

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

Contents

1	Problem Definition	1
2	Requirement Analysis	2
2.1	Requirement Specification	3
3	Process Choice and Justification	4
4	Software Design Strategy	5
4.1	Main Activities	5
4.2	Milestones	5
4.3	Special Requirements	6
4.4	Pseudo-code	6
4.4.1	Bubble Sort	6
4.4.2	Quick Sort	7
4.4.3	Heap Sort	7
4.4.4	Test cases	8
5	Implementation Strategy	8
6	Software Configuration	9
7	Testing Strategy	10
7.1	Test Results	10

1 Problem Definition

This project has two major (and independent) goals:

- 1 - To explore and contrast the speeds at which 3 sorting algorithms can perform 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 our 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 than 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 it 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

```
C:\Users\Nicholas\Anaconda3\python.exe "C:\Program Files (x86)\JetBrains\Py
Testing started at 20:06 ...
timings begun - bubble sort will take some time to complete... please wait
Sorting of 10,000 integers using the following algorithms:

Algorithm      Time (seconds)
-----
quickSort      0.0012302990
heapSort       0.1526874881
bubbleSort     39.9696189928
-----
```

Figure 1: Part II

```
Original set took 0.0002300975 seconds
6726 23633 23747 60248 44642 31501 60677 49623 12247 53167
36385 21228 15769 350 16235 51123 53888 34769 40465 38280
65397 2511 26452 17499 23020 40594 35168 46225 39239 51388
13588 37669 14847 50647 32555 12168 53928 55049 54391 62696
6365 53204 36257 36145 45896 42702 29694 40936 23216 40165
54770 11991 13758 33626 28832 65330 55335 37415 60112 3560
36785 20386 63430 13109 13018 41458 38865 8708 18107 62326
25612 43469 17746 36758 52960 1174 54595 29102 30219 58095
27222 4041 5310 57444 21424 20127 41794 22585 32503 34289
49512 12539 29141 37345 44278 11993 34988 42306 19727 25069
Sorted array - will be tested against took 0.0000174601 seconds
350 1174 2511 3560 4041 5310 6365 6726 8708 11991
11993 12168 12247 12539 13018 13109 13588 13758 14847 15769
16235 17499 17746 18107 19727 20127 20386 21228 21424 22585
23020 23216 23633 23747 25069 25612 26452 27222 28832 29102
29141 29694 30219 31501 32503 32555 33626 34289 34769 34988
35168 36145 36257 36365 36758 36785 37348 37415 37669 38280
38865 39239 40165 40465 40594 40936 41458 41794 42306 42702
43469 44278 44642 45896 46225 49512 49623 50647 51123 51388
52960 53167 53204 53888 53928 54391 54595 54770 55049 55335
57444 58095 60112 60248 60677 62326 62696 63430 65330 65377
Bubble Sort took 0.0018895706 seconds
350 1174 2511 3560 4041 5310 6365 6726 8708 11991
11993 12168 12247 12539 13018 13109 13588 13758 14847 15769
16235 17499 17746 18107 19727 20127 20386 21228 21424 22585
23020 23216 23633 23747 25069 25612 26452 27222 28832 29102
29141 29694 30219 31501 32503 32555 33626 34289 34769 34988
35168 36145 36257 36365 36758 36785 37348 37415 37669 38280
38865 39239 40165 40465 40594 40936 41458 41794 42306 42702
43469 44278 44642 45896 46225 49512 49623 50647 51123 51388
52960 53167 53204 53888 53928 54391 54595 54770 55049 55335
57444 58095 60112 60248 60677 62326 62696 63430 65330 65377
```

