HW 3 Yuqi Jin

1.

(a) False

if double size $N \to 2N$, and excuting time increase by 4,

So $T(2N) = 4 * T(N)$, in this case it is not growing linearly with 4N

it implies exponetial growth.

(b) True

The max number of nodes in a Height h binary tree is $2^{h+1} - 1$

$N(H) = 1 + 2 + 2^2 + 2^3 + \cdots + 2^H = 2^{H+1} - 1$,

At level 0, 1 node, level 1, 2 nodes, level 2, 2^2, 4 nodes, and so on

(c) False

In BST, all nodes in left subtree smaller, all nodes in right subtree greater

if deleting node with key x then node with key y is not same as deleting key y then key x. Because each time has to rearrange the Tree based on the BST property. That might lead to different tree structures.

(d) False,

It will take $O(n \log n)$, The binary-min Heap's height is $\log n$, there are n elements, each insertion operation take $O(\log n)$ by maintaining the min-Heap property

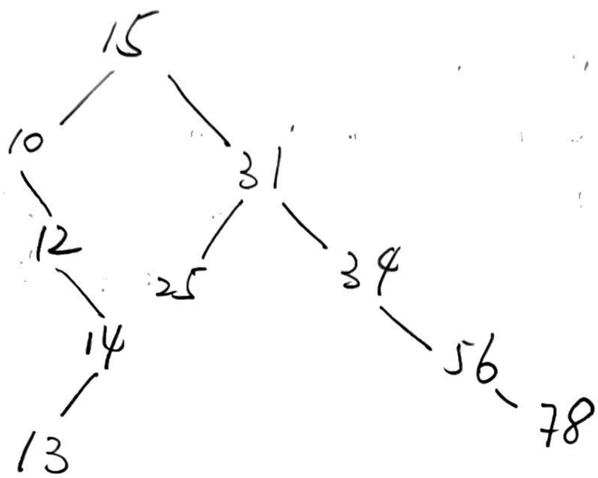
So total $O(n \log n)$

(e) True

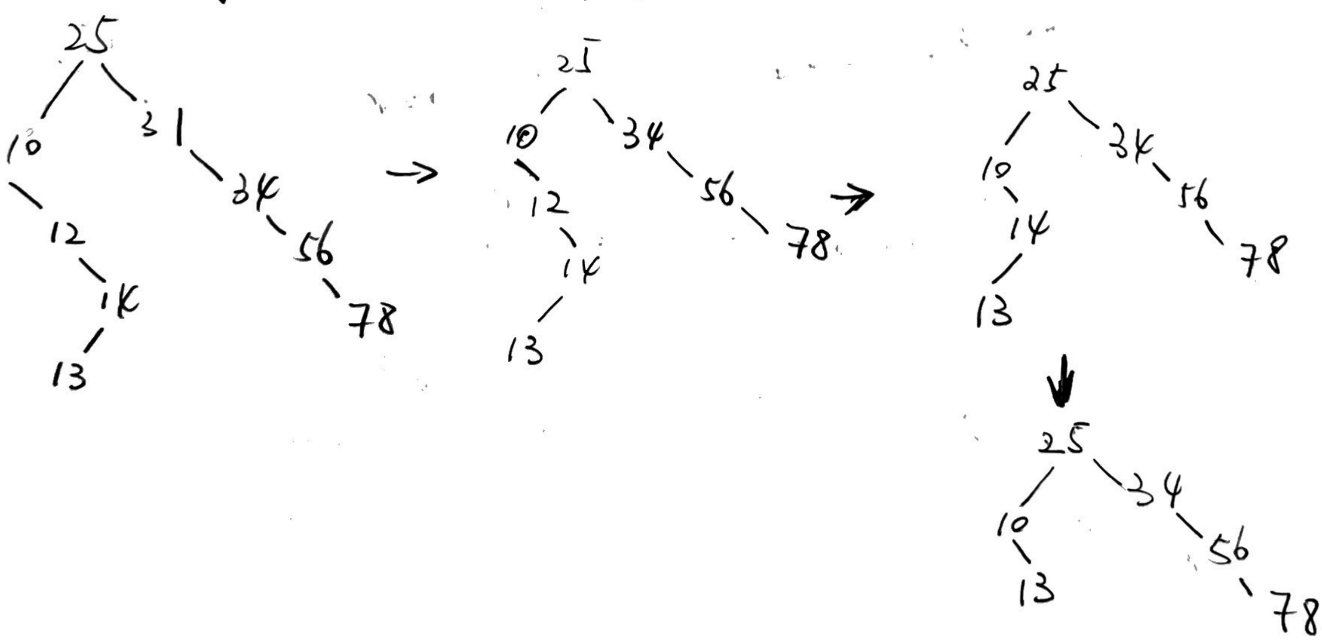
In binary min-Heap, the smallest one is the root at the top of tree. So the second-smallest is the child of the root by the property of min heap.

