Software Engineering Project

Nicholas Branfield
- 444140 -
444140@wits.ac.za

Higher Diploma in Computer Science
School of Computer Science
University of the Witwatersrand

01 June 2014

Contents

1 Problem Definition

This project has two major (and independent) goals:
1 - To explore and contrast the speeds at which 3 sorting algorithms can preform a sorting task
2 - To demonstrate file input and output (IO) using GDBM and text files

The three algorithms to be tested are:

1. Bubble Sort
2. Heap Sort
3. Quick Sort

The first two algorithms will be coded by our team, while the third (quick sort) will be implemented using a library function. Due to its impressive speed, the team decided to use the quick sort library function offered as part of numpy.

As the file IO and sorting algorithms operate independently we have decided to implement them as separate python packages, namely FileIO and Sorts, respectively

The documentation to follow aims to design, develop and test a solution such that the completed end product is useful, usable and used.

2 Requirement Analysis

Our program will be used by students of computer scientists who have recently started adopting python, and are currently exploring empirical analysis of algorithms. The students in question require a small demo program which will execute just three of the many sorting algorithms, which they can expand and a later time if desired.

For empirical analysis to be successful our program will implement timing functionality so each algorithm can be compared based on the amount of time it takes to execute on two input sizes (n = 100 and n = 10 000). The students will take our demo program and will be tasked to code a graphical overlay, which will allow an easy visualization of each algorithm in action - as part of a summer project.

These students will also tasked to used extensive file IO as part of this project, and so we will include a small demo of how text file and GDBM IO in out project for their introduction to these tasks.

This project is fairly simplistic and is designed to be freely-available to all university staff and students. It must be freely-available as it will form part of an academic project. For this reason this project will not be allocated a budget and will be developed and managed by a group of volunteers, who are all software engineering students. Furthermore the source code as well as the documentation should be available to the users, such that they can extend or tailor the program's capabilities if they so desire.

2.1 Requirement Specification

The specific requirements of this project are:

1. All input and output are preformed to the command line

2. Three sorting algorithms will be implemented:

    2.1. Bubble Sort
    2.2. Heap Sort
    2.3. Quick Sort (library function)

3. the ability to record timings for these algorithms

4. file input and output to a text file

5. file input and output to a database with GDBM

3 Process Choice and Justification

Due to the simplistic nature of this project, and having well defined user requirements upfront, the modified waterfall model has been selected to guide the progression of this project. This model will allow the development team to progress from the requirements (laid out in Section 2) to the system design without needing to develop working prototypes for the users to test. This is advantageous as it focuses the project team to spend more time on one project phase at a time, rather then devoting some time and effort to prototyping and user interaction.

As the project team is made up of volunteers, who may enter and leave the project at any time, the emphasis this model places on documentation will be a great asset. As minimal knowledge will not be lost if any particulate volunteer leaves, similarly new volunteers can quickly familiarize themselves by reading the documentation already laid out.

While some evolution of this project is inevitable, there are no plans at this time for the development team to add more features in subsequent versions. The full desired solution should be available to users, as the final end product of this approach. Users will also have access to the source code and the documentation, this requirement synergies well with the waterfall model - as this model naturally places emphasis on the creation of both documentation and source code.

4 Software Design Strategy

This documentation aims to translate the identified requirements into a model which can be easily implemented in the subsequent stage.

4.1 Main Activities

The user will interact with either Sorts or FileIO package through the command line. As these programs have been designed for demonstration purposes they are executed and run without the need for extensive user input or customizability.

4.2 Milestones

For this project the chosen milestones are closely linked to the above activities, however composite activities have been divide into smaller component parts for a more accurate measure of progress. A milestone is considered achieved when the program can execute said step with no runtime or compile time errors occurring.

1. Sorting algorithms
   (a) Generate Array capable of making a new array of desired size using random numbers in certain range
   (b) Permute Array capable of reording elements of array
   (c) Bubble sort is coded in full and can sort a randomly generated array
   (d) Heap sort is coded in full and can sort a randomly generated array
2. FileIO
   (a) Read in initial Text file
   (b) Write this data into a database using GDBM
   (c) Read first 15 and last 15 items from this database
   (d) Write this data to another text file
   (e) Read first 15 and last 15 items from this text file

4.3 Special Requirements

In order to guarantee accuracy of calculations (particularly with real numbers in the sub normal range) software implemented data types, such as the decimal type, will be utilized. This places the need for specialized libraries (which support these data types) or for more recent language builds which contain such classes in their standard libraries. The use of software implemented types will reduce the speed of calculations involving them, however this is a necessary trade-off to meet the user's requirement for accuracy.

