Assignment 1: Part1

The Requirements Analysis, Design, and Test Plan for a Contact Management Application Software

1.) Application Outline

In the digital and connected world the amount of information pertaining to one's numerous personal and business contacts warrants a special-purpose application software, capable of facilitating the everyday storage and use of appropriate records (University of Essex Online, 2021a). Therefore, the purpose of the application covered in this document is to meet this requirement. The objectives of the program in mention would be to provide a few simple functions, namely:

- Add a contact and its respective details to main memory, specifically: contact name, phone numbers, and email address.
- Display a contact and its respective details.
- Delete a contact and its respective details from main memory.
- Search for a particular contact using a substring from the contact string required.

The program's source code will be written in Python's version 3.9.1 programming language, using Python's Integrated Development and Learning Environment (IDLE) version 3.9.1 program (Python Software Foundation, 2021a). Microsoft's Integrated Development Environment (IDE) Visual Studio Community 2019 may also be used, especially during testing and debugging (Microsoft, 2021a). Both Python's interpreter and IDE are present in the standard Python package downloadable from

https://www.python.org/downloads/, which is the official website for the Python development community (Python Software Foundation, 2021b; Python Software Foundation, 2021c). The mentioned software items are free from this platform due to Python being an open-source software (Python Software Foundation, 2021c). Although the program's source code is saved in a plain text format, the IDE creates a file extension of ".py" indicating to the Operating System (OS) to execute the code via the Python Interpreter program, which then parses and translates the high-level code in real-time to machine executable code (Python Software Foundation, 2021a; Microsoft, 2021b). For a user to be able to execute the application the installation of the correct Python interpreter is required, as the file is not compiled like in C and other languages which produce an executable (.exe) file (Swaroop, 2020). The interpreter program required will be based on the OS present on the user's machine, this makes the program portable as its ability to be executed is not based on the source code itself, but rather the correct interpreter program (Thaker & Shukla, 2020). Once the program file is executed, the process can be interacted with via a Command Line Interface (CLI) type User Interface (UI). A user will then follow the on-screen prompts to interact with the application. It is to be noted that as there is no external storage used to store the associated database of contacts, closing the application will remove its respective contents from the running memory, causing any variable values not hard-coded into the source code to be cleared (Brookshear & Brylow, 2018).

2.) Design of Data Structures and Algorithms

2.1) Data Structures

The imperative programming paradigm will be followed even though Python intrinsically stores variables as objects following an object-oriented paradigm (Python Software Foundation, 2021d). Various data structures will be used within the source code, from both the historically “Primitive” and “non-Primitive” high-level programming language abstractions (Brookshear & Brylow, 2018; University of Essex Online, 2021a). However, a single core structure housing the entire contact book will be in the form of a Python Dictionary, which is a dynamic structure that maps “keys” to “values” creating key-value pairs to fill the unit (Python Software Foundation, 2021d). This structure suits the situation almost perfectly as the keys assume contact names, and the values information pertaining to each name as seen in figure 1. Thus, allowing information lookups via name, rather than index position as in the case with lists.

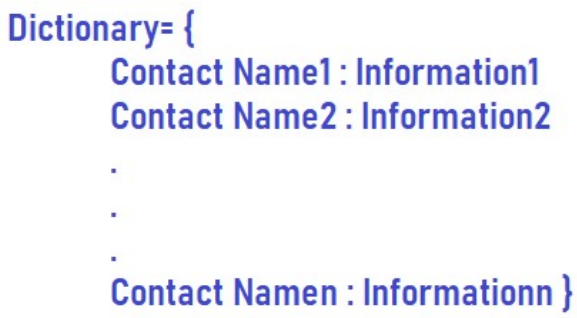

Figure 1 – Core Dictionary Structure

Within the Dictionary the primitive string data type will be used to represent contact names, as well as details. This will allow values to be stored directly from user inputs, and manipulated via the multitude of string and character operations, methods, and functions available in Python (University of Essex Online, 2021b). One last major structure housed by the dictionary will be a list. The list will assume the position of the

value within the dictionary, and provides the benefit of storing various fields in different index positions assisting information entry, segmentation, printing, and manipulation through iterative processing (University of Essex Online, 2021c).

```
Dictionary= {
        ' Name string1' : [ 'Field1', 'Field2',..., 'Fieldn]
        ' Name string2' : [ 'Field1', 'Field2',..., 'Fieldn]
        .
        .
        .
        ' Name stringn' : [ 'Field1', 'Field2',..., 'Fieldn] }
```

Figure 2 - Compound Dictionary

Further to the configuration of the dictionary, the structure will have a global scope allowing it to be accessible by all functions, as well as the main program body (University of Essex Online, 2021d). Although this may be a dangerous option to use because a faulty program module could destroy the entire core structure, it will suffice according to the program requirements (Brookshear & Brylow, 2018).