2.

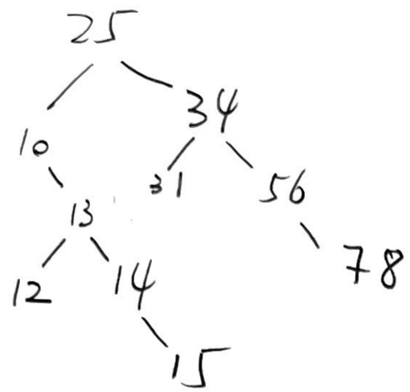
(a) Draw BST 15, 10, 31, 25, 34, 56, 78, 12, 14, 13



(b) delete following nodes 15, 31, 12, 14



(c) reverse order and reinsert into tree 14, 12, 31, 15



After that when taking inorder traversal, it will be 10, 12, 13, 14, 15, 25, 31, 34, 56, 78

it is an ascending order, also it is an BST based on its property

3.

6.1-3

in a max heap, the max-heap property is for every node i other than root $A[\text{Parent}(i)] \geq A[i]$, so the root of the subtree is always the topmost parent, that contains the largest value in whole subtree

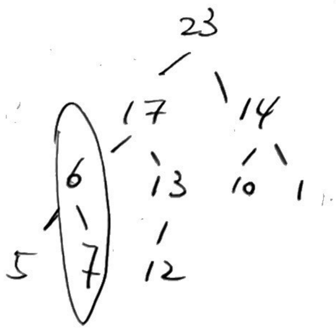
6.1-4

The smallest node may only reside in a leaf node. Based on the max heap property, it can not be the parent node.

6.1-6

$\{23, 17, 14, 6, 13, 10, 1, 5, 7, 12\}$

No, it is not a maxheap



The 7 node is greater than parent 6

6.3-3

Show that there are most $\lceil n/2^{h+1} \rceil$ nodes of height h in any n-element heap.

According to 6.1-7, the elements in the sub array $A[(\lfloor n/2 \rfloor + 1) \dots n]$ are leaves of the tree.

At height 0, there are $\lceil n/2 \rceil$ leaf nodes., it is a base case for a proof by induction rule.

$\lceil n/2^{0+1} \rceil = \lceil n/2 \rceil$ nodes. at $h = 0$. means it has most $\lceil n/2 \rceil$ nodes.

By removing all leaves, $n - \lceil n/2 \rceil$ nodes. $= \lceil n - n/2 \rceil = \lfloor n/2 \rfloor$ for new heap

$\left\lceil \frac{\lfloor n/2 \rfloor}{2^{0+1}} \right\rceil$ for $h = 0$, now based on the equation $\lceil n/2^{h+1} \rceil$

$= \left\lceil \frac{\lfloor n/2 \rfloor}{2} \right\rceil = \lceil \lfloor n/4 \rfloor \rceil = \lfloor n/4 \rfloor$ leaf nodes for an $\lfloor n/2 \rfloor$ elements heap

Based on 6.1-7 the leaves of an n-element heap are index by $\lfloor n/2 \rfloor + 1$, $\lfloor n/2 \rfloor + 2, \dots, n$. So the proof holds for all values of h and n.

