Basic question 1: Complexity

What are the orders of growth of the following two functions?

$$f(n) = 996 \cdot n^2 + 0.002 \cdot n^3 + \frac{10^7}{n^7}$$

$$g(n) = (\log_2 n + \log_{10} n) \cdot (7n + 5n^2)$$

Write your answer in O-notation, and be as exact and simple as possible.

Answer

$f(n) \in O(n^3)$

$g(n) \in O(n^2 \log(n))$

Basic question 2: Insertion sort

Perform in-place *insertion sort* of the following 7-element array of letters:

0	1	2	3	4	5	6
F	A	E	K	H	B	C

Answer

Write down how the array looks like after each iteration of the main (outer) loop:

Solution note: The sublist that is sorted after each step is shown in **purple**.

After step 1:

0	1	2	3	4	5	6
A	F	E	K	H	B	C

After step 2:

0	1	2	3	4	5	6
A	E	F	K	H	B	C

After step 3:

0	1	2	3	4	5	6
A	E	F	K	H	B	C

(can be skipped since it's the same as step 2)

After step 4:

0	1	2	3	4	5	6
A	E	F	H	K	B	C

After step 5:

0	1	2	3	4	5	6
A	B	E	F	H	K	C

After step 6:

0	1	2	3	4	5	6
A	B	C	E	F	H	K

After step 7:

0	1	2	3	4	5	6

Note: depending on how you perform the sorting, you might not need all steps!

Basic question 3: Binary search vs linear search

Assume you have the following *sorted* array of integers:

0	1	2	3	4	5	6	7	8
12	21	37	42	56	64	75	87	93

Now you want to search for the value 70 in the array. What comparisons do you need to do until you can be certain that the value isn't there, if you use linear search or binary search?

Answer

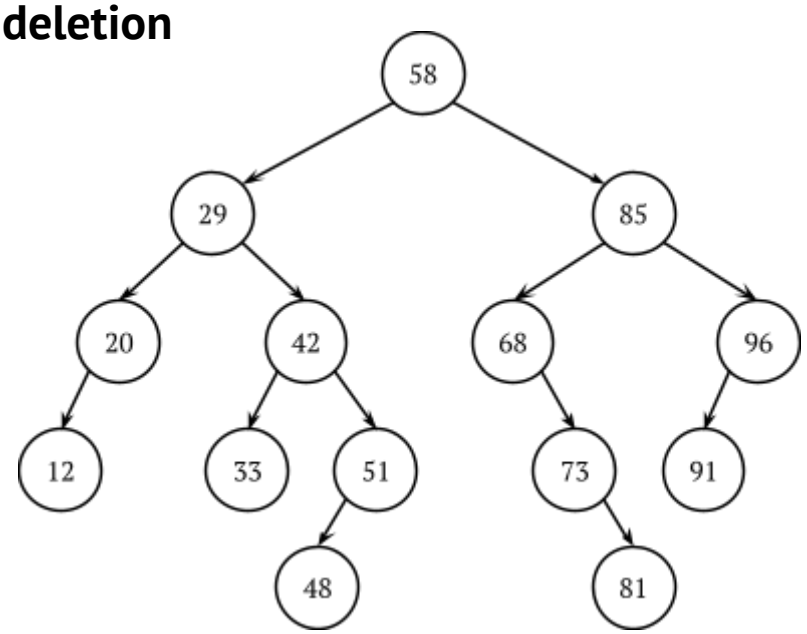
Write one comparison per row, starting from the top. Because you are always comparing with the same value (70), you only have to write the other value that you compare with.

Stop when you are certain that the value (70) is not in the array – you won't need all rows.

Linear search	Binary search
12	56
21	75
37	64
42	
56	
64	
75	

Basic question 4: BST deletion

Delete the value 58 from the binary search tree to the right, using the standard BST deletion algorithm.



Answer

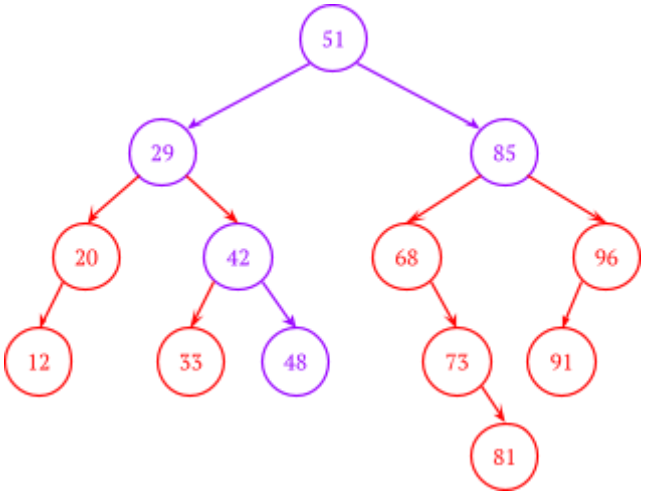
There are two possible solutions, depending on if the new root node is 51 or 68.