Figure 2: Bubble Sort Output

Original set took 0.0002129306 seconds									
34920	8260	3887	15281	57303	62786	33513	6372	12987	8783
39441	16002	4670	55789	13261	47548	42220	52817	27345	42620
32890	49228	2690	43806	59906	46266	3902	17382	62956	34593
14289	41240	49040	60603	39336	16496	14656	47284	42220	13395
63325	30392	41638	288	32296	60662	9627	27801	58926	2517
1124	24299	34084	41779	36875	22325	39938	32447	25356	48934
25727	8476	62207	61783	19273	52130	12266	43392	61667	24044
17078	7454	9545	65318	29376	51845	27829	30796	46663	5866
38342	59265	56441	33960	6166	29128	7674	54954	57031	51214
10192	15143	23544	60466	11962	29398	39510	25322	25737	23752
Sorted array - will be tested against took 0.0000186728 seconds									
288	1124	2517	2690	3887	3902	4670	5866	6166	6372
7454	7674	8260	8476	8783	9545	9627	10192	11962	12266
12987	13261	13395	14299	14656	15143	15281	16002	16496	17078
17382	19273	22325	23544	23752	24044	24299	25322	25356	25727
25737	27345	27801	27829	29128	29376	29398	30392	30796	32296
32447	32890	33513	33960	34084	34593	34920	36875	38342	39336
39441	39510	39938	41240	41638	41779	42220	42220	42620	43392
43806	46266	46663	47284	47548	48934	49040	49228	51214	51845
52130	52817	53325	54954	55789	56441	57031	57303	58926	59265
59906	60466	60603	60662	61667	61783	62207	62786	62956	65318
Heap Sort took 0.0003969484 seconds									
288	1124	2517	2690	3887	3902	4670	5866	6166	6372
7454	7674	8260	8476	8783	9545	9627	10192	11962	12266
12987	13261	13395	14299	14656	15143	15281	16002	16496	17078
17382	19273	22325	23544	23752	24044	24299	25322	25356	25727
25737	27345	27801	27829	29128	29376	29398	30392	30796	32296
32447	32890	33513	33960	34084	34593	34920	36875	38342	39336
39441	39510	39938	41240	41638	41779	42220	42220	42620	43392
43806	46266	46663	47284	47548	48934	49040	49228	51214	51845
52130	52817	53325	54954	55789	56441	57031	57303	58926	59265
59906	60466	60603	60662	61667	61783	62207	62786	62956	65318

Figure 3: Heap Sort Output

Original set took 0.0002132318 seconds									
3948	12979	38122	28474	13605	34087	1747	56079	37406	61344
50743	904	46133	44033	21402	16637	26204	15830	19216	54658
13147	50467	52902	51845	52709	7021	19211	57962	56484	34376
11576	57552	11629	51828	65332	21182	3649	25052	43297	11718
54883	37807	63898	30899	24370	22338	56005	49162	55979	59747
659	38148	29113	3841	24128	49612	48166	15957	5892	48822
60917	65185	6279	42624	58404	21214	51581	25044	28185	63340
32666	63214	5388	12427	25950	39311	33313	4021	63375	19444
11314	3816	48205	58004	64844	9511	5157	22006	37539	3582
52951	34021	47882	60714	61074	29034	12191	57310	48296	30578
Sorted array - will be tested against took 0.0000177689 seconds									
659	904	1747	3582	3649	3816	3841	3948	4021	5157
5388	5892	6279	7021	9511	11314	11576	11629	11718	12191
12427	12979	13147	13605	15830	15957	16637	19211	19216	19444
21182	21214	21402	22006	22338	24128	24370	25044	25052	25950
26204	28185	28474	29034	29113	30578	30899	32666	33313	34021
34087	34376	37406	37539	37807	38122	38148	39311	42624	43297
44033	46133	47882	48166	48205	48296	48822	49162	49612	50467
50743	51581	51828	51845	52709	52902	52951	54658	54883	55979
56005	56079	56484	57310	57552	57962	58004	58404	59747	60714
60917	61074	61344	63214	63340	63375	63898	64844	65185	65332
Quick Sort took 0.0000322257 seconds									
659	904	1747	3582	3649	3816	3841	3948	4021	5157
5388	5892	6279	7021	9511	11314	11576	11629	11718	12191
12427	12979	13147	13605	15830	15957	16637	19211	19216	19444
21182	21214	21402	22006	22338	24128	24370	25044	25052	25950
26204	28185	28474	29034	29113	30578	30899	32666	33313	34021
34087	34376	37406	37539	37807	38122	38148	39311	42624	43297
44033	46133	47882	48166	48205	48296	48822	49162	49612	50467
50743	51581	51828	51845	52709	52902	52951	54658	54883	55979
56005	56079	56484	57310	57552	57962	58004	58404	59747	60714
60917	61074	61344	63214	63340	63375	63898	64844	65185	65332

Figure 4: Quick Sort Output

As it is clear from the testing results no problems were 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.