School of Infocomm Technology | Applied Artificial Intelligence, BSc (Hons) | INF1008 - Data Structure & Algorithms | Project 2
P1T7 | Ashlinder Kaur, 2202636 | Leo En Qi Valerie, 2202795 | Osama Rasheed Khan, 2203385 | Seri Hanzalah Bte Haniffah, 2201601 | Teo Xuanting 2202217

TASK 1

Introduction: Finding the Median Phone Number

The objective of the assignment is to determine the median 10-digit phone number from a provided list. It may include phone numbers with different formats. A suitable sorting algorithm has been implemented for the identification of the median number after sorting the list of numbers.

Assumptions:

1. The input may contain the country code prefix (e.g. +1 or 1)
2. The input numbers are always in a minimum of 10 digits (e.g. 3-digit area code and 7-digit number).
3. The input numbers may contain whitespace inside.
4. The input numbers list might contain duplicates
5. The input numbers list does not contain any alphabet.

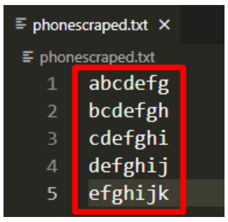
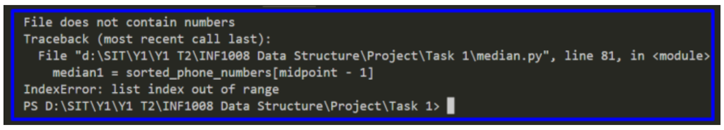
Expected output:

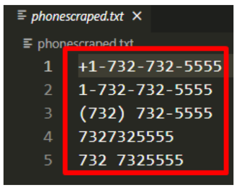
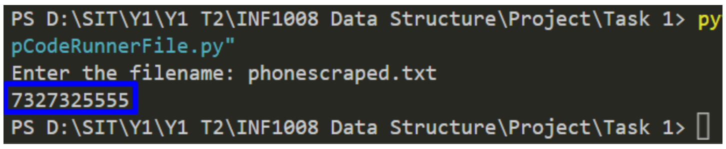
1. If the length of the sorted list is odd, it prints the middle number.
2. If the length of the sorted list is even, it prints the two middle numbers separated by a comma.

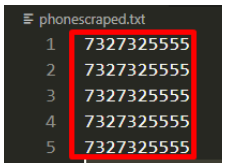
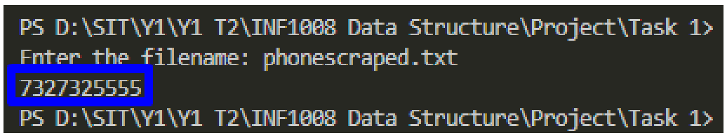
How to Compile / Run: Refer to the README.md file.

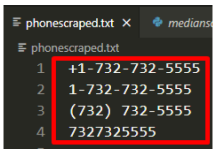
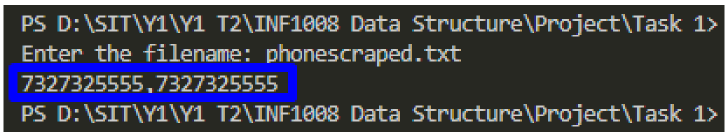
Testing Strategy:

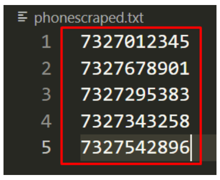
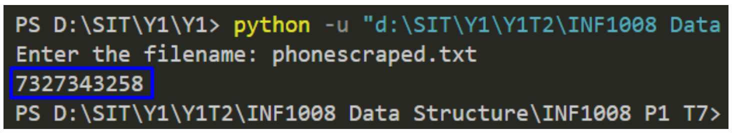
INDEX	INPUT	OUTPUT

1	**Alphabetical**
	[phonescraped.txt]
	
	phonescraped.txt ×
	phonescraped.txt
	`abcdefg`
	`bcdefgh`
	`cdefghi`
	`defghij`
	`efghijk`
		[Terminal Output]
		
