Practice problems 2

Problem 1 - Sorting

Write a function `sort(lst)` that takes a list `lst` of integers, and returns a sorted version of this list.

In this exercise you are not allowed to use `sorted` python built-in (nor any other built-in that just sorts its input).

Note
Full marks for solutions that run in time $O(n \log n)$ -either in the worst case, or on average (for randomized algorithm like QuickSort). Partial marks for slower algorithms.

Reminder
If you want to implement a randomized algorithm (like QuickSort), you might want to chose a random position of a pivot. To do this you can use a function randint from a library random:

```
import random
random.randint(a, b)
```

returns a random integer in range from a to b inclusive.

Example

```
sort([5,3,1,7,11]) == [1,3,5,7,11]
```

Testing
You can use Python implementation `sorted` to test your code. for example:

```
import random
for i in range(100):
    test = [ random.randint(0, 100000) for j in range(30)]
    if sort(test) != sorted(test):
        print("Wrong")
        break
print("Everything OK!")
```

Problem 2

Write a function `is_tree(n, edges)` that takes as an input an integer `n`, and a list `edges`. The function should return `True` if the graph represented by `edges` on `n` vertices is a tree.

Each element of the list `edges` is a pair, consisting of two indices (u, v) from 0 to $n-1$ (with a promise that $u \neq v$ - the graph does not have self-loops).

Hint
A graph on `n` vertices is a tree if and only if it has exactly `n-1` edges and is connected. You can check this using a DFS algorithm discussed on lecture 12.

Examples

```
is_tree(6, [(0, 1), (1, 2), (0, 3), (3, 4), (3,5)]) == True
is_tree(6, [(0, 1), (1,2), (0, 2), (3,4), (3,5)]) == False
```

Problem 3

Write a function `find_collision(n, f)` which takes an integer `n` and a function `f` as an argument. The function `f` takes as an input a integer in the range 0 to n, and outputs an integer in the range 0 to $n-1$.

By pigeonhole principle, there exists a pair x_1, x_2 of numbers between 0 and n such that $f(x_1) = f(x_2)$. Your task is to output any such pair.

Simple
You can implement one of the following two solutions:

1. Solution running in time $O(n^2)$ - iterate over all possible pairs x_1, x_2 of elements between 0 and n, and checking if $f(x_1) = f(x_2)$.
2. Solution running in time $O(n)$ and space $O(n)$: create a `reverse_f` witn n elements. Iterate over all x in the range 0 to n, for each value $f(x)$ store in the array that `reverse_f[f(x)] = x`. Once you encounter collision, report it.

Slightly harder
Implement the following algorithm running in time $O(n)$ and space $O(1)$:
3. Start with $x_0 = n$, and $y_0 = n$.
4. In each iteration compute $x_{k+1} = f(x_k)$ and $y_{k+1} = f(f(y_k))$.
5. If $x_k = y_k$ at some point, we know that $f^{2k}(n) = f^k(n)$.
6. This means that there is a smallest t, such that $f^{k+t}(n) = f^t(n)$; a pair $x = f^{t-1}(n)$ and $y = f^{k+t-1}(n)$ will cause a collision we are looking for.
7. To find it, set $x = n$, and $y = f^k(n)$, and then iterate:
1. if $f(x) = f(y)$ you can return pair (x, y)
2. otherwise set $x = f(x)$ and $y = f(y)$.

Example

```python
N = 100
def my_fun(x):
    return (3*(x**3) + 2*(x**2) + 1)%N

(a, b) = find_collision(N, my_fun) # for example (10, 0)
if my_fun(a) == my_fun(b):
    print("Correct!")
```

Another example:

```python
N = 100
(a,b) = find_collision(N, lambda x: x % N)
if a%N == b%N:
    print("Correct!")
```

Problem 4

Write a function `all_combinations(n, m)` that outputs a list of strings of length $n + m$ -- all possible strings with exactly n letters 'a' and m letters 'b'. The order of the words in the list does not matter.

Example

```python
sorted(all_combinations(3, 2)) == ['aaabb', \
 'aabab', \
 'aabba', \
 'abaab', \
 'ababa', \
 'abbaa', \
 'baaab', \
 'baaba', \
 'babaa', \
 'bbaaa']
```