Programming

You are given 3 questions in this section. Answer **two questions of your choice**. Clearly mark your chosen questions. (Otherwise, we will grade the first two questions you submit.)

Instructions to submit your solutions:

1. The files `QE_prob1.py`, `QE_prob2.py`, and `QE_prob3.py` contain your solution to problems 1, 2, and 3, respectively.
2. Remove any debugging or logging code before you submit. It may disturb the automatic grading process, and as a result, you will likely get a lower score.
3. Compress the three files `QE_prob1.py`, `QE_prob2.py`, and `QE_prob3.py` to a single submission `file 20XX_XXXXX.zip` (`20XX_XXXXX` is your SNU student id, *e.g.*, 2020_12345.zip). The submission file should contain at most three files: `QE_prob1.py`, `QE_prob2.py`, and `QE_prob3.py`.
4. Send the submission file to `gsds_qe@aces.snu.ac.kr` from your SNU email account (if it is not an SNU email account, we will not accept your solution). The title of the submission email should be `[QE] 20XX-XXXXX` (*e.g.*, `[QE] 2020-12345`).
5. Make sure that the attached file is easily downloadable from the email message. We will not accept any submission that requires third-party tools or storages (*e.g.*, Google Drive).

Note: You may use the Internet for API search, but communication with other people in any matter is strictly prohibited. Violation to this will be considered as academic misconduct.

1. Given an integer `array` of size N. Implement a method that returns the smallest positive integer which is not in that array. To get full credit, your implementation needs to run in $O(N)$. (That is, the run time *linearly* increases *only* to the array size, regardless of the element values.) Do not use the built-in `sort` method. If needed, build it on your own.

- Example 1) array = [4, 7, -1, 1, 2] $\rightarrow$ return: 3
- Example 2) array = [100, 101, 102] $\rightarrow$ return: 1
- Example 3) array = [3, 2, 1, 0, -1] $\rightarrow$ return: 4

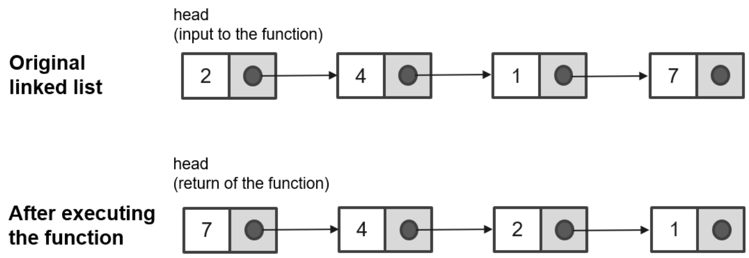
Please implement the `smallest_pos_int(array)` in the file `QE_prob1.py`.

2. There is a singly linked list where each node is `LinkedNode` defined below.

```
class LinkedNode:
  def __init__(self, x):
    self.val = x
    self.next = None
```

Given the head (first node) of the linked list, sort the linked list in descending order (`val` element) and return the head of the sorted linked list. To get full credit, your implementation needs to run in $O(N \log N)$ with $O(1)$ memory space. Do not use the built-in `sort` method. If needed, build it on your own.

- Example)



Please implement the following `sortingLL(head)` in the file `QE_prob2.py`.

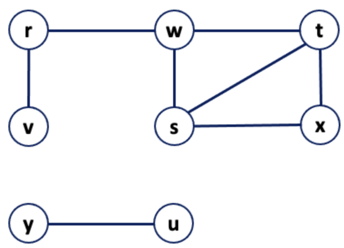
```
def sortingLL(head: LinkedNode) -> LinkedNode:
  # Your Code
```

3. In this problem, you will implement a function that finds out the lexicographically smallest path between two nodes in a graph. A node in an undirected graph is defined as follows (do not modify the node definition in your solution):

```
class GNode:
  def __init__(self, id, color="W";, d=-1, f=-1, p=None):
    self.id = id          # id is a string
    self.color = color    # color (status) of node
    self.d = d            # discover time of node
    self.f = f            # finish time of node
    self.parent = p       # predecessor time of node

  def __str__(self):
    return self.id
```

Consider an adjacency list implementation of an undirected graph using a dictionary. You should implement a function `LexSmallest(G, u, v)` that takes an undirected graph `G` and two vertices `u` and `v` as arguments. It returns the lexicographically smallest path between `u` and `v`. The list contains the id of the nodes on the path (*i.e.*, it is a list of strings). If there is no path between `u` and `v`, `LexSmalles(G, u, v)` returns an empty list. For example, consider the following graph `G`:



You can create `G` in an adjacency list implementation using a dictionary:

```
>> r, s, t, u = GNode('r'), GNode('s'), GNode('t'), GNode('u')
>> v, w, x, y = GNode('v'), GNode('w'), GNode('x'), GNode('y')
>> G = dict()
>> G[r], G[w], G[t], G[u] = [w, v], [s, r, t], [s, x, w], [y]
>> G[v], G[s], G[x], G[y] = [r], [w, t, x], [s, t], [u]
```

Then some behaviors of `LexSmallest()` are as follows:

```
>> LexSmallest(G, t, v)
['t', 's', 'w', 'r', 'v']
>> LexSmallest (G, u, u)
['u']
>> LexSmallest(G, w, y)
[]
```

The lexicographic order (also known as dictionary order) is a generalization of the alphabetical order of the dictionaries to sequences of ordered symbols. For two lists of strings `P1` and `P2`, `P1` is lexicographically smaller than `P2` if and only if there exists an integer k such that `P1[0] = P2[0]`, `P1[1] = P2[1]`, ..., `P1[k-1] = P2[k-1]`, and `P1[k] < P2[k]`. In the example graph `G`, the lexicographically smallest path between `t` and `v` is `[‘t’, ‘s’, ‘w’, ‘r’, ‘v’]`. The returned list must include every vertex in the lexicographically smallest path. Your solution should be based on the Depth-First Search (DFS) algorithm.