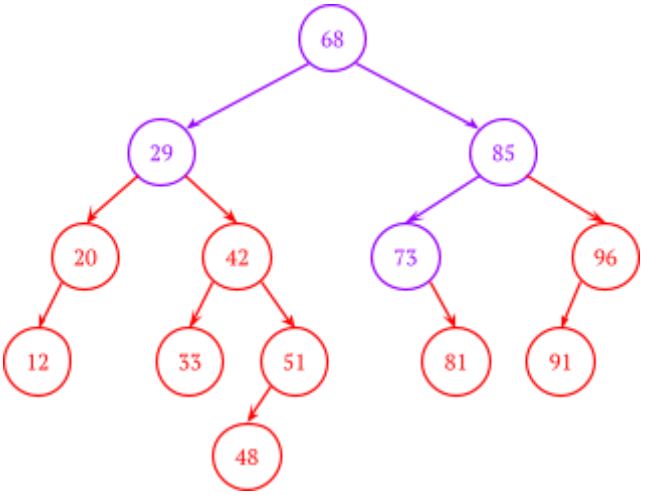
What nodes have a different parent and/or different children after you deleted 58?

51 has different parent and children
48, 29, 85 have different parents
42 has a different right child

68 has different parent and children
73, 29, 85 have different parents
85 has a different left child

Draw the final tree:





Basic question 5: Priority queues

This quote is from the Wikipedia entry of "Heap (data structure)":

> … *a heap is a specialized tree-based data structure that satisfies* ***the heap property*** …

(a) Explain the heap property for a min heap.
(b) Draw a binary min heap with 6 nodes where *exactly one* parent-child link violates the heap property.

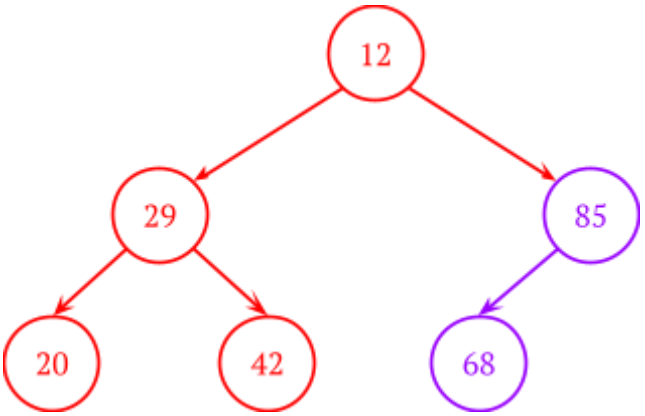
Answers

Explain the heap property for a min heap:

No child node has a smaller value than its parent.

(There are several equivalent formulations of the property).

Draw a 6-node binary min heap where *exactly one* parent-child link violates the heap property.



Which of the parent-child links violates the heap property? 68 is smaller than 85

Basic question 6: Hash tables

Suppose you have the following hash table, implemented using *linear probing*.
You can assume that the table hasn't been resized, and no elements have been deleted.

0	1	2	3	4	5	6	7	8	9
52	19	20	322	703	172				33

The hash function is the digit product modulo 10:

$$h(d_k...d_2d_1d_0) = (d_k \times ... \times d_2 \times d_1 \times d_0) \bmod 10,$$

which means that, e.g., $h(647) = 8$.

a) What are the hash values for each of the elements in the hash table?
b) Which number(s) could have been the first one(s) to be inserted into the table?
c) Why is this a particularly bad hash function?

Answers

Hash values for each element:

$h(19) = \mathbf{9}$	$h(52) = \mathbf{0}$	$h(322) = \mathbf{2}$
$h(20) = \mathbf{0}$	$h(172) = \mathbf{4}$	$h(703) = \mathbf{0}$
$h(33) = \mathbf{9}$		

List the number(s) that could have been inserted first: only 33 and 52
(*Note*: there might be several and you must list all of them)

Why is this a particularly bad hash function?

It has a very skewed distribution. Here are some reasons why: Very many numbers will get a hash code of 0 – whenever there's a 0 in the number, or when there's a 2 and a 5. Even-numbered hash codes will be extremely common – as long as there's an even digit in the number, the hash code will be even. And the digit 1 will not have any effect on the hash code at all.

Basic question 7: Graphs

The below table shows an undirected graph in adjacency list representation, edge costs included.