12.3-2

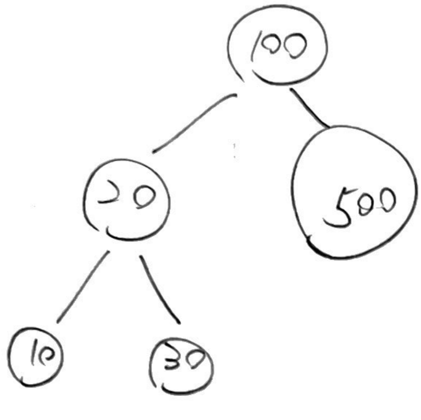
when insert a node in a BST, start with the root, compare the node value and root value. if node value is less than root value, go to left. Otherwise move to right subtree. Repeat this step, until find an empty spot also appropriate then insert it.

For searching for a value, also follows the rules; which is compare to root.val. This time also requires that check the value itself, searching is to search for a target node in BST. So number of nodes examed in searching for a value =

1+ number of nodes examined when the value was first inserted into the

Example

insert 40

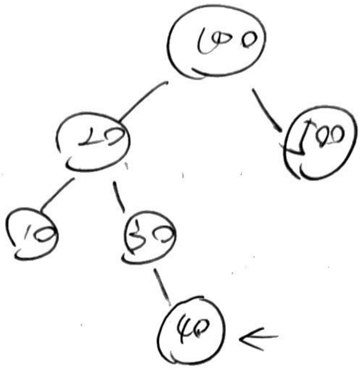


1. compare 100, < 100, go left child

2. Compare 20, > 20, go right child

3. Compare 30, > 30, go right side of 30

it is empty spot and appropriate, then insert

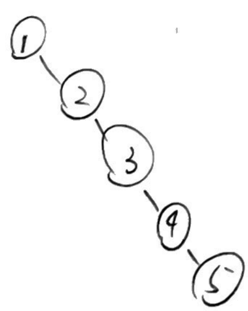


But for searching a node 40, do steps 1, 2, and 3. Then extra step compare 40 equals or not to the node value.

12.3-3

worst case. if the numbers are sorted already. Example: 1, 2, 3, 4, 5

The BST:



The height of tree is n

During insertion, each node takes O(n) time to reach correct position, because the tree is like Linked list

There are n numbers, total insertion is $O(n^2)$

Inorder Traversal take O(n), So sum up $O(n^2+n) = O(n^2)$

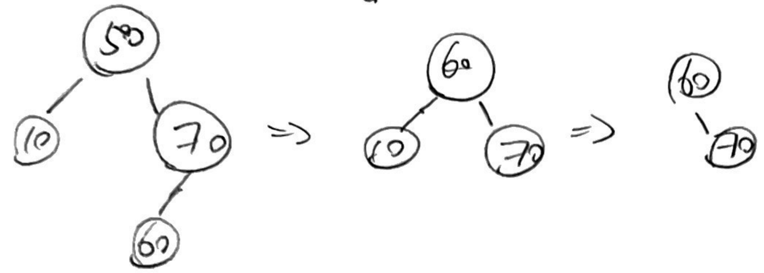
Best case. The tree is balanced, the height does not exceed logn, The time for insertion is O(logn), and there are n numbers, so O(nlogn). In order traversal is O(n). So sum up $O(n\log n + n) = O(n\log n)$

12.3-4

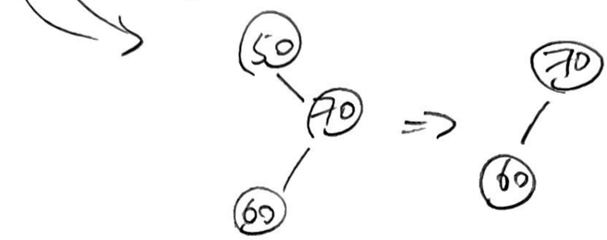
it is not always "Commutative."