		`File does not contain numbers`
		`Traceback (most recent call last):`
		`  File "d:\SIT\Y1\Y1 T2\INF1008 Data Structure\Project\Task 1\median.py", line 81, in <module>`
		`    median1 = sorted_phone_numbers[midpoint - 1]`
		`IndexError: list index out of range`
		`PS D:\SIT\Y1\Y1 T2\INF1008 Data Structure\Project\Task 1>`
		Ans: File does not contain numbers
		IndexError: list index out of range (**PASS**)
		
		Expected error as alphabets are not accepted

2	**Given Default Sample Input**
	[phonescraped.txt]
	
	phonescraped.txt ×
	phonescraped.txt
	`+1-732-732-5555`
	`1-732-732-5555`
	`(732) 732-5555`
	`7327325555`
	`732 7325555`
		[Terminal Output]
		
		`PS D:\SIT\Y1\Y1 T2\INF1008 Data Structure\Project\Task 1> py`
		`pCodeRunnerFile.py"`
		`Enter the filename: phonescraped.txt`
		`7327325555`
		`PS D:\SIT\Y1\Y1 T2\INF1008 Data Structure\Project\Task 1>`
		Ans: 7327325555 (**PASS**)

3	**Duplicate Values**
	[phonescraped.txt]
	
	phonescraped.txt
	`7327325555`
	`7327325555`
	`7327325555`
	`7327325555`
	`7327325555`
		[Terminal Output]
		
		`PS D:\SIT\Y1\Y1 T2\INF1008 Data Structure\Project\Task 1>`
		`Enter the filename: phonescraped.txt`
		`7327325555`
		`PS D:\SIT\Y1\Y1 T2\INF1008 Data Structure\Project\Task 1>`
		Ans: 7327325555 (**PASS**)

4	**Even number of phone numbers**
	[phonescraped.txt]
	
	phonescraped.txt × medians
	phonescraped.txt
	`+1-732-732-5555`
	`1-732-732-5555`
	`(732) 732-5555`
	`7327325555`
		[Terminal Output]
		
		`PS D:\SIT\Y1\Y1 T2\INF1008 Data Structure\Project\Task 1>`
		`Enter the filename: phonescraped.txt`
		`7327325555,7327325555`
		`PS D:\SIT\Y1\Y1 T2\INF1008 Data Structure\Project\Task 1>`
		Ans: 7327325555,7327325555 (**PASS**)

5	**Random Integer values**
	[phonescraped.txt]
	
	phonescraped.txt
	`7327012345`
	`7327678901`
	`7327295383`
	`7327343258`
	`7327542896`
		[Terminal Output]
		
		`PS D:\SIT\Y1\Y1> python -u "d:\SIT\Y1\Y1T2\INF1008 Data`
		`Enter the filename: phonescraped.txt`
		`7327343258`
		`PS D:\SIT\Y1\Y1T2\INF1008 Data Structure\INF1008 P1 T7>`
		Ans: 7327343258

Time-Complexity

We assess the time complexity of the code by examining the time complexity of **each function** in the code and the total number of execution times for each code function.

`normalise_number` function:

`O(n)`, where `n` is the length of the phone number, is the processing time required to remove non-digit characters from the phone number. If a country code is included in the phone number, the process of checking its length and removing it can be performed with constant time complexity. Hence, this function's temporal complexity is `O(n)`, in which `n` is the length of the phone number.

`mergesort` function:

The function applies a divide-and-conquer approach, splitting the input list into two halves until each has only one element. This process has a time complexity of `O(log n)`, with n being the list's length. The halves are sorted and merged, taking `O(n)` time for the entire combined list. The function includes a try-except block to skip non-numeric elements and merge the rest. The overall time complexity is `O(n log n)` for the input list length n.

Reading phone numbers from the file:

Since there are `n` lines (or phone numbers) in the file, the time complexity of reading each line and normalising each phone number is `O(n)`. Since there are `n` phone numbers in the file, the time complexity of reading and normalising it is `O(n)`.