Node	Adjacency list
A	C:6, E:2, G:5
B	C:5, E:1, D:1, F:5
C	A:6, B:5, G:7
D	B:1, E:2, F:4

Node	Adjacency list
E	B:1, A:2, D:2
F	B:5, D:4, G:3
G	A:5, C:7, F:3

Perform **Prim's algorithm** with **starting node A** to construct a minimum spanning tree.

Answer

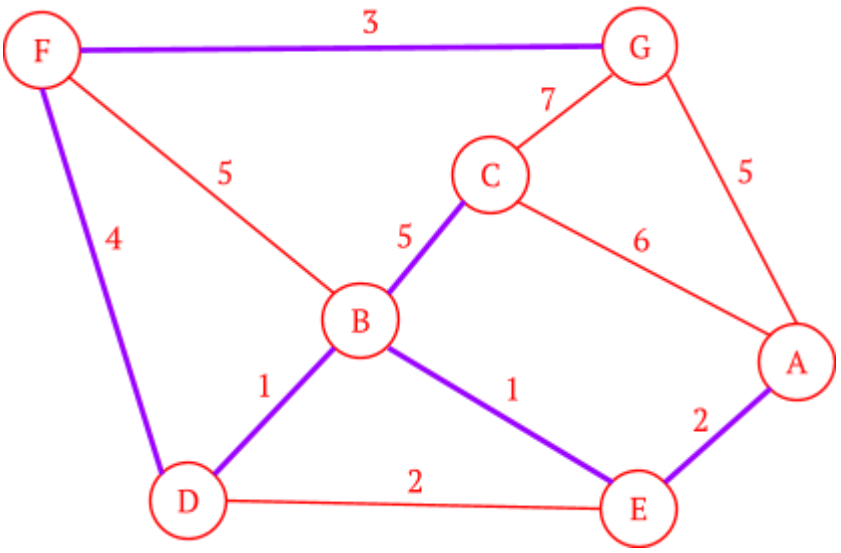
List the edges of the spanning tree *in the order they are produced* by Prim's algorithm.

Write the edges in the form AC, DF, ...
Note: Since the graph is undirected, AC and CA refer to the same edge.

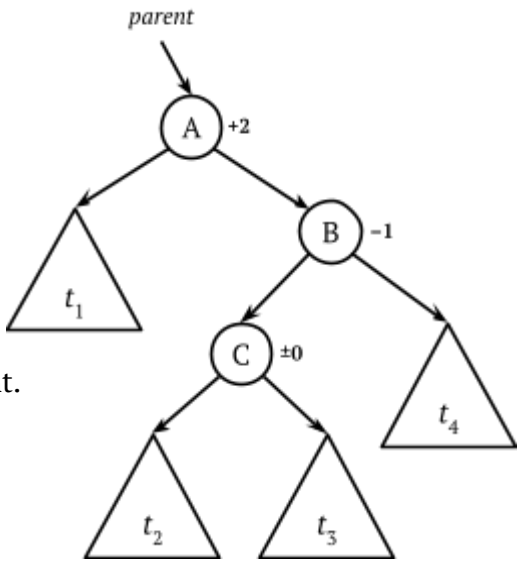
AE, EB, BD, DF, FG, BC

Draw the graph with the edges belonging to the spanning tree marked.

You can, e.g., mark the spanning tree edges by making them thicker.



Basic question 8: AVL rotation



To the right is a generic instance of an unbalanced node in an AVL tree. Show what rotations you need to perform to rebalance the tree.

(The triangles denote subtrees of some unknown size.)

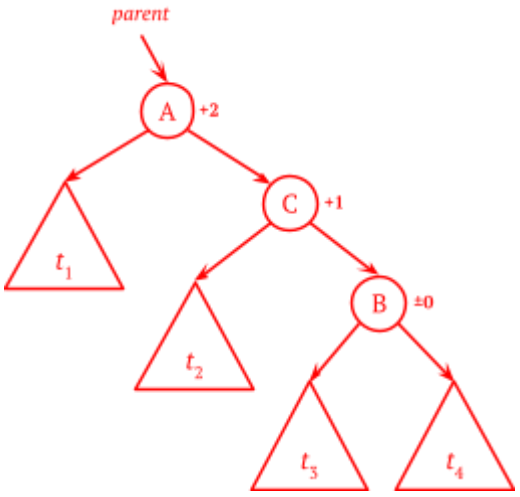
For each rotation you perform, explain what nodes you rotate and show the resulting tree in the same way as to the right.

Use the same node and subtree names (A, B, C, t_1, …, t_4). Don't forget to write the updated balance factors on the nodes.