Counter Example:

delete 50 then 10



delete 10 then 50



B.5-4

Use induction to show that a nonempty binary tree with n nodes has height at least $\lfloor \lg n \rfloor$

Base case, when there is one node in tree. the height is $\lg 1 = 0$

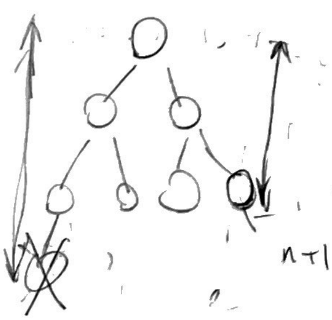
Let the binary tree has n+1 nodes,

if n+1 is not equal to 2 to the power of, then the height is $\lfloor \lg(n+1) \rfloor$

else, choose a leaf node of greatest depth then remove it, the height is at least $\lfloor \lg n \rfloor$

So it means that the tree is complete binary tree on n nodes.

when removing the leaf node, the height will change by 1, So $\lfloor \lg n \rfloor + 1 \geq \lfloor \lg(n+1) \rfloor$



Let $n+1 = 2^k$

height $\lfloor \lg n \rfloor = k-1$, $2^k - 1$ nodes have to be complete binary.

4.

(a) False

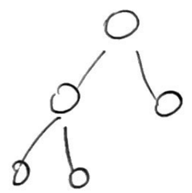
Once build a Heap, takes $O(n)$ time.

the largest Element is extracted and places the end of the array

Each extraction takes $O(\log(n))$ time.

There are n extractions, result is $O(n\log(n))$

(b) True.



ex: $N=5$, $N-1=4$ edges

Tree can traverse in both directions, travel from one node to any other node.. But there're no cycles, start/end node can't be same

Tree is special case of graph, with N nodes, $N-1$ edges.

(C) False

Binary Heap is like complete binary tree., does not have to be full binary tree.

All nodes are as far as left possible. is completely filled, except possibly the last.

The elements not 2 to the power.

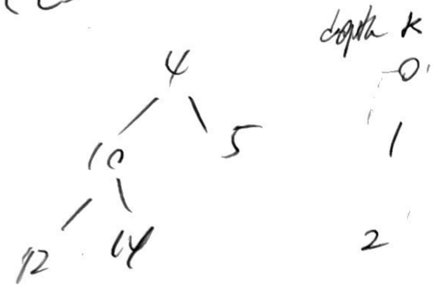
(d) True

4 is smallest.

then $10 > 4$, $5 > 4$

then $12 > 10$, $14 > 10$

(e)



$k=0$, $2^0 = 1$ node

$k=1 = 2^1 = 2$, but it has 3 nodes, wrong

False, for binomial min-Heap, it has exactly 2^k nodes, has depth as k. The root has degree k and children of the root are themselves Binomial trees with order k-1, k-2, ... 0 from left to right.

