Problem 1

(a)

False.

If T(N) is a linear function of N, doubling the size can only let T(N) increase by a factor of 2. If we want it to be 4, $T(N) \in O(N^2)$.

(b)

True.

The maximum number of nodes only occurs for complete binary trees.

(c)

False.

If we delete a node with two children, we need to replace it by the minimum number of the right subtree. So the order of deletion cannot be changed.

(d)

True.

It is in order, so it just takes time N.

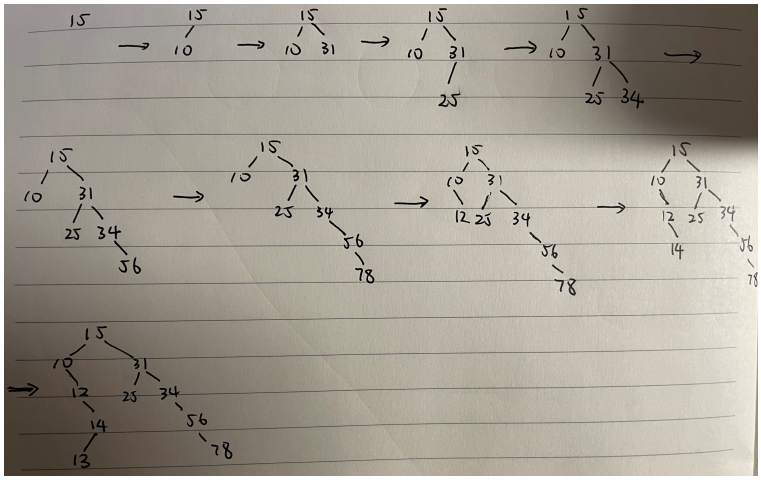
(e)

True.

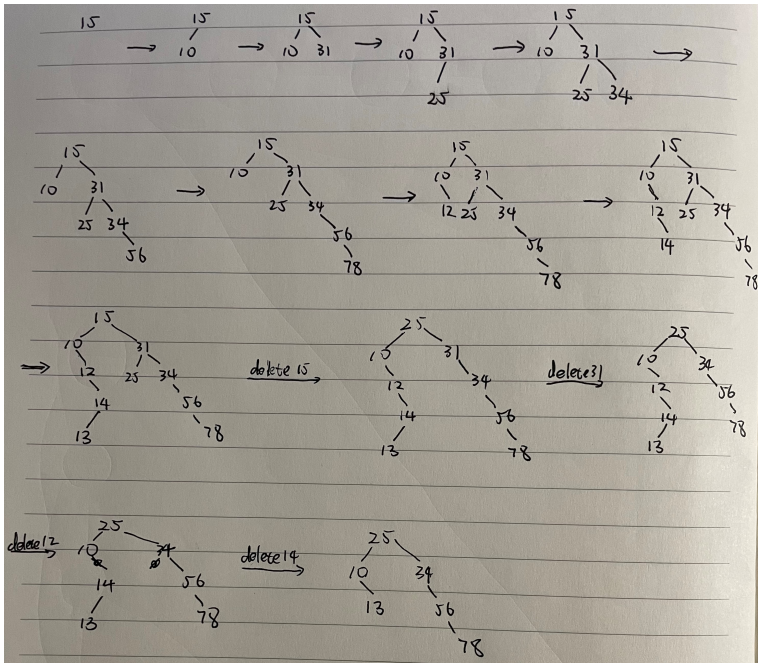
The root must be the smallest number of all. The second smallest must be a child.

Problem 2

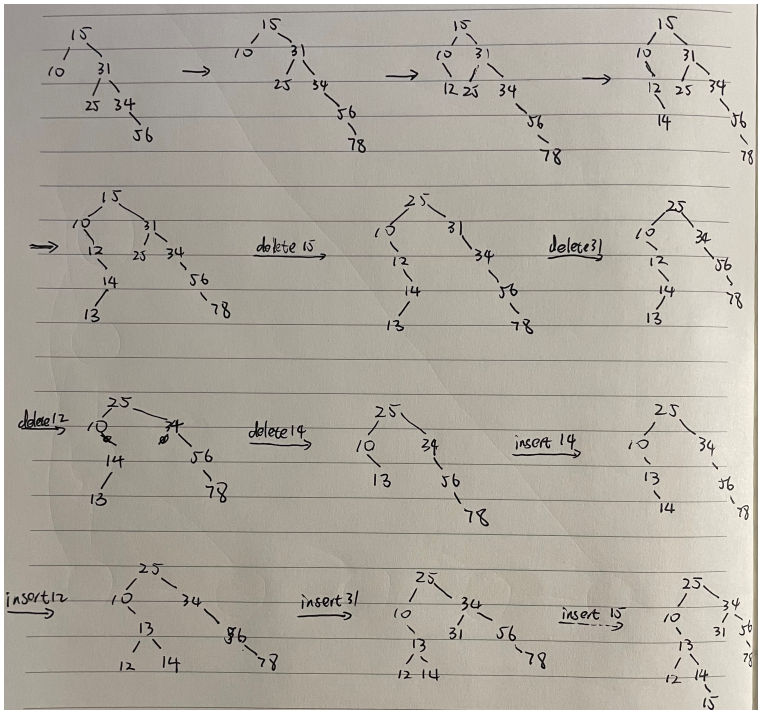
(a)


