Version 2- DSCI-D 321 Assignment 1 Assigned Wednesday, January 18, 2023
Instructor: Dirk Van Gucht Due Friday, January 27, 2023 before 11:45pm

I have made a few corrections to the original Assignment 1 document. All changes are indicated in red

- This is a programming assignment with all code to be written in Python.
- To arrive at yours solution, you can use the appropriate Python code provided in the `lectureCode.py` file.
- You will need to upload your solutions in IU Canvas as a file with named

  `yourNameAssignment1.py`

  So if I were the submit my solutions file, it would be called

  `dirkVanGuchtAssignment1.py`

  This file should also include the name of each student who you have discussed this assignment with.[1]

  Note that you can discuss the assignment but that you will need to submit the solutions in your own writing.
- So that we can more easily read your solutions, provide comments that go along with your Python code.
- Submit a text file `yourNameAssignment1.txt` that provides examples of running your solutions on some test cases.

[1] Actually, to better understand the material, it is best that you can solve each problem on your own.

1. **Problem 1 has been is entirely eliminated from this assignment. It need not be solved.**

2. Consider the recursively defined `mergeSort` function

```
def mergeSort(A):
    if length(A) <= 1: return(A)
    return Merge(mergeSort(leftHalf(A)), mergeSort(rightHalf(A)))
```

   Develop a non-recursive implementation of this function.

   Note that your solution will necessarily require the use of a looping statement.

   Test cases:

   1. $A = []$
   2. $A = [1]$
   3. $A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$
   4. $A = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$
   5. $A = [7, 6, 6, 3, 9, 3, 2, 4, 8, 10, 7, 5, 1, 5, 1, 5, 3, 2, 3, 3, 3, 3, 1, 2]$

3. Consider unordered lists wherein elements may occur multiple times. For example,

$$[5, 3, 7, 5, 5, 1, 2, 10, 3, 3, 3, 2, 3, 4, 7, 1, 1]$$

   Write a Python function `frequencyCount(A)` that takes as input an unordered list A and return the list of pairs (a, n) where a is an element in A that occur n times. Your algorithm should run in $O(|A| \log_2 |A|)$.

   So, applied to the above list, `frequencyCount(A)` should return the list

$$[(1, 3), (2, 2), (3, 5), (4, 1), (5, 3), (7, 2), (10, 1)]$$

   Test cases:

   1. $A = []$
   2. $A = [1]$
   3. $A = [1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1]$
   4. $A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$
   5. $A = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$
   6. $A = [5, 3, 7, 5, 5, 1, 2, 10, 3, 3, 3, 2, 3, 4, 7, 1, 1]$
   7. $A = [7, 6, 6, 3, 9, 3, 2, 4, 8, 10, 7, 5, 1, 5, 1, 5, 3, 2, 3, 3, 3, 3, 1, 2]$

4. Consider a Binary Tree that stores a set of **different** elements. **So there are not duplicate elements.**

   - Write a function `Descendant(element, tree)` that returns a list of all the descendants of the element in the tree.

     Use the assumption that each element is considered a descendant of itself. You can think about it as a descendant of the element at distance 0.

   - Write a function `Ancestors(element, tree)` that returns a list of all the ancestors of the element in the tree.

     Use the assumption that each element is considered an ancestor of itself. You can think about it as a ancestor of the element at distance 0.

     Test cases:

     1. Consider the binary tree that occurs in the lecture nodes.
        Compute, for each element in this tree, its descendants and its ancestors.
     2. Consider the binary search tree that occurs in the lecture nodes.
        Compute, for each element in this binary tree, its descendants and its ancestors.

5. Consider the Binary Search Tree (BST) data structure.

   Let A and B be unordered lists that have no duplicates.

   1. Using the BST data structure, write a function `Intersect(A,B)` that returns the list of all the elements that occur in both A and B.
   2. Using the BST data structure, write a function `Difference(A,B)` that returns the list of all the elements that occur in A but not in B.
   3. Using the BST data structure, write a function `equalSet(A,B)` that returns True if and only if A and B have the same elements.

      So, for example

      ```
      equalSet([1,5,2,7,9],[2,5,9,1,7])
      ```

      should return `True`, whereas

      ```
      equalSet([1,5,2,7,9],[2,6,9,1])
      ```

      should return `False`

   Test cases:

   1. $A = []$, $B = []$
   2. $A = [1, 2, 3]$, $B = []$
   3. $A = []$, $B = [1, 2, 3]$
   4. $A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$, $B = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$

5. $A = [10, 3, 2, 5, 4, 7, 6, 9, 8, 10, 1]$, $B = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$
6. $A = [1, 3, 2, 7, 8, 9, 10, 4]$, $B = [10, 9, 7, 6, 5, 3, 2, 0, 11]$

6. In the lecture notes, we described the `hashTable` data structure.

   Implement this data structure. In particular implement the `isIn(element,A)`, `Insert(element,A)` and `Delete(element,A)` functions for this data structure.

   Your hash table should be set up as an array with indexes in the range $[0, p-1]$ where p is the divisor specified in the hash function

   $$h(\mathbf{k}) = \mathbf{k} \bmod p.$$

   Test cases: Choose $p = 3$:

   1. Use the `Insert` function to put each of the following lists A in their corresponding hash table.
      (a) $A = []$
      (b) $A = [1]$
      (c) $A = [1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1]$
      (d) $A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$
      (e) $A = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$
      (f) $A = [5, 3, 7, 5, 5, 1, 2, 10, 3, 3, 3, 2, 3, 4, 7, 1, 1]$
      (g) $A = [7, 6, 6, 3, 9, 3, 2, 4, 8, 10, 7, 5, 1, 5, 1, 5, 3, 2, 3, 3, 3, 3, 1, 2]$
      Now consider the elements 1, 4, 8, 9, and 10 and determine, using the `isIn` function, whether or not these elements are in the hash table correspond to A.
   2. For each of the lists A above, create the correspond hash table. Then, using the `Delete` function, delete, one at a time, the elements from A that appear at an odd number position in A.

   `Repeat your tests for` $p = 5$.

7. Repeat Problem 3 using an implementation of a hash table with 7 buckets. You should use the hash function

   $$h(\mathbf{k}) = \mathbf{k} \bmod 7$$

   Your algorithm should run in $O(|A| \times$ average length of the buckets$)$.

   Test cases:

   1. $A = []$
   2. $A = [1]$
   3. $A = [1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1]$
   4. $A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$

5. $A = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$
6. $A = [5, 3, 7, 5, 5, 1, 2, 10, 3, 3, 3, 2, 3, 4, 7, 1, 1]$
7. $A = [7, 6, 6, 3, 9, 3, 2, 4, 8, 10, 7, 5, 1, 5, 1, 5, 3, 2, 3, 3, 3, 3, 1, 2]$

8. Repeat question 5 by using the Hash Table data structure.

   Test cases:

   Using $p = 4$.

   1. $A = []$, $B = []$
   2. $A = [1, 2, 3]$, $B = []$
   3. $A = []$, $B = [1, 2, 3]$
   4. $A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$, $B = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$
   5. $A = [10, 3, 2, 5, 4, 7, 6, 9, 8, 10, 1]$, $B = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$
   6. $A = [1, 3, 2, 7, 8, 9, 10, 4]$, $B = [10, 9, 7, 6, 5, 3, 2, 0, 11]$