Basic question 1: Complexity

Here is a simple function that takes two *linked lists* as input and returns a linked list that only contains elements that are present in *both* input lists:

```
function both(xs : list, ys : list) -> list:
    set = new empty set (using a self-balancing BST)
    res = new linked list

    for each x in xs:
        add x to set

    for each y in ys:
        if set is empty:
            return res

        if y in set:
            add y to the front of res
            remove y from set

    return res
```

Assume that the input lists `xs` and `ys` have the same number of elements *N*. We want to know what the asymptotic worst-case time complexity of the function ’`both`‘ is in terms of the number of elements *N*. Write your answer in O-notation. Be as exact and simple as possible. Justify why the complexity of the function has this order of growth.

Answer

Complexity: _____

Justification (you can also add notes directly in the code above):

Basic question 2: Sorting

Perform a quicksort partitioning of the following array, using the *median of all elements* present in the array as pivot:

0	1	2	3	4	5	6	7	8
48	55	64	83	4	20	54	25	61

Note: quicksorting the left and right parts of the partition is not part of this question.

What is the asymptotic worst-case complexity of the partitioning algorithm? Write your answer in O-notation and be as exact and simple as possible. Explain why the complexity of partitioning an array has this order of growth.

Answer

What is the median value you used as a pivot? _____

Write down how the array looks after the partitioning:

0	1	2	3	4	5	6	7	8

Complexity: _____

Justification (you can also add notes directly in the code above):

Basic question 3: Hash tables

Here is a simple data structure that models a teacher:

```
class Teacher:
    name : String
    office : int
```

Suppose we have defined the following teacher objects:

```
L = new Teacher("Lex", 6114)
C = new Teacher("Christian", 6467)
H = new Teacher("Hazem", 6476)
J = new Teacher("Jonas", 6108)
P = new Teacher("Peter", 6125)
```

and have inserted them in the following hash table, which is 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
H	P		C	L	J

The hash function is the *length of a teacher's name* modulo 6:

$h(t)$ = `t.name.size()` mod 6

for example

h(`C`) = `C.name.size()` mod 6 = "`Christian`"`.size()` mod 6 = 9 mod 6 = 3

a) Which object(s) could have been the first one(s) to be inserted into the table?
b) Why is this a particularly bad hash function?

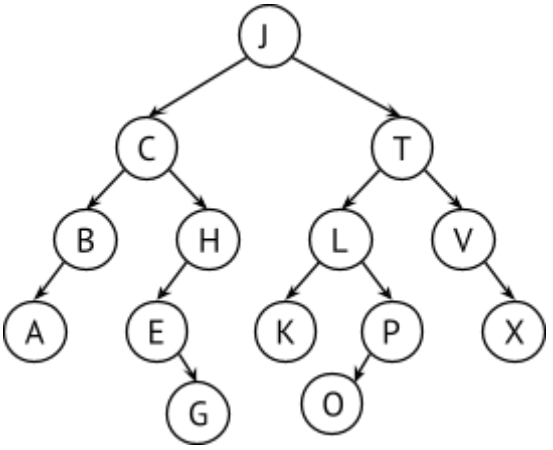
Answers

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

Why is this a particularly bad hash function?

Basic question 4: AVL trees

Consider the AVL tree to the right.



Someone just made an insertion, but forgot to perform the necessary rotation(s) to restore the AVL balance. It is your job to figure out which value got added, which node now violates the AVL property, and perform the necessary tree rotation(s) to restore the AVL balance.

Answer

Which value got added? _____

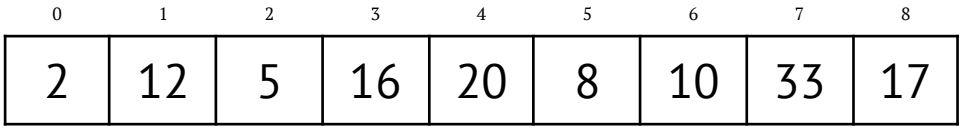
Which node violates the AVL balance property? _____

What rotations does the AVL algorithm do?

Resulting tree after the rotation(s):

Basic question 5: Binary heaps

Here is a minimum-heap `h`, implemented with a binary heap, with integer values:



We have defined an `update` function on a minimum-heap:

```
function update(h : array, oldVal : int, newVal : int):
    int i = find(h, oldVal)
    if i >= 0:
        h[i] = newVal
        if newVal > oldVal:  sink(h, i)   // sift new value down
        else:                swim(h, i)   // sift new value up

function find(h : array, val : int) -> int:
    for i from 0 to h.length - 1:
        if h[i] == val:
            return i
    return -1
```

The update function tries to find a given value (`oldVal`) and replaces it with a new one (`newVal`) in the underlying array (`h`). It subsequently calls the correct binary heap function (`sink` or `swim`) to restore the heap invariant.

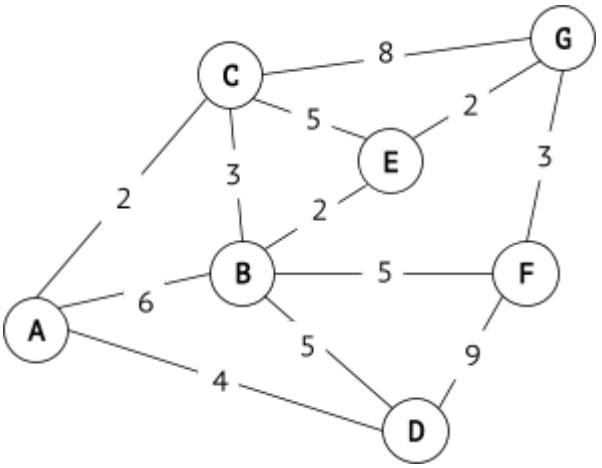
Suppose we update the given heap `h` with: `update(h, 16, 1)`, draw the resulting heap as a tree.

