1. Consider the function f(x,n) where x is assumed to be an object of a number type and n is assumed to be a nonnegative integer.

```
def f(x, n):                                      (c): replace as follows:
    if n == 0:
        return 1
    elif n%2 == 0:
        return f(x, n//2) * f(x, n//2)    -> return f(x, n//2)**2      (*)
    else:
        return x*f(x, n-1)
```

(a) What is this function doing?

(b) What is the asymptotic running time of this function?

(c) Suggest a simple improvement to this function that guarantees an asymptotic running time of $O(\log n)$.

(5+5+5)

(*) This guarantees a recursion depth of $O(\log n)$ with one recursive call per level.
$\Rightarrow$ total cost is $O(\log n)$

(a) It's computing integer powers of x:

$$f(x,n) = x^n$$

via:
- base case: $x^0 = 1$
- n odd: $x^n = x\, x^{n-1}$ and $n-1$ is even
- n even: $x^n = \left(x^{\frac{n}{2}}\right)^2$

(b) The way it is implemented, the recursion is calling $f(x, \frac{n}{2})$ twice, rather than storing the result: Let $c(n)$ be the number of function calls when $f(x,n)$ is evaluated (as a measure of total running time as each function call adds $O(1)$ running time). Clearly

$c(1) = 1$, $c(2) = 2\,c(\frac{2}{2}) = 2$, $c(2^2) = 2\,c(2) = 2^2$ ... $c(2^i) = 2c(2^{i-1})$

$\Rightarrow c(2^i) = 2^i$. If n is not a power of two, we can bound from above by the nearest power of two, so in general $\underline{c(n) = O(n)}$.

2. Sketch, using Python or Python-like pseudocode, the implementation of a function that computes the height of a tree. (10)

```
def height(T, v):
    # Computes height of subtree of T rooted at v
    h = 0
    for c in T.children(v):

        h = max(h, height(T, c))

    return h+1


def height(T):
    return height(T, T.root())
```

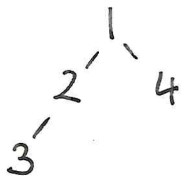
3. (a) Suppose you have array that is sorted in increasing order. Is it a valid array representation of a min-heap?

   (b) You have a valid array representation of a min-heap. Then you reverse the array. Do you get a valid array representation of a max-heap?

Justify your answer in each case. (5+5)

(a) Yes, because the children of any node are located after the node in the array representation. So heap order is satisfied.

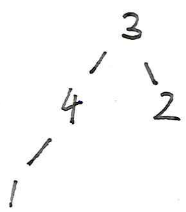
(b) No. Counterexample:



