] and A[right]. And swap with the larger child, this ensures that the max-heap property holds between the parent and its children at this level. After swap, i = largest, moving down to the subtree of a the new i and still not change the elements of i's subtree. So the subtree of i still max-heap

Termination: The loop terminates when A[i] larger than its children, or i exceeds the number of nodes. Since the subtree of i is a max-heap except root i, so the subtree of i including i, restored the max-heap.

4.4 1. since the left nodes sub-tree $\geq$ right sub-tree. Besides the parent's subtree must bigger than the children's subtree. So the maximum number of elements in a subtree happens when the left subtree has the last level full and the right tree has the last level empty.

Suppose the heap has k level.

$$n = \sum_{i=0}^{k-1} 2^i + 2^{k-1} = 3 \cdot 2^{k-1} - 1$$

the element of left subtree: $m = \sum_{i=0}^{k-1} = 2^k - 1$

$$\therefore \frac{m}{n} = \frac{2^k - 1}{3 \cdot 2^{k-1} - 1} = \frac{2}{3} - \frac{\frac{1}{3}}{3 \cdot 2^{k-1} - 1}$$

$$\because 3 \cdot 2^{k-1} - 1 > 0 \qquad \lim_{k \to \infty} \frac{2^k - 1}{3 \cdot 2^{k-1} - 1} = \frac{2}{3}$$

$$\therefore \frac{m}{n} < \frac{2}{3}$$

2. $T(n) \leq T\left(\frac{n}{\frac{3}{2}}\right) + \Theta(1)$

$$n^{\log_{\frac{2}{3}} 1} = n^0 = 1$$

$$f(n) = \Theta(1) = \Theta(n^0 \lg^0 n) = \Theta(1) \quad k = 0$$

$$\therefore T(n) \leq \Theta(\log n)$$

$$\therefore T(n) = O(\log n)$$

4.5 Because the array is sorted, so every call of Max-Heapify is $\Theta(1)$, in BUID-MAX-HEAP calls $\Theta(n)$ times $\therefore \Theta(1) \cdot \Theta(n) = \Theta(n)$

In sort, the first step of change, the Max-HEAPIFY should be $\log n = \Theta(\log n)$, the following steps may be $O(\log n)$, so sum is $\Theta(\log n)$

HeapSort:

BUID-MAX-HEAP(A)	$\Theta(n)$
for i = A.length downto 2 do	$\Theta(n)$
exchange A[1] with A[i]	$\Theta(1)$
A.heap-size = A.heap-size - 1	$\Theta(1)$
Max-HEAPIFY	$\Theta(\log n)$

$\therefore \Theta(n) \cdot \Theta(\log n) = \Theta(n \log n) = \Omega(n \log n)$