15
10
31
25
34
56
78
12
14
13

(b)


delete 15
delete 31
delete 12
delete 14

(c)



Problem 3

6.1-3

From the definition of the max-heap, we know that from a root to a leaf, it must be a decreasing sequence. So for any subtree, the maximum element is the root.

6.1-4

From the definition of the max-heap, we know that the minimum number must be on the leaf of the lowest level.

6.1-6

No, it is not a max-heap. I draw the tree and for 6, its children are 5 and 7. 7 is bigger than 6, so it is not a max-heap.

6.3-3

We know that a heap must be a complete binary tree. So the if the height of a node is h and we assume the depth of the heap is D. The the depth of this node have two possible results. $D_1 = D - h$ or $D_2 = D - h - 1$. For D_1, it means we can find a leaf with depth D from this node and D_2 means there has no leaf with depth D from this node. And we know that the total number of element n, it must be larger than 2^D and smaller than $2^{D+1} - 1$. And the number of all nodes with depth d is 2^d. So if we want to get the upper bound of the number of nodes with height h. It should be $2^{D-h-1} = \frac{2^D}{2^{h+1}} \leq \frac{n}{2^{h+1}}$. So it can be written as it shows in the problem.

12.3-2

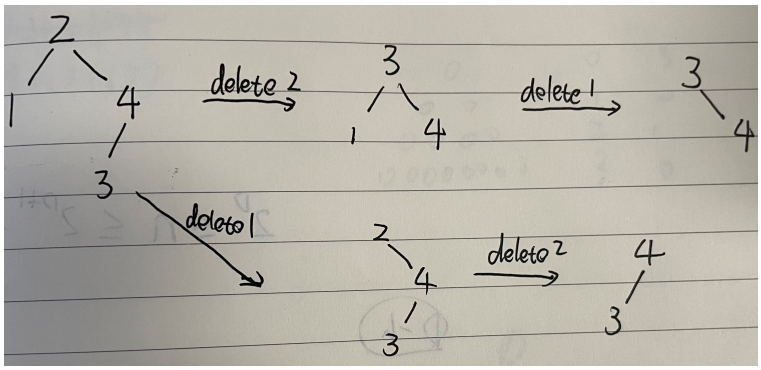
Because when we insert an element to the tree, it compares and follow the path to find the place to insert it. Therefore, if we want to search it, we need to follow the same path. However, we need to do another comparison to justify whether it is the key. So if we use n comparison when we insert, we use n+1 comparison when we search.

12.3-3

The time we used for output is $O(N)$, so we need to consider the time we used for build the tree. The worst case will happen if we build a tree with sorted numbers. We will get a tree with height n if we have n elements. The worst time will be $O(N^2)$. And if we build a tree which has height $O(lgn)$. Then we have the best time. It will be $O(nlgn)$.

12.3-4

No, the deletion is not commutative. And here is an example.



B.5-4

While n=1, the height of this tree is $h(1) \geq \lfloor lg1 \rfloor = 0$, it is true.

While n=k, k is any number, the height of this tree is $h(k) \geq \lfloor lgk \rfloor$.

While n=k+1 , we can divide it into a root, a left subtree and a right subtree. Left subtree has i elements and right subtree has $k-i$ elements. Then the height will be $h(k+1) = max(h(i), h(k-i))+1$. While $i > \frac{k}{2}$, $h(k+1) = h(i)+1 > \lfloor lg(\frac{k}{2}) \rfloor +1 = \lfloor lgk - lg2 \rfloor +1 = \lfloor lgk \rfloor$. Based on this, if k is at the edge condition which means $k = 2^n - 1$. Then $h(k+1) \geq \lfloor lg(k+1) \rfloor$.

So we can prove the argument.

Problem 4

(a)

False.

When we from the heap, we use $\Theta(N)$. And we extract the elements, we use $O(logN)$. So the final time will be $O(NlogN)$.

(b)

True.

It is the defination of the tree.

(c)

False.

Heap is a complete binary tree, not full.

(d)

True.

It follows the rule of min-heap.

(e)

False.

A binomial heap will have 2^n elements in it.

(f)

True.