is a min-heap

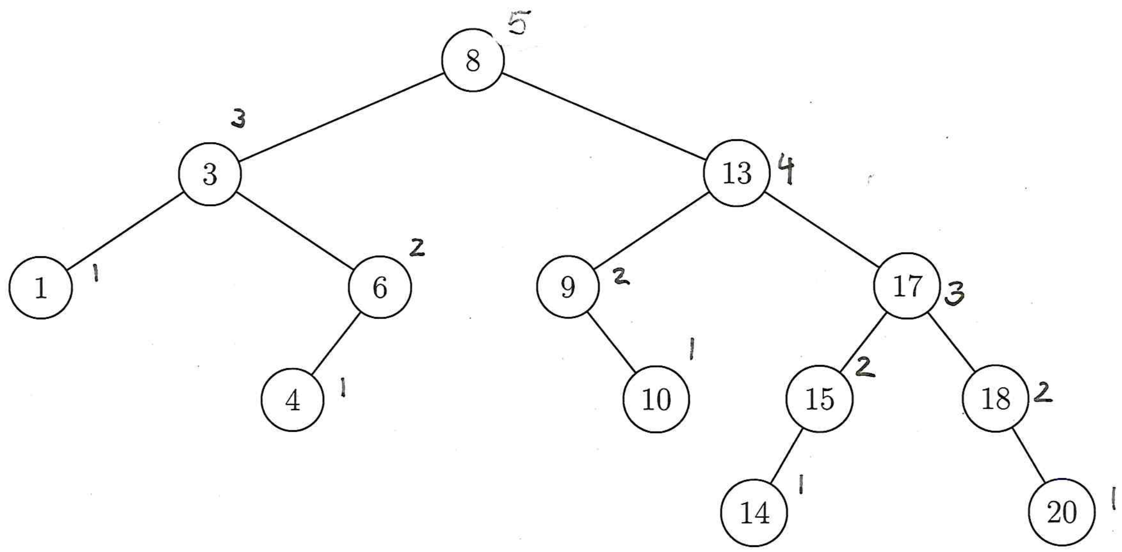
1 2 4 3 is its array representation

3 4 2 1 is the reversed array representation, which encodes the binary tree



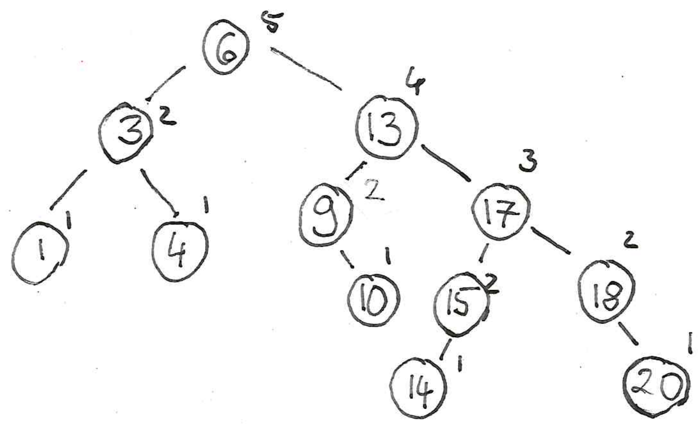
This is not a (max)-heap !

4. Delete the key 8 from the following AVL tree. Indicate all subtree heights before the deletion, and show the changes introduced by the deletion and subsequent rebalancing step-by-step. (You may consult the summary of tree roations on the front page.)



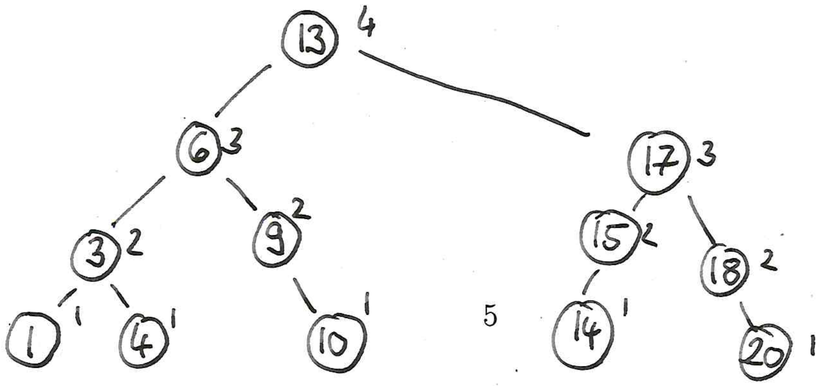
(10)

Convention 1: Move largest element from left subtree into deleted position:



Root node is out of balance!

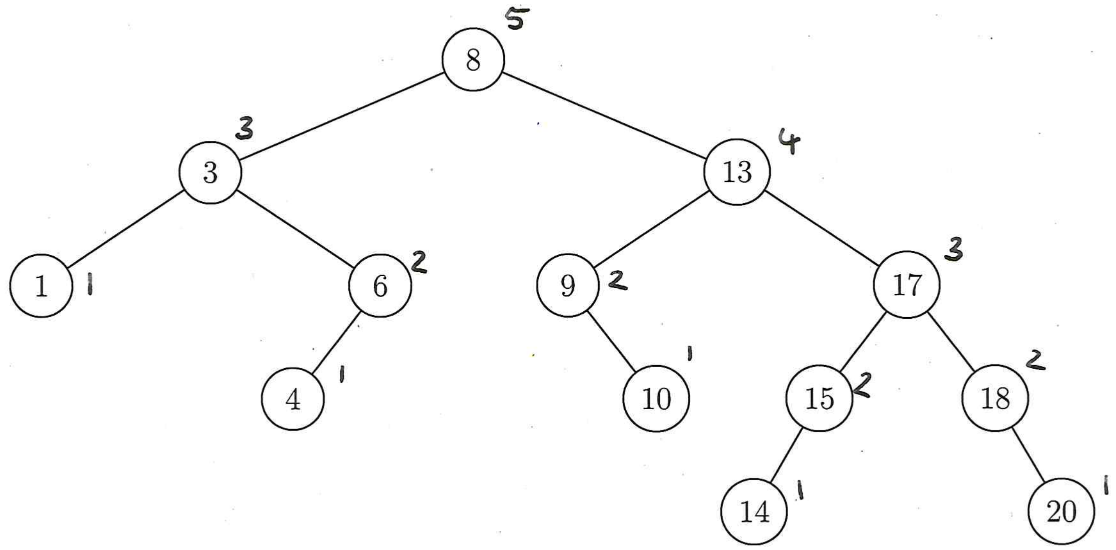
single rotation



Tree is now in balance.

