National Taiwan University
Department of Electrical Engineering
Algorithms, Fall 2018

Handout #2
September 13, 2018
Yao-Wen Chang

Programming Assignment #1: Indexing and Sorting
(due 6pm, October 14, 2018 on-line)

ATTENTION: *Plagiarism is very uncivilized behavior and MUST be avoided at all cost. Every submission will be tested by a plagiarism catcher, running on all submissions from this class as well as those from earlier classes. If plagiarism is discovered, all students involved will receive only partial credits or even zero points. Specifically, if the score received is x and there are n current students involved, then the score for each student is x/n; if plagiarism is related to any version from an earlier class, the score for the plagiarized part will be zero.*

Submission URL & ***Online Resources*****:** https://cool.ntu.edu.tw/courses/155/assignments/459

Problem

You are asked to apply **Insertion Sort**, **Merge Sort**, **Heap Sort**, and **Quicksort** to develop **FOUR** word processors with indexing and sorting (common hidden operations for web search) in the C or C++ programming language. Given is an article with words and numbers.

Input

The input consists of a sequence of words, numbers, and symbols, separated by blanks, punctuation marks, and/or line breakers (end of line). Please note that the word processors sort **words in the order of their alphabetic order (from 'a' to 'z')**, ignoring others such as numbers and blanks. For this problem, we consider that words are **case sensitive**, and a compound word or an abbreviation should be treated as a single word; for example, "on-chip" or "it's" should be treated as a word.

Output

The output contains a number indicating the total number of words and numbers, followed by a list of words (sorted in alphabetic order), and the number *n* denoting that it is the *n*-th ***word*** in the article (ignoring other symbols, such as the numbers and blanks) as shown in the following table. For example, the number following by IC is 6 since it is the 6th word in the article. Note that you don't have to sort the number of *n* so that we can see if the sorter is stable.

Output File Format

The output files *MUST* be in the following format so that they can be verified automatically. Failing to follow the format would incur zero scores or significant

penalties.

```
[total word count]
[word] [the appearing-order number]
…
```

Here is an input/output example:

Sample Input	Output for the Sample Input
a Pentium 4 chip, a state-of-the-art IC (integrated circuit), contains 42000000 transistors.	10 IC 6 Pentium 2 a 1 a 4 chip 3 circuit 8 contains 9 integrated 7 state-of-the-art 5 transistors 10

(Note that the order of "a 1" and "a 4" might be swapped for unstable sorters.)

Required Files

You need to submit the following materials in a .zip file:

(1) Source codes **in *src/* directory** (Please divide the four algorithms into four different programs. e.g. **insertionSort.cpp**, **mergeSort.cpp**, **heapSort.cpp**, and **quickSort.cpp**),

(2) Executable binaries **named as "xxxSort",**

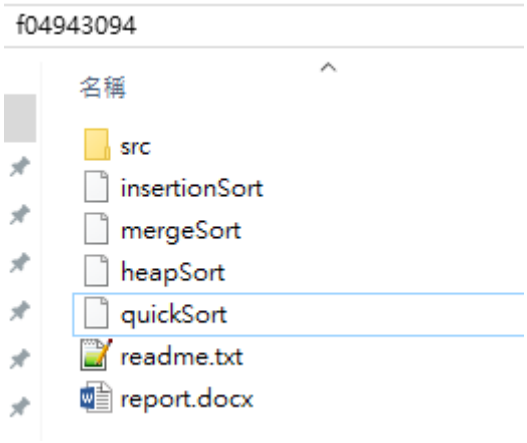
(3) A text readme file **named as readme.txt** describing how to compile and run your programs,

(4) A report **named as report.doc** of the CPU time and memory usage used by the **FOUR** sorters using the following table:

Test Case	Insertion Sort		Merge Sort		Heap Sort		Quicksort	
	Time (Sec)	Memory (MB)	Time (Sec)	Memory (MB)	Time (Sec)	Memory (MB)	Time (Sec)	Memory (MB)
Case1								
…								

(5) Two plots of CPU time vs. problem size (the number of words) and memory vs. problem size for comparing the CPU time and memory usage for the four sorters. (Note that these two plots can be included in the file **report.doc**. ***5-pt Bonus: Use Excel or a tool with the least-square method to find the empirical order/complexity of the running time and space from the two plots.***)

The following image illustrates a submission format example:



Any submission violating the naming and location rules will incur significant penalty.

Language/Platform

Language: C or C++. / Platform: Unix/Linux, Windows, iOS.

