ICS Fall 2021
Assignment 4

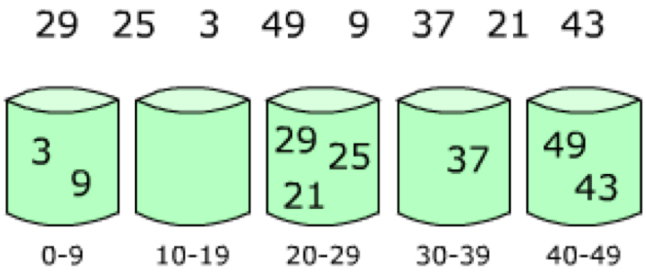
Please note: For some questions, starting codes are available. Please fill the blanks in starting codes. For questions without starting codes, you can write the solutions in any form you like. Since we use an autograder to grade your code, please strictly name the functions and the submitted files as required by the questions. Moreover, when you submit your answers, you should zip the files, not the whole folder of your answers. Lastly, the starting codes may contain some tests for you to check if your code is correct.

The Bucket Sort

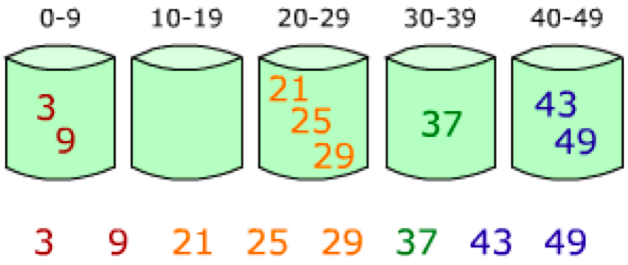
We have learned bubble sort and merge sort. Another very useful sorting algorithm in practice is bucket sort. The intuition is simple:

- You throw items into a set of buckets, each is responsible for a fixed and constant range
- Sort items within a bucket using another other sorting algorithms
- Go through the buckets in order, you are done!

Step 1: distribute items into the buckets



Step 2: sort items within the buckets



Implement bucket sort, we will sort a list of random integers range in 0~99, and will have every bucket responsible for exactly the same range. So, the first bucket takes numbers from 0 to 9, and second 10 to 19, and so on.

Exercise 1 – Using the procedural programming paradigm

The following hints are ways to implement bucket sort using the above intuition:

- Use the starting code provided from **bucket_sort_student.py**
- Use a dictionary to implement buckets, each bucket holds a list.
- Distribute items into the list in each bucket. In our case, if x is a number in the list, x//10 will find its bucket (Actually, here 10 is the size of a bucket). If the bucket doesn't exist, start a list with just x, otherwise append to the list.
- For each bucket, sort its list; you can use built-in list sort, i.e. `mylist.sort()` will sort `mylist` in ascending order
- Then go through the dictionary in order of keys, using `sorted(mydict.keys())`.

Exercise 2 – Using the OOP programming paradigm

The following hints are ways to implement bucket sort using OOP:

- Use the starting code provide from **bucket_sort_oop_student.py**
- define a `Bucketsort` class: it should have 3 attributes
  - `sizeBucket`: the size of a bucket (in Exercise 1, this value is 10, but actually, we can set it as any positive integer).
  - `buckets`: a dictionary
  - `data`: the data need to be sorted, initialized as None
- The `Bucketsort` class should contain 3 methods
  - `__init__(self, sizeBucket)`: this function initializes the class.
  - `_distributeElementsIntoBuckets(self)`: this function distributes the data into the `self.buckets`. In our case, if x is a number in the `self.data`, x//sizeBucket will find its bucket. If the bucket doesn't exist, start a list with just x, otherwise append to the list.
  - `sort(self, inputList)`: this function sorts the elements in each bucket, merges the buckets, and returns the sorted list.

Note: we can also hide the method in a class by starting the method's name with a "_" (i.e., weakly hidden) or "__"(strongly hidden). A hidden method means it is designed to be used only within the class (i.e., it is only called by methods defined in the class), not for being called outside of that class.

Exercise 3 - OOP Basic: the Employee Class (please use the start code given)

(a) Employee class (Please complete the employee_class_student.py)

In practice, it is common to make all of a class's data attributes private and to provide public methods for accessing and changing those attributes. This ensures that the objects owning those attributes are in control of all the changes being made to them.

A method that returns a value from a class's attribute but does not change it is known as an accessor method (sometimes they are called "**getters**"). It provides a safe way for code outside the class to retrieve the values of attributes, without exposing the attributes in a way that they could be changed by the code outside the method.

A method that stores a value in a data attribute or changes the value of a data attribute in some other way is known as a mutator method (sometimes they are called "**setters**"). Mutator methods can control the way that a class's data attributes are modified. When a code outside the class needs to change the value of an object's data attribute, it typically calls a mutator and passes the new value as an argument. If necessary, the mutator can validate the value before it assigns it to the data attribute.

Write a class named `Employee` that holds the following data about an employee in attributes: `name, ID number, department,` and `job title`. In this exercise, you have to use the setters/getters to assign/obtain the values to the attributes.

Once you have written the class, write a program that creates three Employee objects to hold the following data:

Name	ID Number	Department	Job Title

Susan Meyers	47899	Accounting	Vice President
Mark Jones	39119	IT	Programmer
Joy Rogers	81774	Manufacturing	Engineer

The program should store this data in the three objects and then display the data for each employee on the screen.

(b) Employee management system (Please complete the `employee_management_student.py`)

Create a program that stores Employee objects in a dictionary; use the employee ID number as the key. The program should present a menu that lets the user perform the following actions:

- Look up an employee in the dictionary
- Add a new employee to the dictionary
- Change an existing employee's name, department, and the job title in the dictionary
- Delete an employee from the dictionary
- Quit the program

When the program ends, it should pickle the dictionary and save it to a file (named, `emp_database.dat`). Each time the program starts, it should try to load the pickled dictionary from the file. If the file does not exist, the program should start with an empty dictionary.

When the program is running, it should look like the following

```
1.Look up
2.Add new employee
3.Change employee information
4.Delete employee
5.Quit
Please select an operation: 1

Please given the ID: 81774
ID: 81774
Joy Rogers
Manufacturing
Engineer

1.Look up
2.Add new employee
3.Change employee information
4.Delete employee
5.Quit
Please select an operation: 5
```

Your task is to write the code for option 3 and 4.

Note: In this management system, you need to use the `Employee class`. You should import the `employee_class_student.py`.