A local scope variable list will be used in certain functions, where searching through the dictionary keys is performed. In this case portion or all of the key strings from the dictionary will be copied to the list, then sorted and displayed alphabetically. A sequential search is necessary in this scenario not only because the designer has chosen to maintain the dictionary as an unsorted structure (Edureka, 2021), but also due to the difficulty in searching for a substring within each string throughout all contact names. When a key is found to contain the specific searched substring, the entire key string is appended to the list in mention. Lastly, there will be other primitives like

Booleans and Integers used for general flow control of the program (Brookshear & Brylow, 2018).

2.2) Algorithms

As previously mentioned, the imperative programming paradigm will be followed, therefore program modules in the form of functions will be created and called within the main body of the program, as well as from other functions (Brookshear & Brylow, 2018). This will assist to improve program readability, testing, isolation, and future modification (Brookshear & Brylow, 2018). In terms of program specifications such as the length of the source code, encapsulation and isolation requirements, and other considerations (Brookshear & Brylow, 2018), the design does not warrant a more sophisticated object-oriented approach. Although the program requires that only five contacts be housed the design will consider future expansion, yet within the bounds of approximately 500 contacts. This will determine the code efficiency requirements. Furthermore, because the number of contacts is small and the assumption is that the user has sufficient memory on the machine running the application, space complexity will not be a consideration (Erickson, 2019).

Beginning with the program's main body (figure 3), a tightly coded section will present options to the user, read the input provided, and direct the program to the associated function's code. This will loop infinitely unless an exception is raised or the user elects to terminate the program. Please note that some of the words in the Pseudocode are incorrectly highlighted, this is because they are Python keywords (Python Software

Foundation, 2021f). The context will however differentiate actual keywords from pseudocode.

```
while (User does not elect termination):
    print(Available options)

    selectVariable = read user selection

    if (selectVariable is option1):
        call function1
    elif (selectVariable is option2):
        call function2
    elif (selectVariable is optionn):
        call functionn
    elif (selectVariable is termination):
        break
    else:
        print(incorrect input)
        continue
```

Figure 3 - Program Main Body Pseudocode

In terms of the functions that will be coded, please see the following list:

- General sort abstraction for other functions.
- Add function to enable insertion of a new contact and respective details.
- Search function to search for and display a particular contact.
- Delete abstraction to delete a chosen contact.
- Display-all function to print all contacts stored.

The sort function will use an insertion sort algorithm which has a worst-case time complexity of $O(n^2)$ (Bender et al., 2006). Although a more scalable function response is desirable (Brookshear & Brylow, 2018), the small number of contacts specified make this inefficient order of complexity acceptable. Please see figure 4 below.

```
def insertion_sort(List):
    temporary_storage = None
    for iteration through List:
        temporary_storage = element_value
        while (the values before the temporary value are greater):
            shift the values down the List
            if (the comparison reaches the first element):
                break the loop
        Place the temporary_storage value back into the List
```

Figure 4 - Insertion Sort Pseudocode (Brookshear & Brylow, 2018).