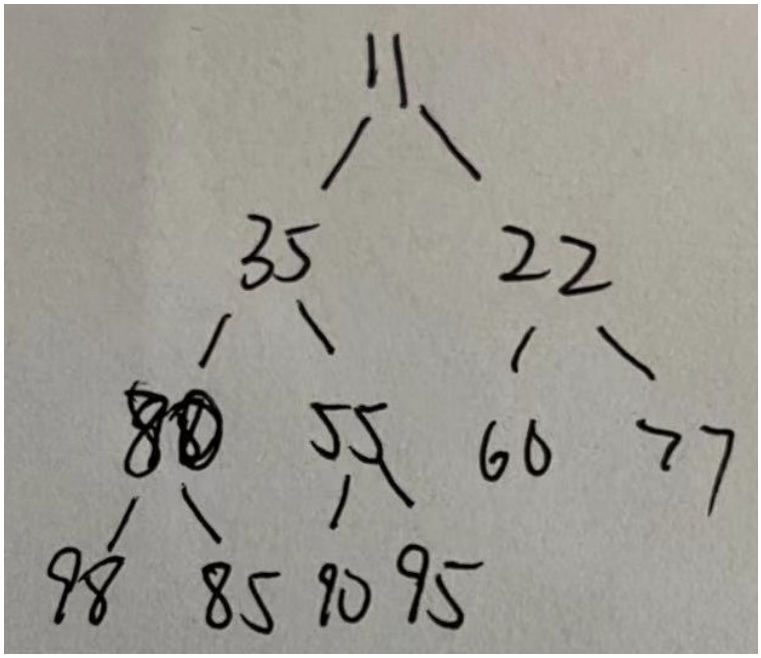
For any two numbers i and j, $j > i$. And another two elements A and B, $B > A$. Then we compare $i * A + j * B$ and $i * B + j * A$.

$$\begin{aligned} i * A + j * B - i * B - j * A &= i * (A - B) - j * (A - B) \\ &= (i - j) * (A - b) > 0 \end{aligned}$$

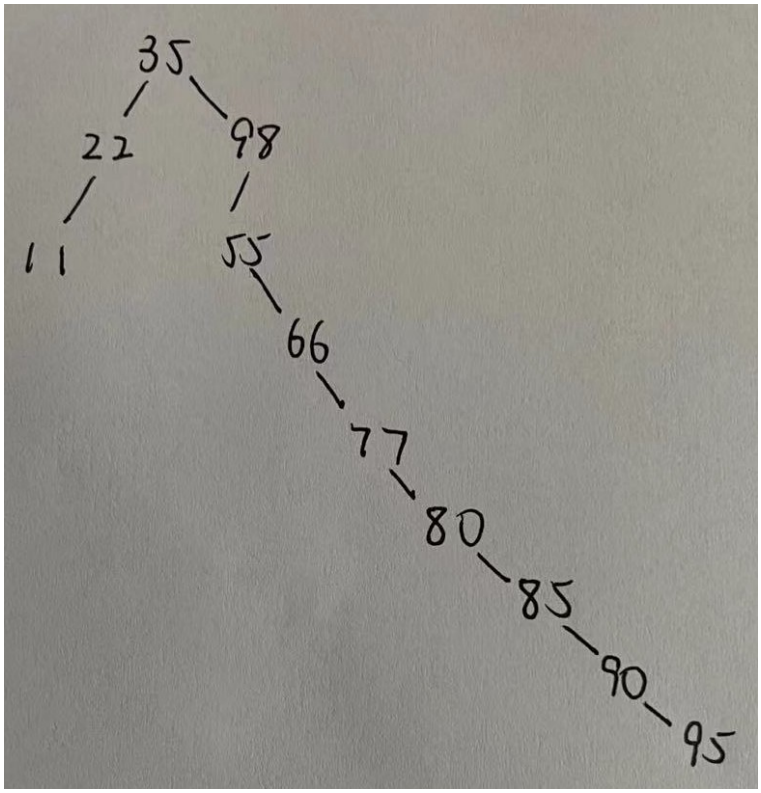
So an sorted array will get the maximum value.