(f). True

Sort the array in ascending order and compute the sum $i \times a[i]$

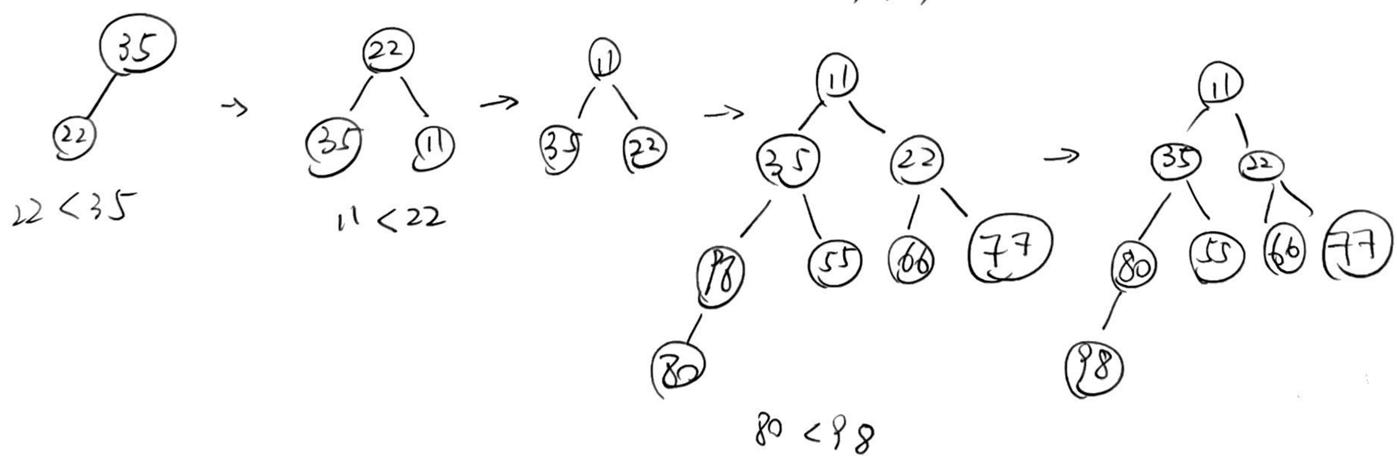
Once the array is sorted, the max value is on the largest index, the min value is on the smallest index. The time worst case is $O(n \log(n))$

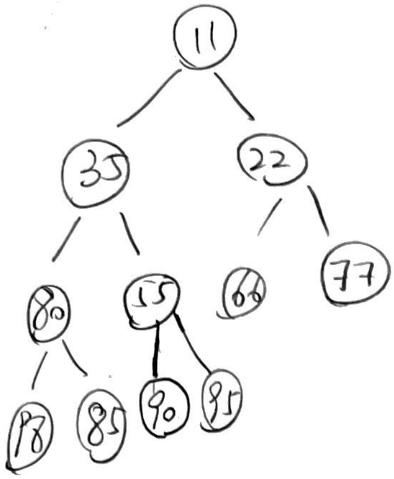
Sort it array $\log(n)$ + iterate array to sum up max n

5.

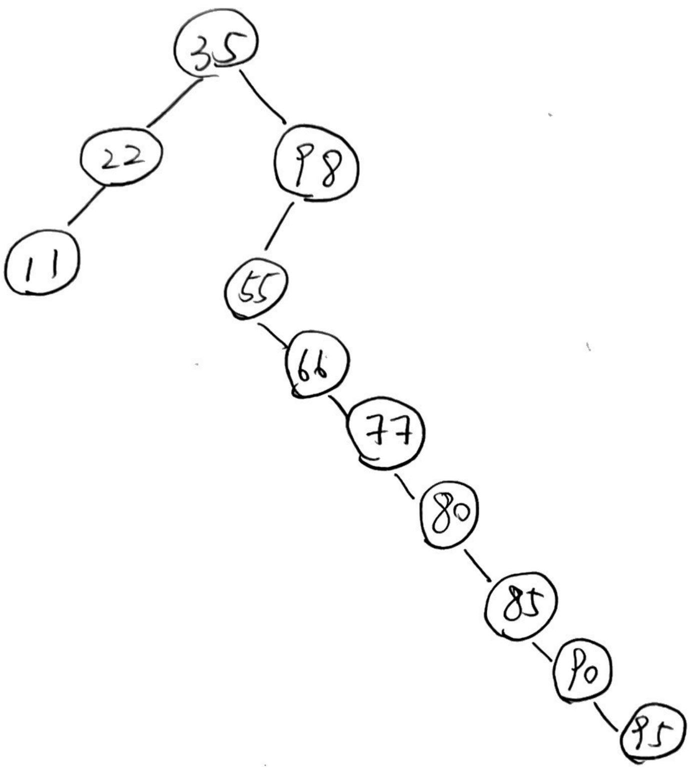
35, 22, 11, 98, 55, 66, 77, 80, 85, 90, 95

(a)



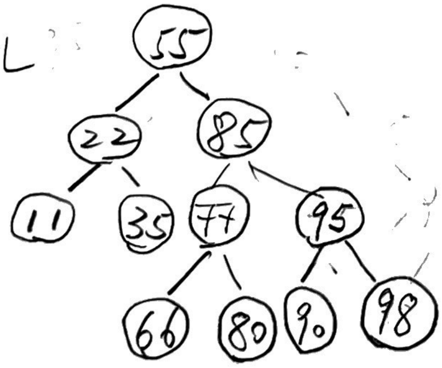


(b) BST.