The add function (Figure 5) will request the user enter each element of the dictionary entry separately. Thereafter the user will be asked if they are sure that they want to proceed with the insertion, if “yes” the entry is added, if “no” the function is exited and control handed back to the main program module.

```
def add():

    field_a = ask user for input and read value
    field_b = ask user for input and read value
    field_n = ask user for input and read value

    while True:
        check if the user is sure they want to proceed with insertion
        if(yes):
           add values to dictionary
            break
        elif(no):
            break
        else:
            print("Please enter a correct option")
```

Figure 5 - Insertion Pseudocode

The search function (Figure 6) will use a sequential/linear search, as it iterates through each key/contact in the dictionary. This will have a time complexity of O(n) (Zindros, ND; Brookshear & Brylow, 2018), which will suffice for the requirements. A substring will be compared to each value produced by subsequent iterations, to see whether it is contained within the contact’s name. If present the whole value/contact’s name will

be appended to a local scope variable list, which will be sorted alphabetically at the end of the search and then displayed to the user.

```
def search(substring):
    global dictionary
    List= []
    for values in dictionary:
        if the substring is in the value:
            append the value to the List
    if the length of the List is zero:
        print("No names were found that match")
        return False
    else:
        sort the List
    for element in List:
            print corresponding dictionary value and keys
```

Figure 6 - Insertion Pseudocode

Deleting an entry from the dictionary will require a substring to search, thereafter all matches will be displayed alphabetically. The full name is then requested from the user, thereafter the user is asked if they are sure that they would like to proceed with the deletion, the response read, and the corresponding action performed.

```
def delete(substring):
    global book

    search dictionary for substring
    if substring not in dictionary:
        return
    else:
        print all values which contain the substring
    while True:
        remove= ask for and read full name to delete
        check = check whether the user is sure thay wanto to delete
        if check is yes:
            try:
                delete the contact
                break
            except:
                print("you have entered an incorrect name")
        elif if check is no:
            break
        else:
            print("Please enter a correct option")
```

Figure 7 - Deletion Pseudocode

A display-all function (figure 8) will display all contact names which are currently saved in the dictionary in an alphabetical name-only format. The function copies all the values in the dictionary to a local scope variable list, which is then sorted and iterated through to display each element separately.

```
def display_all():
    global dictionary
    temporary_List= []

    for every key in the dictionary:
        append the key to the temporary_List
    sort the temporary_List
    for name in temporary_List
        print(name)
```

Figure 8 – Display-All Pseudocode

3.) Test Plan

As a fundamental testing principle each line of code will need to be executed (Brookshear & Brylow, 2018). To ensure that this occurs the goal is to simulate various conditions activating specific lines of code (Brookshear & Brylow, 2018). Every function program unit will be tested in a separate isolated program, feeding it test data through global variables such as the dictionary, or hard-coded parameter inputs (Brookshear & Brylow, 2018). To aid simulations the core global dictionary data structure will be hard-coded with the test details depicted in figure 9. These details are just random strings representing examples of user inputs. Once tested each function can then be inserted into the main program and the program run as a whole.

```
Dictionary = {
    'michael botha':['Phone1: 079 777 8888','Phone2:','Email: Mbotha@gmail.com'],
    'Amy Botha':['Phone1: 0728874545','Phone2: 079 784 3333', 'Email: Abotha.com'],
    'Louise wath':['Phone1: 0821234567','Phone2:', 'Email: LW@haha.co.za'],
    'Pin Code:)':['Phone1: 987654','Phone2:', 'Email: Johan@Thunder.com'],
    'Emergency Services':['Phone1: 911','Phone2: 0611115554','Email: fun@him.co.za']}
```

Figure 9 - Test Data Structure

Debugging using the previously specified IDE may be necessary to view the last value of variables, should certain portions of code not work as intended or raise errors, and the problem not easily locatable through simply reviewing the code (Microsoft, 2021c). It is essential to test for program response to known incorrect value insertion, as well as exception handling should a predetermined or unforeseeable error arise. Any visible program lag will certainly be noted, and investigated.

The expected test results for initial testing on the different program units would surely yield some minor syntactical and semantic errors. Preliminarily some of the more complicated portions of code may produce a multitude of incorrect output values as well as built-in Python exceptions. Errors indicated via the interpreter program may include incorrect type, loops which have overrun, and incorrectly indexed list or dictionary (Python Software Foundation, 2021g). Essentially once all errors have been cleared the final code should easily execute and respond to user requests and inputs within a 1 second response time.

4.) References

Bender, M., Farach-Colton, M., Mosteiro, M. (2006) Insertion Sort is O(n log n). *Theory of Computer Systems* 39**:** 391–397.

Brookshear, J., Brylow, D (2018) *Computer Science: An Overview*. 13th ed. London: Pearson.