4.4 Pseudo-code

Due to page limitations, exhaustive pseudo-code of entire project is not possible and so only the skeletons of the three sorting algorithms have been included below.

4.4.1 Bubble Sort

```
repeat
    hasChanged := false
    decrement itemCount
    repeat with index from 1 to itemCount
        if (item at index) > (item at (index + 1))
            swap (item at index) with (item at (index + 1))
            hasChanged := true
until hasChanged = false
```

4.4.2 Quick Sort

```
function quicksort(array)
    less, equal, greater := three empty arrays
    if length(array) > 1
        pivot := select any element of array
        for each x in array
            if x < pivot then add x to less
            if x = pivot then add x to equal
            if x > pivot then add x to greater
        quicksort(less)
        quicksort(greater)
        array := concatenate(less, equal, greater)
```

4.4.3 Heap Sort

```
function heapify(a,count) is
   (start is assigned the index in a of the last parent node)
   start := (count - 2) / 2

   while start ≥ 0 do
      (sift down the node at index start to the proper place
       such that all nodes below the start index are in heap
       order)
      siftDown(a, start, count-1)
      start := start - 1
   (after sifting down the root all nodes/elements are in heap order)

function siftDown(a, start, end) is
   (end represents the limit of how far down the heap to sift)
   root := start

   while root * 2 + 1 ≤ end do       (While the root has at least one child)
      child := root * 2 + 1           (root*2+1 points to the left child)
      (If the child has a sibling and the child's value is less than its sibling's...)
      if child + 1 ≤ end and a[child] < a[child + 1] then
         child := child + 1           (... then point to the right child instead)
      if a[root] < a[child] then     (out of max-heap order)
         swap(a[root], a[child])
         root := child                (repeat to continue sifting down the child now)
      else
         return
```

4.4.4 Test cases

For the sorting algorithms testing is automatically performed in the testing script, provided by python's native unittest functionality. This testing involves timing the performance of each algorithm as its sorts a list of 100 randomly generated numbers.

In each case the array is permuted so it remains random before each algorithm works on it. Each of these tests outputs the data set before sorting and again after sorting - to allow the student user the ability to verify sorting has occurred.

If the user desires a more concise summary, a table of timings can be produced for these algorithms performance on a larger data set (10 000 random numbers) within the PartII script.

As the FileIO is meant as a demonstration/example program for the student user's learning, further testing was not considered necessary. The details of these testing process, as well as the results of said test cases, are laid out in Section 7

5 Implementation Strategy

As most of the computers to be used by the target user base are Linux machines and most of the users are expected to continue to use python for years to come. Our project was written using python 3.4 grammar. In addition, as mentioned in Section 4.3 the use of the numpy library is required. For these reasons a note has been added to the readme file to raise the user's awareness to these issues, so they can update their version of python if needed.

The implementation of this project has no special hardware requirements. The target computers are assumed to have a working version of Linux and python installed - meeting the minimum hardware requirements of these is sufficient for our program to operate.

The executable program, the source code and the documentation will be available freely to download from the git-hub repository - `https://github.com/DarkLordZul/444140_SEProject/tree/Final`

This repository can be cloned by each user and once downloaded the user merely has to execute say the *IOManager.py* file from the command line terminal using the command:

```
python \IOManager.py
```

Or alternatively the can make use of a *setup.py* provided to copy all the python scrips to the appropriate directory for third-party modules in their Python installation.

As mentioned in Section 3 both the source code and documentation will be available for the user to evolve or tailor to their own needs - they available from the git-hub repository.

6 Software Configuration

As mentioned in Section 3 while some evolution of this project is inevitable, there are no plans at this time for the development team to add more features in subsequent versions. The full desired solution should be available to users, as the final end product of this approach. Users will also have access to the source code and the documentation, from the `https://github.com/DarkLordZul/444140_SEProject/tree/Final` on-line repository.

There is no special configuration required for this software solution, as it has been specifically designed to run directly from a terminal by calling the command laid out in Section 5.