MinHeap is Top Bottom sorted, BST is left-Right sorted.

(c) AVL



AVL is self Balanced BST, $|HL - HR| = 0, 1$

6.

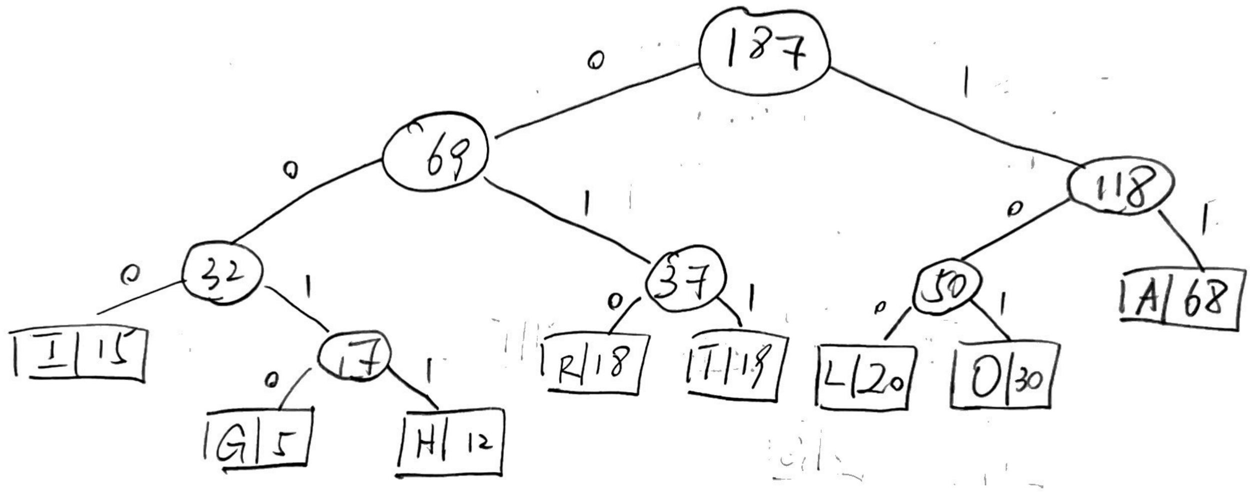
(10)

alphbat :	A	L	G	O	R	I	T	H
weights :	68	20	5	30	18	15	19	12

(a) Sort the weight

G	H	I	R	T	L	O	A
5	12	15	18	19	20	30	68

Arrange tree small weights to left



(b)