Edureka.(2020) Binary Search in C. Available from: https://www.edureka.co/blog/binary-search-in-c/ [Accessed 26 June 2021]

Erickson, J. (2019) *Algorithms*.1st ed. Illinois: Jeff Erickson.

Microsoft. (2021a) Visual Studio. Available from: https://visualstudio.microsoft.com [Accessed 25 June 2021].

Microsoft. (2021b) Common File name Extensions in Windows. Available from: https://support.microsoft.com/en-us/windows/common-file-name-extensions-in-windows-da4a4430-8e76-89c5-59f7-1cdbbc75cb01 [Accessed 25 June 2021].

Microsoft. (2021c) What is Debugging. Available from: https://docs.microsoft.com/en-us/visualstudio/debugger/what-is-debugging?view=vs-2019 [Accessed 28 June 2021].

Python Software Foundation. (2021a) IDLE. Available from: https://docs.python.org/3/library/idle.html [Accessed 25 June 2021].

Python Software Foundation. (2021b) Python. Available from: https://www.python.org/downloads [Accessed 25 June 2021].

Python Software Foundation. (2021c) Python Software Foundation. Available from: https://www.python.org/psf-landing [Accessed 25 June 2021].

Python Software Foundation. (2021d) Data Model: Objects, Values, and Types. Available from: https://www.python.org/psf-landing [Accessed 25 June 2021].

Python Software Foundation. (2021e) Data Structures. Available from: https://docs.python.org/3/tutorial/datastructures.html#dictionaries [Accessed 25 June 2021].

Python Software Foundation. (2021f) Lexical Analysis. Available from: https://docs.python.org/3/reference/lexical_analysis.html#identifiers [Accessed 27 June 2021].

Python Software Foundation. (2021g) Built-in Exceptions. Available from: https://docs.python.org/3/library/exceptions.html [Accessed 28 June 2021].

Swaroop, C. (2020) *A Byte of Python*. 10th ed. https://python.swaroopch.com/about.html: Swaroop,C

Thaker, N., Shukla, A. (2020). Python as Multi Paradigm Programming Language. *International Journal of Computer Applications*. 177(31): 38-42. DOI: 10.5120/ijca2020919775.

University of Essex Online (2021a) *Unit 7: Data Abstraction* [webpage]. LCS_PCOM7E MAY 2021. University of Essex Online.

University of Essex Online (2021b) *Learning Python: Strings* [Codio lessons]. LCS_PCOM7E MAY 2021. University of Essex Online.

University of Essex Online (2021c) *Learning Python: Lists* [Codio lessons]. LCS_PCOM7E MAY 2021. University of Essex Online.

University of Essex Online (2021d) *Learning Python: User Defined Functions: Variable Scope* [Codio lessons]. LCS_PCOM7E MAY 2021. University of Essex Online

Zindros, Dionysis (N.D) A Gentle Introduction to Algorithm Complexity Analysis. Available from: https://discrete.gr/complexity/ [Accessed 24 June 2021]

Assignment 1: Part 2

README: Commentary on Implemented Solution

General Implementation Strategy

Encoding the aforementioned design was relatively straightforward except for one or two areas which required more effort, namely the sort and search functions, as their related algorithm complexity and possible number of solutions were larger than the others (Brookshear & Brylow, 2018). Bubble sort and Binary search functions have been added to the source code, and commented out of the execution as a reference and for possible future use. It was considered essential to ensure code was as clean and readable as possible, as well as to follow a principle of minimum complexity enabling ease of maintenance and further code expansion (Brookshear & Brylow, 2018). These goals were aided by Python's semantics which force the use of strict indentations (Python Software Foundation, 2021a), unlike languages similar to C which uses curly braces and semicolons to enclose and demarcate code, but then allow extreme flexibility in terms of code layout, often making it difficult for secondary personnel to read and interpret code if a strict standard is not followed (Griffiths & Griffiths, 2012). Additionally, it was important to provide well-structured and informative comments, with a focus on not interrupting any code flow for a reader with interjecting comments, irregularly dispersed (Brookshear & Brylow, 2018). It was imperative to use descriptive variable names, to enhance readability (Brookshear & Brylow, 2018). An attempt to make instructions to the user as concise and descriptive as possible was made.

