COEN 352 Fall 2023 Assignment 2

This assignment further develops the Warehouse database based on Assignment 1. Please refer to Assignment 1 for the inventory Warehouse database description. We will refer to the same dataset as in Assignment 1.

In the database technology, the term *index* is a data structure to query and retrieve data with improved speed. For example, based on the *Inventory Values*, the inventory records can be sorted. However, sorting directly on the original database will result in exchange of elements at positions. Even worse, if the query changes to other field such as *Unit Price*, then the sort will shuffle the positions of the records again according the new requested field. This may lead to chaos of the datasets and low efficiency.

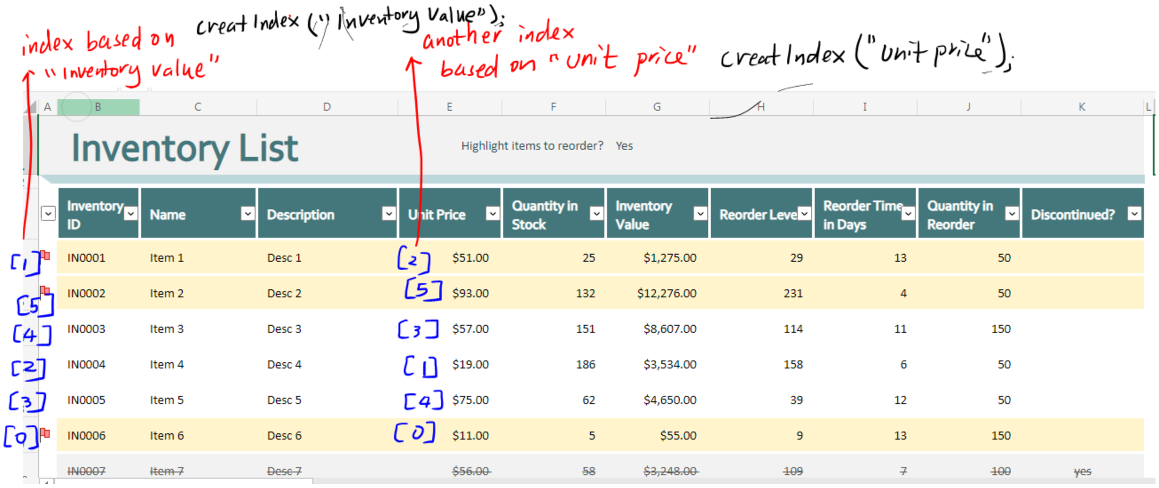
The solution is to build **an index** for a specific query. An index on the field of *Inventory Values* is a data structure that keeps the position of records (i.e. each row of the spreadsheet) that are sorted, such as the sorted **Inventory Value** of each record. The index does not contain a record itself, but only the position in the sorted order.

Problem 1. Add one new member function to the Dictionary ADT as below. Revise your implementation of Dictionary and program the member function to your Dictionary class.

1.1 (20 Marks) Select just one of Dictionary implementation such as Array-based or Linked List or Double Linked List based dictionary to implement the method.

```
public E[] createListIndex(String attribute); or
```

The return is an **E[]** follows either ascending or descending order of the original records in the Dictionary. The original order of the records in the WarehouseDB SHOULD NOT be changed. An illustration is shown in the Figure 1 below. ***The index created is the sorted position rather than swapping records to the sorted position. That means the original records remain unchanged. Please think the solution to achieve this requirement, and discuss your solution's asymptotic analysis in the function's comments on the time complexity.***



index based on "Inventory value"

creatIndex ("Inventory Value");

another index based on "unit price"

creatIndex ("unit price");

Inventory List

Highlight items to reorder? Yes

	Inventory ID	Name	Description	Unit Price	Quantity in Stock	Inventory Value	Reorder Level	Reorder Time in Days	Quantity in Reorder	Discontinued?
[1]	IN0001	Item 1	Desc 1	[2] $51.00	25	$1,275.00	29	13	50	
[5]	IN0002	Item 2	Desc 2	[5] $93.00	132	$12,276.00	231	4	50	
[4]	IN0003	Item 3	Desc 3	[3] $57.00	151	$8,607.00	114	11	150	
[2]	IN0004	Item 4	Desc 4	[1] $19.00	186	$3,534.00	158	6	50	
[3]	IN0005	Item 5	Desc 5	[4] $75.00	62	$4,650.00	39	12	50	
[0]	IN0006	Item 6	Desc 6	[0] $11.00	5	$55.00	9	13	150	
	~~IN0007~~	~~Item 7~~	~~Desc 7~~	~~$56.00~~	~~58~~	~~$3,248.00~~	~~109~~	~~7~~	~~100~~	~~yes~~

One way to create the index is to create a temporary storage to make a copy of the values of the attribute of interest, such as E[]. The next step is to do sorting on E[] to get each element's sorted position. Please select a sorting algorithm in Big-Theta(n log n) in the average case to sort the index in the ascending order or descending order based on an attribute.

1.2 (20 Marks) Build the index as a Binary Search Tree-based data structure and use the produce the ascending order. Please first make the choice of which traversing (pre-order, in-order, post-order) could produce ascending order

```
public BSTNode<E> createTreeIndex(String attribute);
```

Problem 2. (20 marks) In your WarehouseInventory class, please develop a query method based on the query string as the attributes of the inventory list. The query method print the inventory ID of the **top-N** ranking inventory records according to the attribute.

```
public void query(String attribute, int top);
```

The pre-condition is that the attribute should have an index created. If the value of parameter top = 5, it means top-5 ranking.

Problem 3. (20 marks) Please develop two unit test cases on query two different attributes for the method query(). Two unit tests should produce PASS results. This problem needs to be solved using unit testing techniques, not main() function based solutions.

```
public void query(String attribute, int top);
```

Problem 4. (10 Marks) Please write a 1-2 page report of your asymptotic analysis for your programming solution in problem 1.1 and 1.2 of creating index. The file is named as report.pdf

The report should be structured as follows based on your implementation. For any invocation based on released code of data structure and sorting algorithm, you can refer to the case of the data structure or algorithm's asymptotic analysis and skip the code segment.

Problem 1.1

- Code Segment 1 (screenshot)
- Code Segment 1 Analysis
- Code Segment 2 (screenshot)
- Code Segment 2 Analysis
- and so on
- Finally the final aggregation of the analysis

Problem 1.2

- Code Segment 1 (screenshot)
- Code Segment 1 Analysis
- Code Segment 2 (screenshot)
- Code Segment 2 Analysis
- and so on
- Finally the final aggregation of the analysis
-

Submission Specification

1. Program the solution in a single project, thus under one ***src*** folder
2. The src should contain all the Java files
3. The src folder and the report.pdf should be archived together as a single file, following the naming convention [SID_1]_[SID_2]_A2.zip or
   [SID_1]_[SID_2]_A2.gz or
   [SID_1]_[SID_2]_A2.tar

No .rar file is accepted. Do not following the naming convention will cause delays in releasing the marking grade.

4. Grace period is 24 hours later. 20% penalty is 24 hours later. Submission is through Moodle site ONLY. Submission in emails is not accepted.