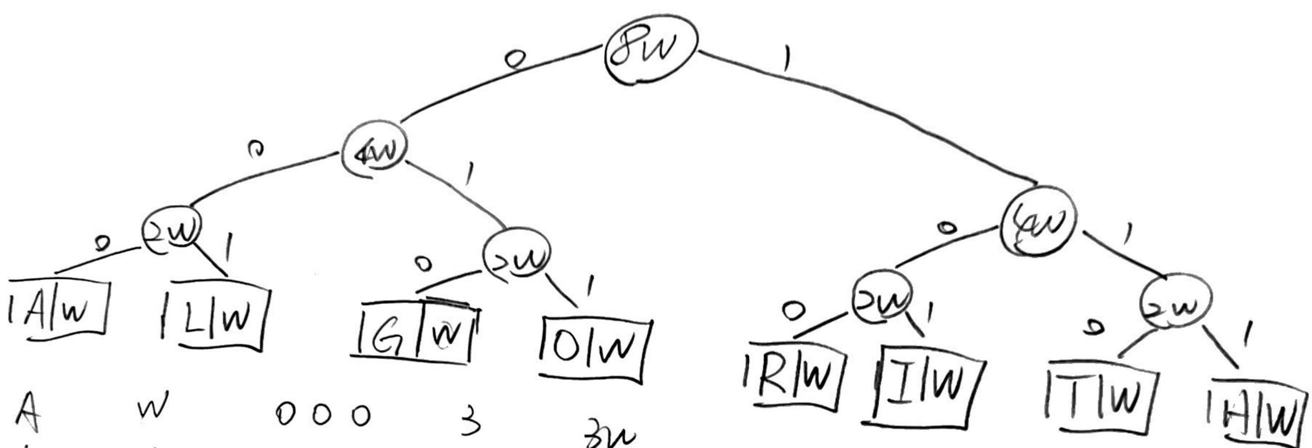
letter	Weight	Code	Bits	Count
I	15	000	3	45
G	5	0010	4	20
H	12	0011	4	48
R	18	010	3	54
T	19	011	3	57
L	20	100	3	60
O	30	101	3	90
A	68	11	2	136

Total 187 510

Average Bits = 510 / 187 = 2.727

Yes, it is less than 3, 2.727 < 3

(c) If each symbol has the same weight w



A	w	000	3	3w
L	w	001	3	3w
G	w	010	3	3w
O	w	011	3	3w
R	w	100	3	3w
I	w	101	3	3w
T	w	110	3	3w
H	w	111	3	3w

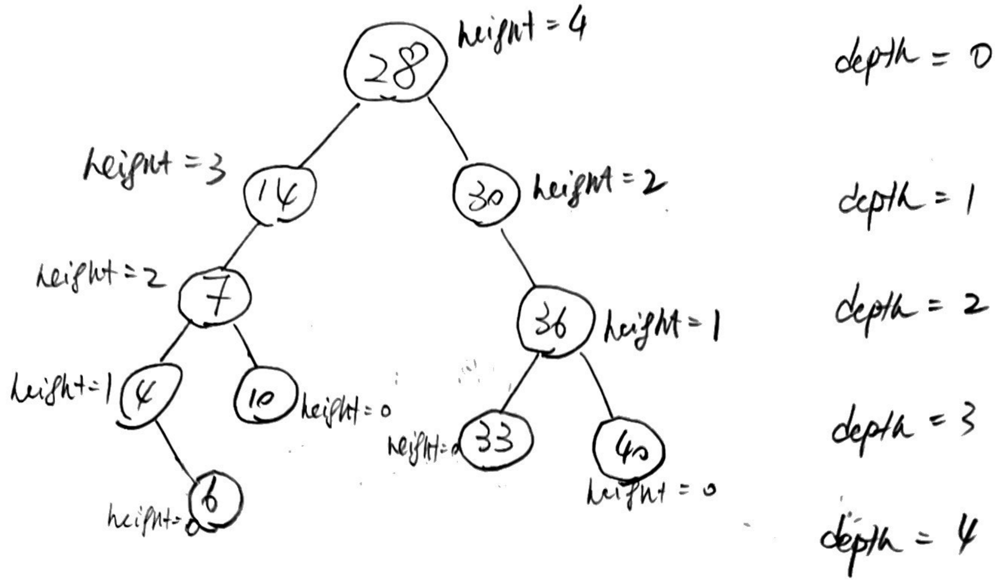
Average bits = 24W / 8W = 3

Yes, the Huffman code tree always a full tree.

7.