Implementation of Data Structures

No module importation was required, as all functions used in the code were called from those built into the Python Interpreter (Python Software Foundation, 2021b). The global compound dictionary was created at the start of the code as it is referenced throughout the program, and hence requires existence early on (University of Essex Online, 2021). Five contacts with their associated information were hard-coded into the data structure during its definition, to provide initial examples for the user and to aid elementary testing. Although Python dictionaries are mutable, and hence global scope variables of that type can be altered from within a function, the “global” keyword is used as a safety mechanism to ensure that no immutability issues can arise (Python Software Foundation, 2021c). Furthermore, the keyword also increases readability as it clearly points to the single global structure. All local scope variables, namely the “temp” list and counting integers, are defined within the respective functions to guarantee no interference with other functions using the same variable names (University of Essex Online, 2021). The function definitions for those required reside between the dictionary creation and the main body of the program. This allows the main program body, or put differently, the portion of the program dictating the flow of the entire execution order of the program, to use the functions as abstract tools (Brookshear & Brylow, 2018). There are five user-defined functions, namely: “sort”, “add”, “search”, “delete”, and “display_all”.

Implementation of Main Body and Functions

The main body of the program uses a while loop to infinitely loop the program unless the user decides to terminate the related process (University of Essex Online, 2021b). Initially options are printed to the screen using the built-in Python print function (Python Software Foundation, 2021d). Thereafter the inputted chosen option is read and stored as a string (Python Software Foundation, 2021d), which is used in unison with an equality operator and a string/character literal to produce the required Boolean result for if or elif statements (University of Essex Online, 2021c). Once the correct if or elif's respective Boolean resolves to true the related option's function is called (University of Essex Online, 2021c). Otherwise, the block of code related to the else statement is run (University of Essex Online, 2021c).

The "sort" function uses an outer for loop to iterate through the list fed as a parameter to the function (University of Essex Online, 2021d; University of Essex Online,2021e), and selects as well as stores each respective temporary element/name. The inner while loop continues comparing the lowercase version of all the names in the list to the temporarily stored string (Python Software Foundation, 2021d), and shifts the elements accordingly. An if statement and its respective code ensure the while loop is broken should the bottom of the list be reached.

Inspecting the "add" function reveals that it asks the user for each required field's related string independently, and stores the associated values (Python Software Foundation, 2021d). Once the values are received and stored the user is prompted for certainty, after which the stored strings are manipulated through concatenation to

produce the required storage format (University of Essex Online, 2021f). The user is then notified of a successful operation and the newly added data printed to the screen.

Observing the “search” function, one will notice that the dictionary keys representing contact names are iterated through using a for loop structure (University of Essex Online, 2021d). A lower-case version of the keyword string which was passed as a parameter is used with the “in” keyword, to determine if it resides in the particular key of an iteration cycle (Python Software Foundation, 2021d). If true, the specific key/contact name is appended to the “temp” list (Python Software Foundation, 2021d). If the temporary list is empty the user is notified that no names matched the search. Otherwise, the list is sorted and the elements as well as related dictionary data is displayed.

Considering the “delete” function it is evident that the abstraction first seeks to match a keyword passed as a parameter via a call to the search “function” (University of Essex Online, 2021g). Once complete the user is prompted to enter the name as displayed after the search, and then asked to confirm their desire to delete a name. Thereafter, either the name is deleted and the function exited, the function exited, or the user prompted to enter a correct name string (University of Essex Online, 2021c).

The “display_all” function simply copies all names in the dictionary to a list, sorts the list, and then iteratively prints each name alphabetically with an accompanying number.

References

Brookshear, J., Brylow, D (2018) *Computer Science: An Overview*. 13th ed. London: Pearson.

Griffiths,D., Giffiths, D. (2012) *Head First C*. California: O'Reilly Media

Python Software Foundation. (2021a) Style Guide for Python Code. Available from: https://www.python.org/dev/peps/pep-0008/ [Accessed 03 July 2021].

Python Software Foundation. (2021b) Modules. Available from: https://docs.python.org/3/tutorial/modules.html [Accessed 03 July 2021].

Python Software Foundation. (2021c) Data Structures. Available from: https://docs.python.org/3/tutorial/datastructures.html#dictionaries [Accessed 03 July 2021].

Python Software Foundation. (2021d) Built-in Functions. Available from: https://docs.python.org/3/library/functions.html [Accessed 03 July 2021].