Sorting normalised phone numbers:

Mergesort requires `O(nlogn)` time to sort the list of normalised phone numbers, where `n` is the total number of phone numbers in the list.

Finding the median phone number(s):

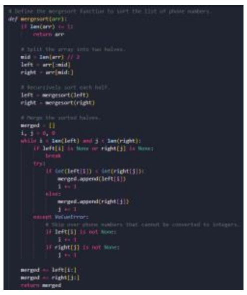
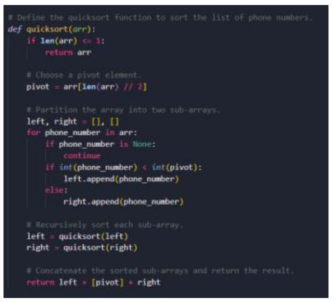
In order to find the median phone number(s), it takes a constant time of `O(1)`, regardless of the number of phone numbers. Therefore, the total time complexity of the code is the summation of the time complexity of reading the phone numbers from the file and the time complexity of sorting normalised phone numbers. Where `n` is the total number of phone numbers in the file:

`O(n) + O(n log n) = O(n log n)`

Data Structure

The code uses Python's built-in **list data structure** to store phone numbers read from a file. These phone numbers are normalised using a function and stored in a separate list, which is sorted using mergesort. The median phone number(s) is then determined from the sorted list.

The team chose merge sort for finding the median due to its `O(n log n)` worst-case time complexity, which is more efficient than quicksort's `O(n^2)` worst-case time complexity for large datasets. Merge sort is also a stable sorting algorithm that preserves the order of equal elements, ensuring precise median calculation. Furthermore, its "divide-and-conquer" strategy simplifies implementation and comprehension compared to Quicksort's intricate partitioning algorithm. Hence, merge sort was preferred for its efficiency, accuracy, and ease of use.

MERGE SORT	QUICK SORT
	
● Split array into halves recursively ● Sorts each half & merge the sorted halves	● Choose pivot element ● Partitions array into 2 sub-arrays based on pivot ● Recursively sorts each sub-array ● Concatenates sorted sub-arrays

```python
# Define the mergesort function to sort the list of phone numbers.
def mergesort(arr):
    if len(arr) <= 1:
        return arr

    # Split the array into two halves.
    mid = len(arr) // 2
    left = arr[:mid]
    right = arr[mid:]

    # Recursively sort each half.
    left = mergesort(left)
    right = mergesort(right)

    # Merge the sorted halves.
    merged = []
    i, j = 0, 0
    while i < len(left) and j < len(right):
        if left[i] is None or right[j] is None:
            break
        try:
            if int(left[i]) < int(right[j]):
                merged.append(left[i])
                i += 1
            else:
                merged.append(right[j])
                j += 1
        except ValueError:
            # Skip over phone numbers that cannot be converted to integers.
            if left[i] is not None:
                i += 1
            if right[j] is not None:
                j += 1

    merged += left[i:]
    merged += right[j:]
    return merged
```

```python
# Define the quicksort function to sort the list of phone numbers.
def quicksort(arr):
    if len(arr) <= 1:
        return arr

    # Choose a pivot element.
    pivot = arr[len(arr) // 2]

    # Partition the array into two sub-arrays.
    left, right = [], []
    for phone_number in arr:
        if phone_number is None:
            continue
        if int(phone_number) < int(pivot):
            left.append(phone_number)
        else:
            right.append(phone_number)

    # Recursively sort each sub-array.
    left = quicksort(left)
    right = quicksort(right)

    # Concatenate the sorted sub-arrays and return the result.
    return left + [pivot] + right
```

TASK 2

Introduction: Finding the K Unique Nearest Numbers

The task of this assignment is to find the K unique numbers nearest to the given 10-digit number after sorting the list in Task 1. The programme will take in two additional command-line arguments: a 10-digit number that starts with a non-zero digit and an integer value k. We have implemented an algorithm to search for the k nearest number(s) to the target number.

