### 1 Introduction

In this experiment, you’re expected to analyze sorting methods. You’ll need to research, visualize, benchmark, and compare given two different algorithms.

### 2 Background Information

#### 2.1 Sorting Algorithms

Sorting algorithms make a totally ordered out of a list of comparable elements. Ordered lists can be used as input to other algorithms (searching, merging, comparison, etc) or as a better way to present data to the user. They vary in underlying data structure, implementation complexity, space and time complexity, and valid input domains. For example most of the algorithms run in either $O(N^2)$ or $O(n \log n)$ time. However some $O(N^2)$ algorithm may be much easier to implement than a faster one. On the other hand, there may be a special algorithm that runs on integers only, while it cannot be (efficiently) applied to other domains (like strings).

#### 2.2 Online Resources

You’ll need to utilize online algorithm resources to achieve the goals set in this experiment. In addition to the search engines you regularly use, you are also encouraged to investigate the following sites.


---


2 A general sorting algorithm requires at least $O(n!)$ = $O(n \log n)$ comparisons.
3 Problem

This experiment consists of several simple subtasks. Each task depends on the previous ones, and each of them will be evaluated separately.

3.1 Algorithm Implementations

Each one of you will be assigned two sorting algorithms. You may find the implementation of two sorting algorithms on World Wide Web or from a textbook. However you need to make the following modifications and document them as described in the next section:

- Rewrite one the algorithms in Turkish for your report (i.e.: use Anlat 2005 algorithm language and convert all the identifier names to meaningful Turkish counterparts)
- Add counters for “comparison” and “exchange”. You’ll need these counters while preparing the testing software
- You must fix any coding standard errors (naming, indentation, etc)

3.2 Algorithm Animation

To describe the algorithm in your report, you’ll also need animations of the behaviors of algorithms. The animations are of the snapshots of the intermediate data which will help understanding the data movements. An example can be seen in Figure 2. For this purpose you must:

- Find applets or applications for animation of your sorting algorithms (from a different source than mentioned in the figure)
- Modify them so that they work on range [1, 30] (normally they use a larger range)
- Modify them so that you can take image snapshots and learn number of exchanges at the beginning, at the end, and after each 25th comparison (or at least 8 snapshots if there are less than 150 comparisons)
- Take defined snapshots and comparison and exchange counters and include them in your report

3.3 Your Testing Software

As the third task you need to design and implement software for testing sorting algorithms. The software must provide at least these facilities:

- Ability to select one of the sorting algorithms
- Ability to load a data set of lists and sort them with the selected algorithm
- Ability to export performance metrics. The performance metrics are:
  1. Running time in milliseconds
  2. Number of comparisons
  3. Number of exchanges

3.4 Testing with Predefined Data Sets

You will use your software to test the algorithms with a number of given data sets. Each data set consists of 10 lists and each list will contain at most 100,000 items.

---

3 Algorithm assignments will be announced on experiment webpage
4 These data will too be available on the experiment webpage
5 An \( O(N^2) \) algorithm is expected to finish in less than one minute on a 2Ghz machine
The sets contain lists with different characteristics. The lists will contain increasing number of items. You need to identify the characteristic of each data set and the effect on the performance of the algorithms.

3.4 Performance Graph with Excel or GNUPLOT

Finally, you must build 3 charts (running time, comparisons and exchanges) for each data set given and include them in your report. You will need to export the data in CSV or tab delimited format and process them with Excel or GNUPLOT.

There will be links to guides for using these programs on the experiment webpage.

![Running Time Graph](image)

**Figure 1:** Sample algorithm comparison chart. Please note that vertical scale is logarithmic, data size is larger than defined maximum and only 6 lists are included.

![Visualization Applet](image)

**Figure 2:** Several phases of visualization of behavior of the radix sort algorithm. Applet available at [http://www.cse.msu.edu/~alheepau/SortVisualizer/sorting.html](http://www.cse.msu.edu/~alheepau/SortVisualizer/sorting.html)
4 Specifications

4.1 Input Format
You will download data sets as plain text files from the experiment webpage\(^6\). They will be in UNIX line ending format and each will contain 10 different lists with increasing number of items.

Each line in the dataset will describe a different list. List items will be integers in the range \([0 - 1,000,000]\) and be separated by a single space character.

A sample data set that contains three lists:

```
1 2 3 4 5
40 90 20 20 13 32 11
10 23 22 33 10 42 91 20 43 11 04
```

4.2 CSV and TAB Delimited File Formats
Your testing software must export all the statistics for a data set and an algorithm in a single text file. The format (which is a user selectable option) can be either comma separated or tab delimited.