28 14 7 4 6 30 36 33 10 40

(a)



$T_H(N) = 4+3+2+2+1+1+0+0+0+0$
$= 13$

$T_D(N) = 0+1+1+2+2+3+3+3+3+4$

$= 22$

(b) $N = 10$, $H = 4$

$\log_2(10)\ 10 = (3.3219)\,10 = 33.219$, $T_H(N) + T_D(N) = 13 + 22 = 35$

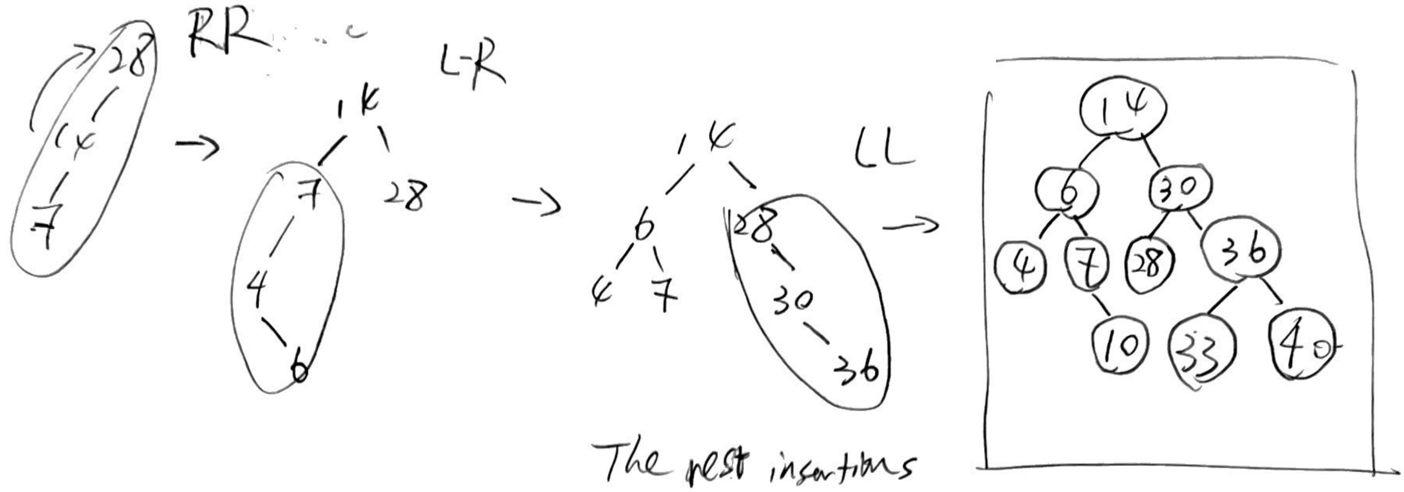
$N(N-1) = 10(9) = 90$

$33.219 \le 35 \le 90$

So, $\log_2(N)\ N \le T_H(N) + T_D(N) \le (N-1)\,N$

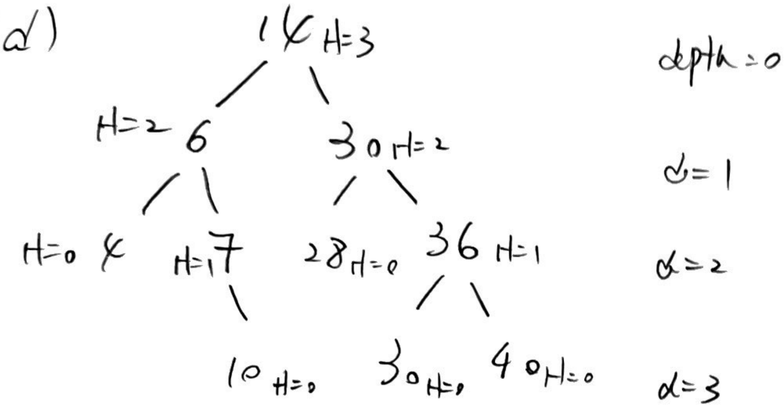
(c)

28 14 7 4 6 30 36 33 10 40



The rest insertions do not destroy the AVL Property, just insert them based on the values comparsion

(d)



$T_H(N) = 3+2+2+0+1+0+1+0+0+0$
$= 9$

$T_D(N) = 0+1+1+2+2+2+2+3+3+3$
$= 19$

compare previous answers, AVL decreases the $T_H(N)$, $T_D(N)$

$T_H(N) + T_D(N) = 9 + 19 = 28$