Exercise Sheet, Week 4

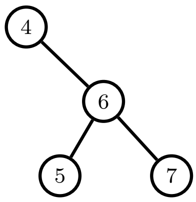
1 Trees in Mem

We represent trees in `Mem`. Each node takes 3 locations in memory:

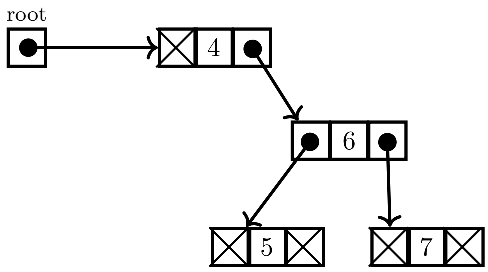
- 0th location: stores the pointer to the left subtree (or `END` if it is empty)
- 1st location: stores the value stored in the node
- 2th location: stores the pointer to the right subtree (or `END` if it is empty)

We store the address of the root node of the tree in variable `root`.

For example:



becomes



Exercises:

1. Use the function `allocate_memory(n)` to create a representation of the tree shown above. Make sure that you store the address of the root node in variable `root`.
2. Starting from `root`, write a code that inserts a node with `8` as the right child of the node `7` into the same tree as you created in (1).
3. Write a function `void insert(int root, int x)` which inserts the value `x` into the binary search tree with the root stored in `root` (you can assume that the tree is not empty).
4. Write a function `int branchSum(int root)` which computes the sum of all numbers on the rightmost branch of the tree. (In the above case, it would be $4 + 6 + 7 = 17$.)
5. Write a function `int sum(int root)` which computes the sum of all numbers stored in the nodes of the tree.
6. What is the time complexity of your function `sum` from (5)? Express the time complexity with respect to $n =$ the **size** of the tree.
7. **Bonus:** Write a function `int maxLessThan(int root, int x)` which finds the largest value stored in the tree which is $\leq x$.

2 Challenging: Amortized complexity

Consider a modification of dynamic arrays, which we did in the class:

1. initially allocate an array of 1000 entries
2. whenever the array becomes full, increase its size by 100, 100^2, 100^3, 100^4, 100^5, ... elements.

What is the amortized complexity of insertion now?
What is the problem with this approach?