General structure of the data exported will be like the following table:

<table>
<thead>
<tr>
<th>Input Size</th>
<th>Compares</th>
<th>Exchanges</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.01</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>1</td>
<td>0.01</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>5</td>
<td>0.01</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>10</td>
<td>0.03</td>
</tr>
<tr>
<td>5</td>
<td>25</td>
<td>20</td>
<td>0.05</td>
</tr>
</tbody>
</table>

A comma separated representation of the data above is like:

```
Input Size;Compares;Exchanges;Running Time
1;1;0;0.01
2;4;1;0.01
3;9;5;0.01
4;16;10;0.03
5;25;20;0.05
```

A tab delimited representation of the same data is like: (notice that the column headers are missing and decimal separator is different):

```
1 1 0 0.01
2 4 1 0.01
3 9 5 0.01
4 16 10 0.03
5 25 20 0.05
```

4.3 Source Code Style
You testing software must confirm to the established coding standards of the language you’re using. A coding standard document for each possible language will be available on the experiment webpage. The evaluation of your code will be in these areas:

- Indentation and spacing (especially consistency)

4.4 User Interface Expectations

You need to prepare a user interface for your testing software, since its runtime behavior will be evaluated.

You can make a command line or a graphical user interface. If it’s graphical, all functions (algorithm selection, data set loading, running and results export) must be accessible from the GUI (See Figure 3).

For command line applications you must make use of the following arguments:

```bash
$ ./SortTestSoftware <algorithm name> <dataset file> CSV|TAB
Example:
$ ./SortTestSoftware BubbleSort dataset1.txt TAB
```

Algorithms names are listed in the appendix.

In either case your software must be able to export a tab delimited or comma separated file.

4.4 Valid Platforms

There are several valid platforms to build your application. Please note that you should test your work on dev.cs.hacettepe.edu.tr if you want to use a Linux based platform.

<table>
<thead>
<tr>
<th>Language</th>
<th>Platform Version</th>
<th>Operating System</th>
<th>GUI libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>C#</td>
<td>.Net 1.1</td>
<td>Windows</td>
<td>Windows.Forms or Gtk-Sharp</td>
</tr>
<tr>
<td>C#</td>
<td>Mono 1.0.6</td>
<td>Linux</td>
<td>Windows.Forms or Gtk-Sharp</td>
</tr>
<tr>
<td>Java</td>
<td>Sun J2SE 5</td>
<td>Linux</td>
<td>Swing or AWT</td>
</tr>
<tr>
<td>Python</td>
<td>ActivePython 2.4.0</td>
<td>Windows</td>
<td>wxPython</td>
</tr>
</tbody>
</table>

There will be a detailed list and information to obtain and install these platforms on the experiment webpage.

---

7 Please note that code comments are usually signs of badly designed or incomplete code. A properly written code does not need “code comments”. However “documentation comments” are different story. Please see [http://c2.com/cgi/wiki?ToNeedComments](http://c2.com/cgi/wiki?ToNeedComments) for more discussion on this topic.
Figure 3: An example user interface
5 Evaluation

5.1 Required Files
This experiment requires a ZIP file in the following structure for evaluation. An invalid
structured archive will cause you partial or full score loss. Example archives for each
programming language will be available on the experiment webpage.

The archive requires following directories with case sensitive names:

<table>
<thead>
<tr>
<th>Directory</th>
<th>Files</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>report</td>
<td>report.*</td>
<td>Your report (.doc, .html, .pdf or .sxw)</td>
</tr>
<tr>
<td>algorithms</td>
<td>AlgorithmName.c</td>
<td>Original algorithm sources and rewritten Anlat2005 ones (names are defined in the appendix)</td>
</tr>
<tr>
<td></td>
<td>AlgorithmName.txt</td>
<td></td>
</tr>
<tr>
<td>animation</td>
<td>Java applet source</td>
<td>Modified java source and resulting binary applets.</td>
</tr>
<tr>
<td></td>
<td>Makefile</td>
<td>This directory does not need any file naming standards</td>
</tr>
<tr>
<td>source</td>
<td>AlgorithmName.c</td>
<td>Modified algorithm, your testing software sources and a make file. Algorithms must be stored in separate source files.</td>
</tr>
<tr>
<td></td>
<td>*.c</td>
<td>Makefile or Solution File</td>
</tr>
<tr>
<td>rawdata</td>
<td>*.csv or *.txt</td>
<td>Your experiment results in CSV or TAB delimited text format</td>
</tr>
</tbody>
</table>

5.2 Acceptance Procedure
Your works will be accepted in person from April 1st to April 7th, 04:30 PM, sharp.
Any late arrival will not be accepted.

Works will be first checked by computer software, and you will need to sign a form for
authenticity. Any archive with significant errors (not a ZIP file, Microsoft Word or other
invalid format report, missing directories, etc) will not be accepted. This is not a valid
excuse for late arrival (i.e.: if your work is rejected at 04:15 PM April 07th, you will only
have 15 minutes to fix your errors).