University of Essex Online (2021a) *Learning Python: User Defined Functions: Variable Scope* [Codio lessons]. LCS_PCOM7E MAY 2021. University of Essex Online

University of Essex Online (2021b) *Learning Python: Loops: While loops* [Codio lessons]. LCS_PCOM7E MAY 2021. University of Essex Online

University of Essex Online (2021c) *Learning Python: Conditionals* [Codio lessons]. LCS_PCOM7E MAY 2021. University of Essex Online

University of Essex Online (2021d) *Learning Python: Loops: For Loops* [Codio lessons]. LCS_PCOM7E MAY 2021. University of Essex Online

University of Essex Online (2021e) *Learning Python: User-Defined Functions: Parameters* [Codio lessons]. LCS_PCOM7E MAY 2021. University of Essex Online

University of Essex Online (2021f) *Learning Python: Strings* [Codio lessons]. LCS_PCOM7E MAY 2021. University of Essex Online

University of Essex Online (2021g) Learning Python: User-Defined Functions: Returning Values [Codio lessons]. LCS_PCOM7E MAY 2021. University of Essex Online

Assignment 1: Part 2

Testing Strategy Implemented

Functionality testing

Initially every program unit, namely the main program body, as well as “sort”, “add”, “search”, “delete”, and “display_all” functions were tested for bugs (Brookshear & Brylow, 2018). Executing each abstraction separately allowed the Python Interpreter to raise exceptions relating to that specific portion of code (Brookshear & Brylow, 2018). Therefore, preliminarily any basic syntactic or semantic errors were raised (Brookshear & Brylow, 2018). Once all statements were in the correct format and structures, the goal of each particular program unit could be checked using hardcoded variables where required.

To further test the main body program unit, the debugger in Microsoft Visual Studio’s platform could be used to simulate the program flow when specific available and unavailable options were fed to the program, using breakpoints which paused the code at desired areas (Microsoft, 2021a). Furthermore, basic print statements were used to test that a block of code executed when it should.

The “sort” function was fed a premade list entered with unsorted names (University of Essex Online, 2021a). This allowed the addition of a temporary iterative printing portion of code to display the output to see whether the elements were sorted into the correct order (University of Essex Online, 2021b).

Testing the “add” function required that each conditional portion of code be checked via entering the various inputs requested, and those not requested, to ensure correct program flow and execution (University of Essex Online, 2021c). This entailed using the test dictionary structure, as the function required the referencing of such in the actual program.

Functionality testing with regards to the “search” function required the actual parameter be hardcoded elsewhere in the program (Brookshear & Brylow, 2018). Once again the test dictionary structure was required for source data. The “sort” function also needed to be added to the test, as the “search” function is to print the sorted list of matched names.

When inspecting the “delete” function, it was fed a hardcoded actual parameter representing the keyword string. The global test dictionary structure was added, as well as the “sort” and “search” functions, as the “search” function is called explicitly and the “sort” indirectly (Brookshear & Brylow, 2018). It was crucial to test the conditional statements, especially the exception handling “try” and “except” blocks which were added to ensure that if an incorrect key be referenced in the dictionary for deletion, that the exception raised had an appropriate recourse (Python Software Foundation, 2021). Therefore, all valid and invalid inputs were injected and the program response investigated.

The “display_all” function required the test dictionary structure and “sort” function as abstracted tools (Brookshear & Brylow, 2018). The output was then observed to ensure correctness according to designed the output.

Performance Analysis

Ensuring that a program does what it is supposed to is important, but efficiency is always a design goal (Brookshear & Brylow, 2018). Furthermore, the user experience is of uttermost importance to ensure customer satisfaction (Brookshear & Brylow, 2018). Additionally, should a program need to function with larger datasets, it needs to be determined how its responses scale (Brookshear & Brylow, 2018). Therefore, basic performance analysis was performed using a larger dataset in mind, as well as the performance profiling tools available in Microsoft Visual Studio (Microsoft, 2021b).

Figure 1 is a table representing a few statistics regarding the program’s related functions, after it executed the five options of adding, searching, deleting, displaying all, and terminating the program, available to the user.

