Practical Task 3.4

(Distinction Task)

Submission deadline: 10:00am Monday, April 20
Discussion deadline: 10:00am Saturday, May 2

General Instructions

Bucket Sort is an ***algorithm*** that sorts data by distributing the elements of an array into a number of ***buckets***. Each bucket is then sorted individually, either using a different sorting algorithm, or by recursively applying the Bucket Sort. It is normally fed with two arguments: a number of available buckets and a collection of elements to sort. The sorting process can be formally represented by the following pseudocode.

```
function BucketSort(b, array)
{
    buckets ← new array of b empty lists
    M ← the maximum key value in the array
    for i = 1 to length(array) do
        insert array[i] into buckets[floor(b × array[i] / M)]
    for i = 1 to b do
        nextSort(buckets[i])
    return the concatenation of buckets[1], ...., buckets[b]
}
```

Here, **`array`** represents a collection of data elements and **`b`** is a given number of buckets to use. The ***maximum key*** value can be computed sequentially (that is, in linear time) by looking up all the elements once. The **`floor`** function must be used to convert a ***floating number*** to an ***integer***. The function **`nextSort`** is a sorting function used to sort each bucket individually. Conventionally, ***Insertion Sort*** would be used, but other algorithms could be used as well, e.g. the **`BucketSort`** function itself (applied recursively).

In this practical task, you need to implement the Bucket Sort algorithm as suggested by the above pseudocode. For this purpose, start a new C# Console Application project and import the current version of the `Account` class that you have completed in Task 3.2. You must augment the `Account` class with one more property required to get the current balance of the account; that is

- **decimal Balance**
  Property. Gets the `_balanace` of the account.

Subsequently, create a new ***static*** class called `AccountsSorter`. You must ensure that the following two static methods in this class can be accessed by a user:

- **void Sort(Account[] accounts, int b)**
  Sorts the specified one-dimensional array of accounts in the ascending order of their balances. Each element of `accounts` is an instance of the class `Account`.
- **void Sort(List<Account> accounts, int b)**
  Sorts the specified collection of accounts in the ascending order of their balances. Each element of `accounts` is an instance of the class `Account`.

Note that you may implement the required `nextSort` function of the pseudocode either by calling the respective version of the `Sort` method from the above list or by delegating this task to a respective native method of the .NET Framework, e.g. `Array.Sort` or `List<T>.Sort`.

Complete and test your program for potential logical issues and runtime errors within the `Program` class of the project. When testing your class, think about special cases where your methods might fail, for example when the given collection is `null`.

Further Notes

- To get a better understanding of a data collection as a concept and how to apply it in your programs, read Section 3.2 of SIT232 Workbook available in CloudDeakin → Resources.
- The following links will give you more insights about arrays, lists, and the way to use them:
  - https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/arrays/
  - https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/collections
  - https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1
- Explore the links below to get details on the standard sorting methods provided by .NET Framework as part of the Array and the List<T> classes.
  - https://docs.microsoft.com/en-us/dotnet/api/system.array.sort
  - https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1.sort
- Information on the Bucket Sort available in Wikipedia should be sufficient to get enough details about the sorting process it performs.
  - https://en.wikipedia.org/wiki/Bucket_sort

Marking Process and Discussion

To get your task completed, you must finish the following steps strictly on time.

- Make sure that your program implements the required functionality. It must compile and have no runtime errors. Programs causing compilation or runtime errors will not be accepted as a solution. You need to test your program thoroughly before submission. Think about potential errors where your program might fail.
- Submit the expected code files as an answer to the task via OnTrack submission system. Cloud students must record a short video explaining their work and solution to the task. Upload the video to one of accessible resources, and refer to it for the purpose of marking. You must provide a working link to the video to your marking tutor in OnTrack.
- On-campus students must meet with their marking tutor to demonstrate and discuss their solution in one of the dedicated practical sessions. Be on time with respect to the specified discussion deadline.
- Answer all additional questions that your tutor may ask you. Questions are likely to cover lecture notes, so attending (or watching) lectures should help you with this **compulsory** interview part. Please, come prepared so that the class time is used efficiently and fairly for all students in it. You should start your interview as soon as possible as if your answers are wrong, you may have to pass another interview, still before the deadline. Use available attempts properly.

Note that we will not accept your solution after the submission deadline and will not discuss it after the discussion deadline. If you fail the deadline, you also fail the task and this may impact your performance and your final grade in the unit. Unless extended for all students, the deadlines are strict to guarantee smooth and on-time work throughout the unit.

Remember that this is your responsibility to keep track of your progress in the unit that includes checking which tasks have been marked as completed in the OnTrack system by your marking tutor, and which are still to be finalised. When marking you at the end of the unit, we will solely rely on the records of the OnTrack system and feedback provided by your tutor about your overall progress and quality of your solutions.