5.3 Compilation of the Your Program
Your testing software and the applet you modified will be compiled before evaluation. The
compilation will be done via the make file or the solution file you supplied.

5.4 Reports
The report must adhere to the Hacettepe University Computer Science Department
Report Writing Guidelines. In addition to that the following items must be available on
your report:

- In data structures and algorithms:
  - Description of the sorting algorithms, in your words
  - The algorithms in Anlat 2005 language
  - Behavior description (using snapshots and counters from the applets)
  - Performance charts and comparison of the algorithms

- In solution:

---

Also, you do not need to supply a make file for python projects since they are not compiled
o The sources you found the algorithm implementations and the applets
o The modification you had to perform (a visual difference will be better)

A hardcopy of the report is also required.

5.5 Testing and Grading
Your submissions will be tested and evaluated according to several criteria which are defined below. A fully working, error resilient, elegant code, alongside with a good report will receive 100 points.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm in Anlat 2005</td>
<td>5</td>
</tr>
<tr>
<td>Testing Software</td>
<td>35</td>
</tr>
<tr>
<td>Runtime</td>
<td>20</td>
</tr>
<tr>
<td>Interface</td>
<td>5</td>
</tr>
<tr>
<td>Code Style</td>
<td>10</td>
</tr>
<tr>
<td>Report</td>
<td>60</td>
</tr>
<tr>
<td>Problem definition</td>
<td>5</td>
</tr>
<tr>
<td>Solution Approach</td>
<td>5</td>
</tr>
<tr>
<td>Spelling and grammar</td>
<td>5</td>
</tr>
<tr>
<td>Aesthetic appeal</td>
<td>5</td>
</tr>
<tr>
<td>Algorithm animations</td>
<td>10</td>
</tr>
<tr>
<td>Algorithm description</td>
<td>5</td>
</tr>
<tr>
<td>Performance charts, analysis and comparison</td>
<td>15</td>
</tr>
<tr>
<td>Other sections</td>
<td>10</td>
</tr>
</tbody>
</table>

Penalties

<table>
<thead>
<tr>
<th>Category</th>
<th>Penalty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attendance</td>
<td>-10 points for each missed class</td>
</tr>
<tr>
<td>Compilation warnings</td>
<td>-1 point for each warning</td>
</tr>
<tr>
<td>Broken make file</td>
<td>-5 points</td>
</tr>
<tr>
<td>Trivial Fixes</td>
<td>-5 points each</td>
</tr>
<tr>
<td>Invalid archive format</td>
<td>-3 points for each mistake</td>
</tr>
</tbody>
</table>

Modified rules apply for documentation style, so that:

\[
report_{final} = report_{initial} \times \left(0.5 + 0.5 \times \frac{software}{35}\right)
\]

Parts of your work will be evaluated as a black box, without any alterations to your source code. Only exceptions are the trivial compilation fixes. However they need to match these criteria:

- They must be trivial, i.e. they must be a single step fix
- They must be spotted by the evaluator on his own
- They must not fix runtime behavior, but only compilation errors
5.6 Group Work
Group work is strongly discouraged. Any detected group work will be considered as cheating.

Notes
SAVE all your work until the experiment is graded.
The assignment must be original, INDIVIDUAL work. Downloaded or modified source codes will be considered as cheating. Also the students who share their works will be punished in the same way.
You can ask your question via course’s news group. (Discussion of the solution on the newsgroups will be considered “group work”, which means “cheating”).
Respect the office hours of your advisor.

References
Experiment Web Page

BIL236 News Groups
Courses.BIL236 on news.cs.hacettepe.edu.tr

Wikipedia
http://en.wikipedia.org

Appendix I: Sorting Algorithms
Below are the sorting algorithms for each section of the class and the names of them to be used in the software

<table>
<thead>
<tr>
<th>Course Section</th>
<th>Sorting Algorithm</th>
<th>Name in the Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Binary Tree Sort$^9$</td>
<td>TreeSort</td>
</tr>
<tr>
<td></td>
<td>Radix Sort</td>
<td>RadixSort</td>
</tr>
<tr>
<td>21</td>
<td>Heap Sort</td>
<td>HeapSort</td>
</tr>
<tr>
<td></td>
<td>Quick Sort</td>
<td>QuickSort</td>
</tr>
<tr>
<td>22</td>
<td>Merge Sort</td>
<td>MergeSort</td>
</tr>
<tr>
<td></td>
<td>Shell</td>
<td>ShellSort</td>
</tr>
</tbody>
</table>

Example Usage (For Binary Tree Sort and Radix Sort):

Invocation of your testing software
$ ./SoftTestSoftware TreeSort input1.txt CSV

Names of the program modules
$ ls source
TreeSort.cpp RadixSort.cpp Main.cpp Makefile

$^9$ Binary tree sort does not try to balance the trees like heap sort