Function Name	Number of Calls	Elapsed Inclusive Time %	Elapsed Exclusive Time %	Avg Elapsed Inclusive Time	Avg Elapsed Exclusive Time	Module Name
input	14	99.98%	99.98%	5,533.01	5,533.01	builtins
print	28	0.02%	0.02%	0.50	0.50	builtins
App_Debugging (module)	1	100.00%	0.00%	77,476.45	0.06	App_Debugging.py
App_Debugging.search	2	0.01%	0.00%	2.48	0.03	App_Debugging.py
App_Debugging.add	1	40.15%	0.00%	31,106.29	0.02	App_Debugging.py
App_Debugging.sort	3	0.00%	0.00%	0.01	0.00	App_Debugging.py
App_Debugging.delete	1	12.67%	0.00%	9,817.00	0.01	App_Debugging.py
App_Debugging.display_all	1	0.00%	0.00%	2.36	0.01	App_Debugging.py
str.lower	40	0.00%	0.00%	0.00	0.00	Unidentifiable Method
list.append	7	0.00%	0.00%	0.00	0.00	Unidentifiable Method
len	5	0.00%	0.00%	0.00	0.00	builtins

Figure 1 - Performance Profile of Program Functions

To produce the above results the program was run with the dictionary data structure containing the five hardcoded contacts previously mentioned. It is noteworthy that the test data resident in the dictionary producing the above results, was not necessarily stored in a format that represented a worst-case scenario for functions like those involving sorting and searching. The number of function calls for both the user-defined and Python built-in functions are displayed (Microsoft, 2021c). Furthermore, various time related details are displayed (Microsoft, 2021c). A figure of note is the Average Elapsed Exclusive Time, which represents the average number of seconds that a particular function took to execute all its code, excluding any function calls within the function (Microsoft, 2021c). From the related data it would seem that the print function is the worst performing, and that under the specific test conditions the “sort” and “search” functions, which can be areas of concern performed relatively well.

Extrapolating the information for both the “sort” and “search” functions and adding an extra ten percent for inefficiencies. Furthermore, using the goal of five hundred contacts and the worst-case O notation time complexity functions for the two program units, the estimated time for the linear “search” function would be 3 seconds as per figure 2’s calculations below.

```
Number_of_contacts = a.t
5 = a.0.03
Therefore a = 5/0.03

Extrapolating for 500:
500 = (5/0.03) x t
t = 3
adding 10% -> 3 + 3x0.2 = 3.6 sec
```

Figure 2 - Extrapolating for 500

Without extrapolation it is difficult to produce scaled results, as acquiring the required dataset in the correct format can be challenging, but essentially the goal specification of one second for response was not met for a larger dataset. Extrapolation for the "sort" function which has a time complexity of $O(n^2)$ is not necessary at this point. Should faster code execution be required a compiled programming language like C++ would ramp-up performance by producing executable files, after which further algorithm analysis could be performed to attain more efficient logics (Brookshear & Brylow, 2018).

References

Brookshear, J., Brylow, D (2018) *Computer Science: An Overview*. 13th ed. London: Pearson.

Microsoft. (2021a) What is Debugging. Available from: https://docs.microsoft.com/en-us/visualstudio/debugger/what-is-debugging?view=vs-2019 [Accessed 02 July 2021].

Microsoft. (2021b) First Look at Profiling Tools. Available from: https://docs.microsoft.com/en-us/visualstudio/profiling/profiling-feature-tour?view=vs-2019 [Accessed 02 July 2021].

Microsoft. (2021c) Functions View - Instrumentation Data. Available from: https://docs.microsoft.com/en-us/visualstudio/profiling/functions-view-instrumentation-data?view=vs-2017 [Accessed 02 July 2021].

Python Software Foundation. (2021) Errors and Exceptions. Available from: https://docs.python.org/3/tutorial/errors.html [Accessed 04 July 2021].

University of Essex Online (2021a) *Learning Python: User Defined Functions: Variable Scope* [Codio lessons]. LCS_PCOM7E MAY 2021. University of Essex Online

University of Essex Online (2021b) *Learning Python: Loops: For Loops* [Codio lessons]. LCS_PCOM7E MAY 2021. University of Essex Online

University of Essex Online (2021c) *Learning Python: Conditionals* [Codio lessons]. LCS_PCOM7E MAY 2021. University of Essex Online