Problem 5

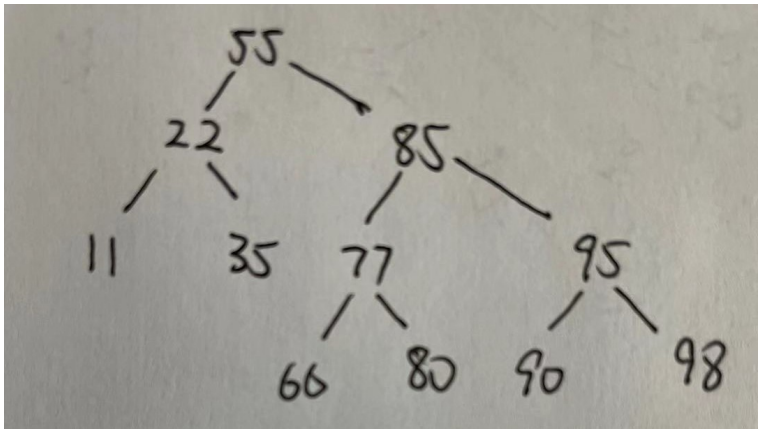
(a)



(b)

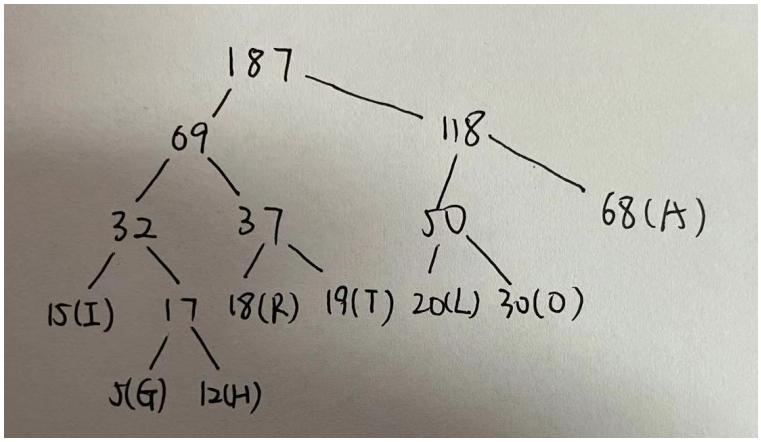


(c)



Problem 6

(a)



(b)

letter	weight	code	bits
A	68	11	2
O	30	101	3
L	20	100	3
T	19	011	3
R	18	010	3
I	15	000	3
H	12	0011	4
G	5	0010	4

So the average number of bits is,

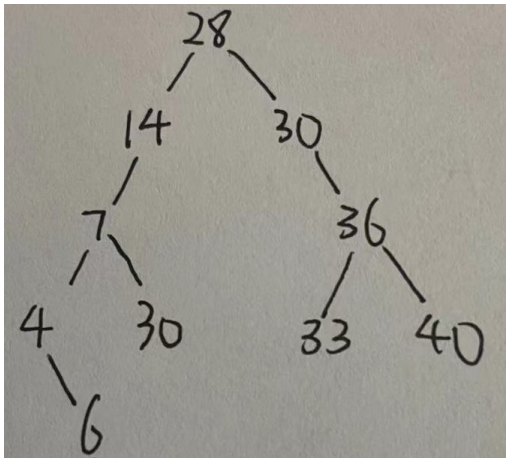
$$\begin{aligned} Average &= \frac{68*2+30*3+20*3+19*3+18*3+15*3+12*4+2*4}{68+30+20+19+18+15+12+5} \\ &= 2.663 < 3 \end{aligned}$$

(c)

We can let all weights of these letter be the same. Then the average bit is 3. The tree will be a complete full binary tree, for all cases.

Problem 7

(a)



From the tree, we know that $T_D(N) = 22$ and $T_H(N) = 13$. Besides, $H = 4$.

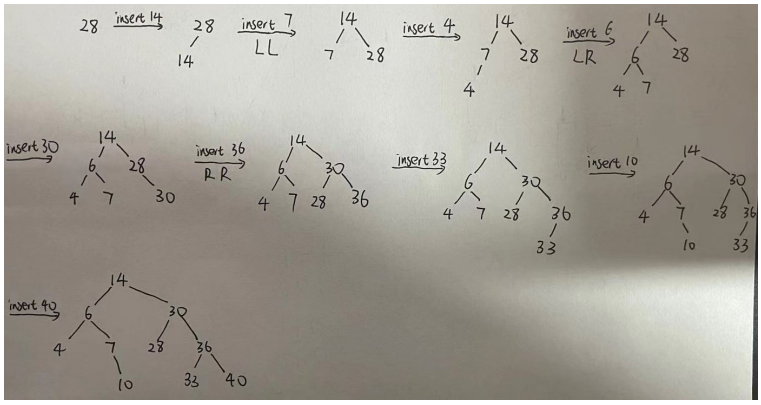
(b)

From (a), we know that $T_D(N) = 22$, $T_H(N) = 13$, $H = 4$. So we know,

$$T_D(N) + T_H(N) = 35 < 40$$

So the sum rule is wrong for binary search tree.

(c)



(d)

It decreased the value, and the new values are $T_D(N) = 19$, $T_H(N) = 9$, $H = 3$.

$$T_D(N) + T_H(N) = 28$$