Answer

Draw the updated min-heap as a tree.

Basic question 6: Graphs

We can represent any graph using an *adjacency list*. Your task is to write down such a representation for the *undirected*, *weighted* graph on the right. Note that the weights must be present in the adjacency lists.



In addition, perform Dijkstra's algorithm on the graph to the right, starting in node **A**. In which order does the algorithm visit the nodes, and what is the computed distance to each of them?

Answer

Adjacency lists:

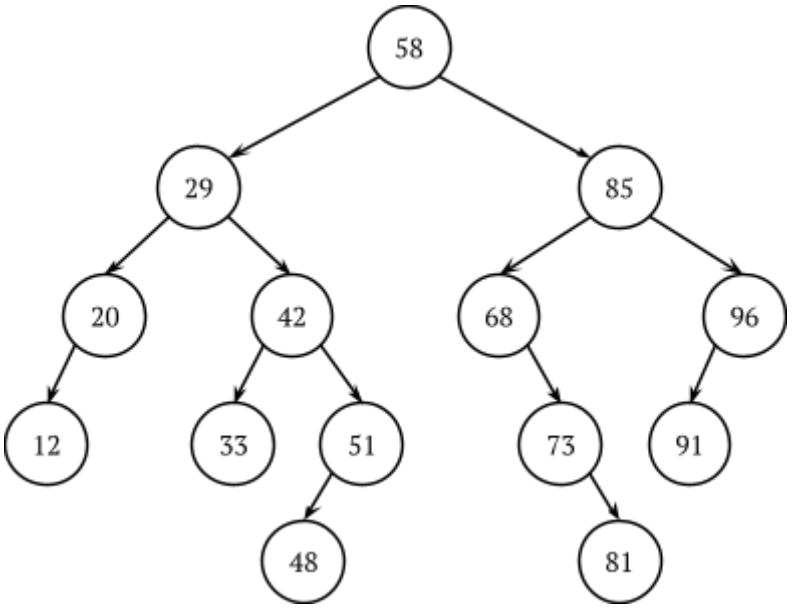
Node	Adjacency list

Dijkstra:

	first visited						*last visited*
vertex	A						
distance from A	0						

Advanced question 7: Binary search trees

The figure below show a binary search tree with integers:



Your task is to define a function that finds the closest value to a given value in a *balanced* binary search tree. The value x is closest to y when |x - y| is minimum. For example, the value closest 85 is 81, and 20 is closest to 12. You may assume that the given value is present in the tree, and if there are multiple nodes that are equally close, you may just choose one. Your function should be written in either pseudocode, Haskell, Java or Python.

To get awarded one point your solutions should have a linear complexity, $O(n)$, or better. For two points your solution should have a logarithmic complexity, $O(\log\ n)$.

Explain the complexity of your solution.

Answer

Advanced question 8: Cyclic order binary search

Let **A** be an array of $n \geqslant 1$ integers. Assume that **A** is in *strict cyclic order*: $\mathbf{A}[i] < \mathbf{A}[(i + 1) \bmod n]$ for all indices i except one. (Note that this implies that all elements of **A** are distinct.)

Part A

Find an algorithm that computes the index of the minimum of **A** and runs in time $O(\log n)$.

```
int findMinIndex(int[] A)
```

Part B

Find an algorithm that checks if **A** contains an element v and runs in time $O(\log n)$:

```
boolean contains(int[] A, int v)
```

Give your algorithms in pseudocode or a suitable programming language, which we mentioned before. Justify why your algorithms work, and why they have these complexities.

Solving part A will give one point, solving both part A and B gives two points.

Answer

Advanced question 9: Design a matrix data structure

Your task is to design a data structure for storing a *matrix*, or two-dimensional array, of integers. The data structure should support the following operations:

- `zero(n, m)` Create an n-by-m matrix. *Initially, all values in the matrix should be zero*.
- `get(i, j)` Return the value at row `i`, column `j` of the matrix. Assume that rows and columns are numbered starting from 0.
- `set(i, j, x)` Set the value at row `i`, column `j` of the matrix to `x`.

All three operations must have asymptotic worst-case time complexity $O(\log\ N)$, where N is the maximum number of integers: $N = n \times m$. Implementing these operations will give you one point.

Note! The creation of an array of length n takes $O(n)$ time.

To get two points, your data structure should support two more operations:

- `scale(k)` multiply every number `x` in the matrix with `k`.
- `setRow(i, x)` set all values at row `i` to `x`.

These operations should be $O(\log\ N)$ as well, or better.

Your answer should specify what design you have chosen, how each operation is implemented, and motivate the time complexity of your solution.

You should write your answer in pseudocode, Haskell, Java or Python. You may freely use standard data structures and algorithms from the course without explaining how they are implemented. Furthermore, you may assume that a type or class exists that represents a pair of numbers, and that comparisons take $O(1)$ time.

Answer