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

- 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. Consider the Binary Search Tree data structure and the recursively defined `insert` function

```
def insert(element,tree):
  if isEmpty(tree):
     return makeTree(element, emptyTree(), emptyTree())
  if element < root(tree):
     return makeTree(root(tree),
                     insert(element,left(tree)),
                     right(tree))
  else: return makeTree(root(tree),
                        left(tree),
                        insert(element,right(tree)))
```

   Develop a non-recursive, iterative, implementation of this function.

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

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.

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

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

4. Consider a Binary Tree that stores a set of elements.

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

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`

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

   Implement this data structure. In particular implement the `isIn`, `Insert` and `Delete` 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

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

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 $O(|A| \times \text{average length of the buckets})$.

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