Assumptions:

1. The phone number in text file are one per line
2. The phone number has a minimum of 10 digits
3. If there are duplicates for the k closest number, all of these duplicates in text file are printed to the output

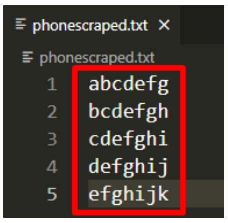
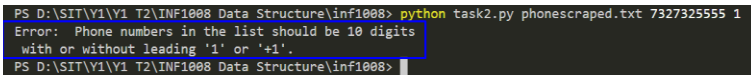
Expected Output:

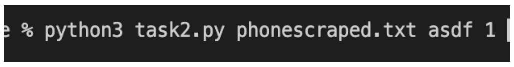
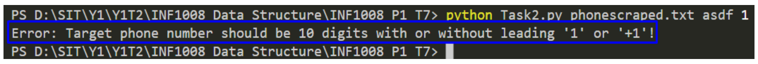
1. Print out the K unique numbers nearest (in terms of numerical difference) to the given 10-digit target number
2. If errors are present in the text file or system argument input such as any invalid format or values, error message will print out with respect to the type of error and the program will halt

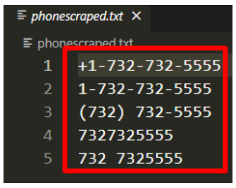
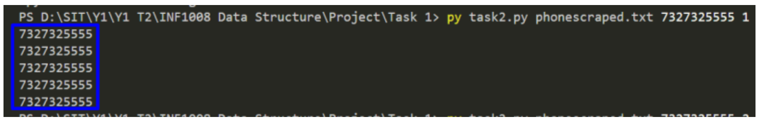
How to compile / Run: Refer to the README.md file.

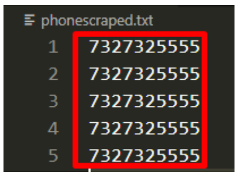
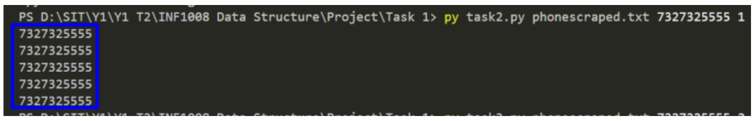
Testing Strategy:

INDEX	INPUT	OUTPUT

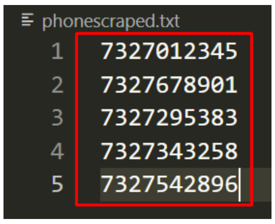
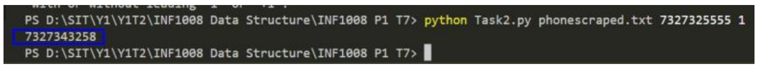
1	**Alphabetical**
	[phonescraped.txt]
	
	phonescraped.txt ✕
	phonescraped.txt
	`abcdefg`
	`bcdefgh`
	`cdefghi`
	`defghij`
	`efghijk`
		[Terminal Output]
		
		`PS D:\SIT\Y1\Y1 T2\INF1008 Data Structure\inf1008> python task2.py phonescraped.txt 7327325555 1`
		`Error:  Phone numbers in the list should be 10 digits`
		` with or without leading '1' or '+1'.`
		`PS D:\SIT\Y1\Y1 T2\INF1008 Data Structure\inf1008>`
		Ans: Error: Phone numbers in the list should be 10 digits with or without leading ‘1’ or ‘+1’.
		(**PASS**)
		Expected error as alphabets are not accepted in the list

2	**Target number input check**
	[terminal input]
	
	`e % python3 task2.py phonescraped.txt asdf 1`
		[Terminal Output]
		