7 Testing Strategy

Result screen shots for testing of Sorts package given below

7.1 Test Results

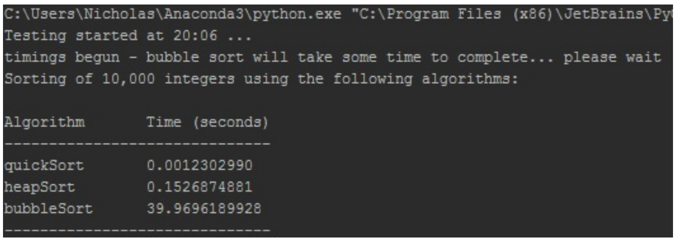

Figure 1: Part II

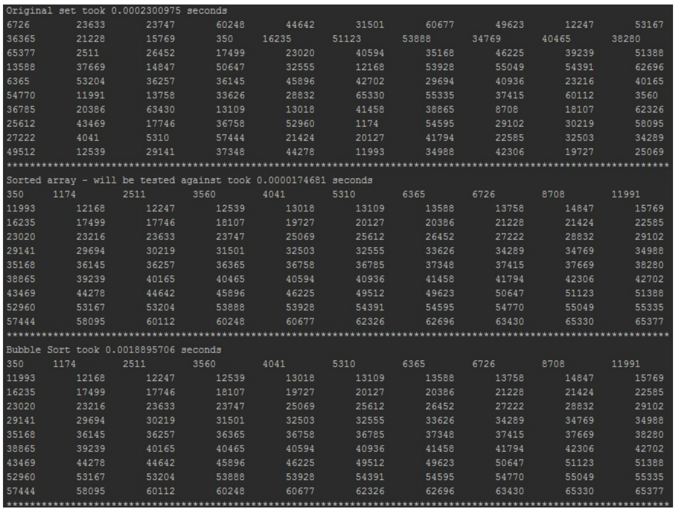

Figure 2: Bubble Sort Output

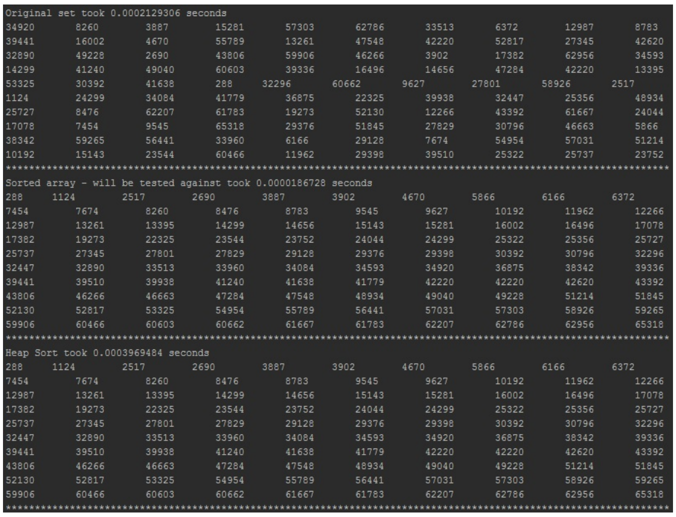

Figure 3: Heap Sort Output

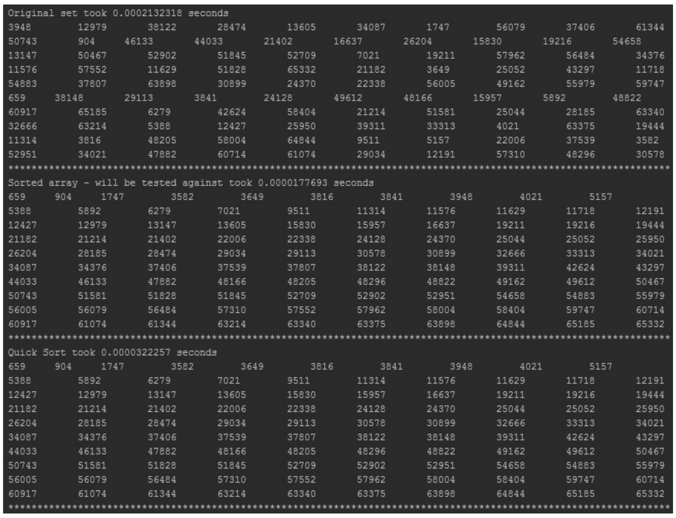

Figure 4: Quick Sort Output

As it is clear from the testing results no problems where encountered with the Sorts package. However in the interests of full disclosure, no elegant way of displaying test results for the FileIO package could be decided on. It must be said that there is an error in FileIO which causes the items in the database to be unsorted, even if the items read in from the program are sorted. The cause of this error is at this time undetermined but as this is mainly for demonstration and/or learning purposes - perhaps the computer science students this project is designed for can come up with a solution.