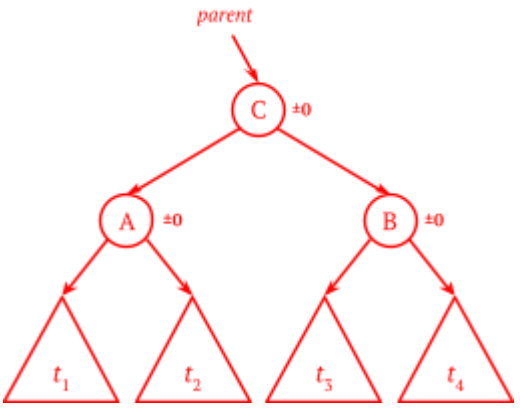
Answer

This is a right-left case, so we need two rotations.

(1) rotate B right, making it a right child of C.



(2) rotate A right, making it a left child of C.



Advanced question 9: Complexity

Here is the formal definition of big-O (in Swedish "ordo"):

$$f(x) \in O(g(x)) \quad \textbf{iff} \quad \exists c, x_0.\ \forall x \geq x_0.\ |f(x)| \leq c \cdot g(x)$$

Prove the following law, using the above definition:

$$\textbf{if } f(x) \in O(g(x)) \textbf{ and } g(x) \in O(h(x)), \textbf{ then } f(x) \in O(h(x))$$

Answer

$f(x) \in O(g(x))$ means $\exists c_0, x_0.\ \forall x \geq x_0.\ |f(x)| \leq c_0 \cdot g(x)$

$g(x) \in O(h(x))$ means $\exists c_1, x_1.\ \forall x \geq x_1.\ |g(x)| \leq c_1 \cdot h(x)$

Now, since $|g(x)| \leq c_1 \cdot h(x)$, we can insert this into the first inequality:

$$|f(x)| \leq c_0 \cdot g(x) \leq c_0 \cdot c_1 \cdot h(x)$$

This inequality holds for all $x \geq \max(x_0, x_1)$.

But this means that $|f(x)| \leq c_2 \cdot h(x)$ for all $x \geq x_2$, where $x_2 = \max(x_0, x_1)$ and $c_2 = c_0 \cdot c_1$.

So we have proved the following:

$$\exists c_2, x_2.\ \forall x \geq x_2.\ |f(x)| \leq c_2 \cdot h(x)$$

Which is the same as $f(x) \in O(h(x))$.

Advanced question 10: Meldable heaps

Here is a simple recursive data structure for binary heaps:

```
class Heap:
    value : int
    left  : Heap
    right : Heap
```

Assume that you have a $O(\log n)$ function for melding two heaps, creating one single heap:

```
function meld(h₁ : Heap, h₂ : Heap) → Heap
```

Your task is to define the following standard heap functions in terms of the function `meld`:

```
function getMin(h : Heap) → int
function removeMin(h : Heap) → Heap
function add(h : Heap, n : int) → Heap
```

Apart from `meld` you are allowed to use the `Heap` instance variables, and to construct new heaps (using the `Heap` constructor).

All functions should have time complexity $O(\log n)$.
You may write your answer in Java, Python, Haskell, or pseudocode.

Answer

```
function getMin(h : Heap) → int:
    return h.value

function removeMin(h : Heap) → Heap:
    return meld(h.left, h.right)

function add(h : Heap, n : int) → Heap:
    return meld(h, new Heap(n, null, null))
```

Advanced question 11: What does this code do?

Assume that you have a binary tree data structure with the following definition:

```
class Tree:
    value : string
    size  : int
    left  : Tree
    right : Tree
```

The `size` property gives the total number of values in this tree, including the top node. The `left` and `right` nodes can be **`null`**, as usual.

Now, what does the following function do?

```
function f(t : Tree, k : int) → string:
    if t.left ≠ null:
        if k < t.left.size:
            return f(t.left, k)
        k = k - t.left.size
    if k == 0:
        return t.value
    return f(t.right, k-1)
```

Answer

The function returns the inorder element n:o *k*, i.e., it treats the binary tree as an array.

Note: it uses 0-based indexing, so the leftmost node is n:o 0.

Advanced question 12: Robot navigation in a BST

A robot is navigating the nodes of a binary search tree (BST). It can execute the commands:

```
is_left_child() → bool
is_right_child() → bool
has_left_child() → bool
has_right_child() → bool

move_to_parent()
move_to_left_child()
move_to_right_child()
```

The robot starts at some node. Write an algorithm that moves the robot to the next node in the order of the keys. You may assume this node exists. Use only the above functions.

You may write your answer in Java, Python, Haskell, or pseudocode.

Answer

```
function next_node():

    if has_right_child():
        move_to_right_child()
        while has_left_child():
            move_to_left_child()

    else:
        while is_right_child():
            move_to_parent()
        move_to_parent()
```