		`PS D:\SIT\Y1\Y1T2\INF1008 Data Structure\INF1008 P1 T7> python Task2.py phonescraped.txt asdf 1`
		`Error: Target phone number should be 10 digits with or without leading '1' or '+1'!`
		`PS D:\SIT\Y1\Y1T2\INF1008 Data Structure\INF1008 P1 T7>`
		Ans: Error: Target phone number should be 10 digits with or without leading ‘1’ or ‘+1’!
		(**PASS**)
		Expected error as alphabets are not accepted as target number input

2	**Given Default Sample Input**
	[phonescraped.txt]
	
	phonescraped.txt ✕
	phonescraped.txt
	`+1-732-732-5555`
	`1-732-732-5555`
	`(732) 732-5555`
	`7327325555`
	`732 7325555`
		[Terminal Output]
		
		`PS D:\SIT\Y1\Y1 T2\INF1008 Data Structure\Project\Task 1> py task2.py phonescraped.txt 7327325555 1`
		`7327325555`
		`7327325555`
		`7327325555`
		`7327325555`
		`7327325555`
		Ans: 7327325555
		7327325555
		7327325555
		7327325555
		7327325555 (**PASS**)

3	**Duplicate Values**
	[phonescraped.txt]
	
	phonescraped.txt
	`7327325555`
	`7327325555`
	`7327325555`
	`7327325555`
	`7327325555`
		[Terminal Output]
		
		`PS D:\SIT\Y1\Y1 T2\INF1008 Data Structure\Project\Task 1> py task2.py phonescraped.txt 7327325555 1`
		`7327325555`
		`7327325555`
		`7327325555`
		`7327325555`
		`7327325555`
		Ans: 7327325555
		7327325555

		7327325555 7327325555 7327325555 (**PASS**)

4	**Random Integer Values**
	[phonescraped.txt]
	
	phonescraped.txt
	`7327012345`
	`7327678901`
	`7327295383`
	`7327343258`
	`7327542896`
		[Terminal Output]
		
		`PS D:\SIT\Y1\Y1T2\INF1008 Data Structure\INF1008 P1 T7> python Task2.py phonescraped.txt 7327325555 1`
		`7327343258`
		`PS D:\SIT\Y1\Y1T2\INF1008 Data Structure\INF1008 P1 T7>`
		Ans: 7327343258 (**PASS**)
		Shows the sorting of the phone numbers

Time-Complexity:

The time complexity of the code is analysed by looking at the time complexity of each function and loop in the code.

As mentioned in Task 1, the time complexity for `def normalise_number(phone_number)` function is `0(n)` and the time complexity for `def merge_sort(arr) function is O(nlogn)`

`closest_num_dict` function: The function has a time complexity of `O(nlogn)`. It creates a dictionary with keys as the absolute difference between the target number and each element in the input list in linear time `O(n)`. Then, it sorts the keys using the merge_sort function in `O(nlogn)`. Finally, it appends the corresponding values to the output list by iterating over the sorted keys in linear time `O(n)`.

The code's time complexity is O(n log n) for file input and phone number processing, sorting phone numbers, and finding the k closest numbers.

As a result, the `closest_num_dict` and `merge_sort` functions, both with a time complexity of O(n log n), dominate the overall time complexity, resulting in a total time complexity of O(n log n).

Data Structure:

The code utilises **lists as its primary data structure**. Phone numbers, sorted phone numbers, and output numbers are stored in lists. Dictionaries are also used to store key-value pairs for the difference between a given phone number and a target phone number, and the corresponding phone numbers. The merge sort algorithm divides and merges subarrays using lists. Integers are used to represent the target number and the value of k in a few instances.

REFERENCES

1. Sedgewick, R. (2013). Algorithms (4th ed.). Addison-Wesley.
2. GeeksforGeeks. (n.d.). Merge Sort. Retrieved March 17, 2023, from https://www.geeksforgeeks.org/merge-sort/
3. GeeksforGeeks. (n.d.). Quick Sort. Retrieved March 17, 2023, from https://www.geeksforgeeks.org/quick-sort/