Beginselen van Programmeren
Exercise Session 8:
Sorting and Searching

Ex. 1: (Random) list generator

Write a parameterized function that generates random lists of integers (within a certain interval) of specified length. Also, write a parameterized function that generates sorted lists of specified length (you may use `list.sort()` here).

Ex. 2: Shotgun sort

Implement the Shotgun sort algorithm. The Shotgun sort algorithm is a particularly inefficient one. Although, it is very simple and consists of two alternating stages: (1) randomly shuffle the list you wish to sort, and (2) check if the list is in order.
1. Implement the random shuffler: Write a separate function that randomly shuffles the list to sort. You can use `random` from Python.
2. Implement the checking function: Write a separate function that, given a list, checks if the list is sorted correctly.
3. Bring them together Write a function that uses the two previous functions to sort an input list. Test the function on some automatically generated random lists.

Ex. 3: Bubble sort

3.1

Implement the Bubble sort algorithm by writing a function `bubblesort(l)` that takes a list `l` as its argument and sorts it, using Bubble sort. Bubble sort works by iterating several times over the list. For each iteration it goes through the whole list, and make a pairwise comparison between consecutive values. Two values are to be swapped if they are in the wrong order ($l_i > l_{i+1}$). We show an example of one iteration on the unsorted list $[5, 2, 3, 1, 4]$:

$[\mathbf{5}, \mathbf{2}, 3, 1, 4]$
$[2, \mathbf{5}, \mathbf{3}, 1, 4]$
$[2, 3, \mathbf{5}, \mathbf{1}, 4]$

$[2, 3, 1, \mathbf{5}, \mathbf{4}]$
$[2, 3, 1, 4, 5]$

This procedure is repeated until the list is fully sorted (worst case `len(l)` times).

3.2

Prove that your bubble sort algorithm works.

Ex. 4: Binary Search

Implement the Binary Search algorithm using an iterative approach. In binary search, we want to search for a certain value in a sorted list, and return its index if the value is in the list, and return `None` if it is not in the list. In binary search we look at the middle item of the list, and check if the value that we are searching for should be in the left or right part of the list (by using the order of the list). Then we continue with the sub-list in which we expect the value to be, and repeat the procedure until we either find the element, or not (in case it is not in the list).

Ex. 5: Binary Search Tree

Implement the Binary Search Tree. The Binary Search Tree is a tree structure with the following contraints:

1. each node has at most 2 subtrees.
2. the nodes in the left subtree of some node A are all smaller than or equal to A's value.
3. the nodes in the right subtree of some node A are all larger than A's value.
4. both subtrees are binary search trees.

Write a function `insert(tree, item)` which inserts an item in the tree and a function `search(tree, item)` that retrieves the given value and the path to the given value. Return `None` if the value is not in the tree. You may use recursion.

Ex. 6: Binary Tree Sort

Implement Binary Tree Sort using the Binary Search Tree. The Binary Tree Sort algorithm builds a Binary Search Tree of a given list and outputs the sorted list by traversing the tree. Write a function `build_tree(list)` that, given a random list of integers, builds a binary search tree and returns it. Reuse the function `insert` from the previous exercise. Write

a function `build_list(bstree)` that, given a Binary Search Tree, returns the original list. You may use recursion.

Run some tests to check if your algorithm works correctly. Tip: reuse the checking function you wrote for Shotgun sort.

Ex. 7: Compare sorting performance of your sorting algorithms and .sort in Python

Compare the performance of Binary Tree Sort and .sort in Python. You can also include Shotgun Sort here for very small lists because Shotgun sort takes a lot of time on bigger lists. Reuse the random list generator you wrote previously. Reuse the checking function you wrote for Shotgun Sort. Use the `timeit` package.

Write the Insertion Sort algorithm (p. 605) to compare the Binary Tree Sort algorithm with. Test both algorithms with different input list sizes (say, 10, 20, 30, 50, 100, 200, 300, 1000, 10000). What do you see?

Extra: use the module `matplotlib` to plot the sorting time against the size of the list to sort.