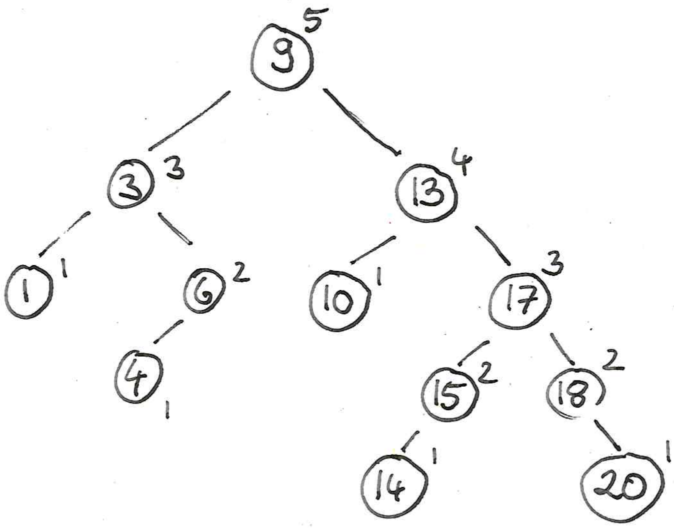
ALTERNATIVE SOLUTION

4. Delete the key 8 from the following AVL tree. Indicate all subtree heights before the deletion, and show the changes introduced by the deletion and subsequent rebalancing step-by-step. (You may consult the summary of tree roations on the front page.)



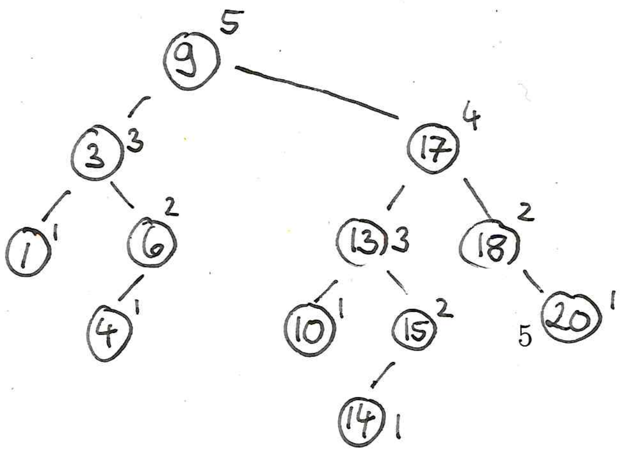
(10)

Convention 2: Move smallest element from right subtree into deleted position:



Node 13 is out of balance

single rotation



Tree is now in balance

5. We noted in class that every implementation of a sorted map can be turned into a sort algorithm as follows: Given a list L to be sorted, take an initially empty sorted map M and move the elements of L one-by-one into the sorted list. Then traverse M in-order, moving the elements back into L.

For each of the following data structures that can be used to implement a sorted map, state the asymptotic running time in two cases: worst case, and L already sorted. (10)

Implementation	**Worst case**	**L already sorted**
Sorted linked list	$\Theta(n^2)$	$\Theta(n^2)$
Reverse sorted linked list	$\Theta(n^2)$	$\Theta(n)$
Skip list	$\Theta(n^2)$ but $O(n \log n)$ expected	$\Theta(n^2)$ but $O(n \log n)$ expected
AVL tree	$\Theta(n \log n)$	$\Theta(n \log n)$
Splay tree	$\Theta(n \log n)$	$O(n)$

6. (a) Show that every rooted tree has at least one leaf.

(b) For a connected graph $G = (V, E)$, let $v = |V|$ denote the number of vertices and $e = |E|$ denote the number of edges. Show that G is a tree if and only if

$$v = e + 1 .$$

(c) How many connected components[1] does a forest[2] with $v = 20$ vertices and $e = 15$ edges have?

(5+5+5)

(a) Let v be any vertex. If it is not already a leaf, it has at least one child. Let v then be one of the children, and repeat. If this process does not terminate with finding a leaf in a finite number of steps, the graph is not a tree, or it is infinite (which we do not consider!).

(b) So long as G has a leaf, iteratively remove the leaf and its connecting edge. This leaves the remaining graph connected, and the difference $v-e$ does not change. Eventually, one of two cases occurs:

(i) There is only one vertex left. Then G is a tree and $v-e=1$.

(ii) There is no leaf $\Rightarrow$ G is connected but not a tree $\Rightarrow$ G has a cycle. Within the cycle, there are as many edges as vertices, for every additional vertex, there is at least one edge connecting it. $\Rightarrow e \geq v \Rightarrow v-e \leq 0$.

This proves that G is a tree iff $v-e=1$.

[1] A connected component of a graph G is a maximal connected subgraph of G.

[2] A graph G is a forest if it contains no cycles. (So every connected subgraph of a forest is a tree.)

(c) For every "missing" edge relative to the balance $v-e=1$, there is one additional connected component. Here: $v-e=5$, i.e. the forest has 5 trees.