File-name Rule

The submission filename should be <student_id>_p1_v[version_number].zip (e.g. **b00901001_p1_v1.zip**). If you have a modified version, please add _v[version_number] as a postfix to the filename and resubmit it to the submission website (e.g. **b00901001_p1_v2.zip**, etc.).

Command-line Parameter

You have to add command-line parameters in your program to specify the input and output file name as the format (e.g., `quickSort input.dat output.dat`):

`[executable_file_name] [input_file_name] [output_file_name]`

Memory Measurement

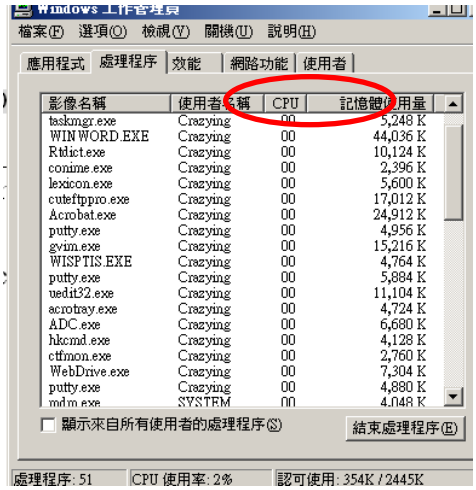
You can add a *getchar()* at the end of sorting to suspend the program. Then use "*top*" for Linux or *taskmgr* for WinXP to measure the memory usage.

Linux: use the *top* instruction.

```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
32599 dovecot   16   0  3116 1508 2892 S  0.0  0.6   0:00.01 imap-login
32592 dovecot   16   0  3116 1508 2892 S  1.0  0.6   0:00.02 imap-login
32588 dovecot   16   0  3116 1508 2892 S  0.0  0.6   0:00.02 imap-login
32399 crazying  25   0 13300 9.8m 2360 S  0.0  3.9   0:06.72 AlgSort
```

The "`VIRT`" field measures the memory usage in the unit of Kbyte.

WinXP: use the Ctrl+Alt+Del or execute *taskmgr* to measure the memory.



Evaluation

The individual score per test case is determined by the (1) accuracy (the correctness of output results, output file format, and **report.doc**), (2) time efficiency, and (3) memory requirement. Bonuses will be given for most efficient programs.

The following gives the information on the evaluation platform:

- 8C16T Intel Xeon CPU @ 2.93GHz, 48GB Ram
- gcc version: 4.8.3

Appendix: The Parser Class, AlgParser

Declaration:

```
class AlgParser
{
    public:
    AlgParser(void);
    // Give the parsing file name and then parse the file
    void Parse( const string& input_file_name );
    // Return the total number of strings in the file
    int QueryTotalStringCount(void);
    // Return the ith string in the file
    string QueryString(const int& ith);
    // Return the line number of the ith string in the file
    int QueryLineNumber(const int& ith);
    // Return the word order of the ith string in the corresponding line
    int QueryWordOrder( const int& ith );
};
```

To use the class, you need to include the “parser.h” and declare a functional object first in your program. Then specify the input file name by passing the parameter to ***AlgParser::Parse()***. The ***Parse()*** function will record the strings as static data. You can retrieve the total number of

strings by ***AlgParser::QueryTotalStringCount()***, the *i*th string by ***AlgParser::QueryString()***, and the line number of *i*th string by ***AlgParser::QueryLineNumber()***.

The Timer Class: AlgTimer

Declaration:

```
class AlgTimer
{
    public:
    AlgTimer(void);
    // Strat the timer
    void Begin(void);
    // Return the accumulated time in seconds
    double End(void);
};
```

First, you still need to include “parser.h” and declare a functional object ***AlgTimer()***. The function ***AlgTimer::Begin()*** starts the timer, and ***AlgTimer::End()*** returns the accumulated time in seconds.

There is an example for using these two provided classes:

```
#include <cstdio>
#include <iostream>
#include "parser.h"
int main( int argc, char** argv )
{
    // Declare the functional objects
    AlgParser p;
    AlgTimer t;

    // Pass the arguement 1 as the input file name
    p.Parse( argv[1] );

    // Start timer
    t.Begin();

    // Display all strings and word numbers
    for( int i = 0 ; i < p.QueryTotalStringCount() ; i++ )
    {
        cout << p.QueryString(i) << " " << p. QueryWordOrder(i)  << endl;
    }
```

```
    // Display the accumulated time
    cout << "The execution spends " << t.End() << " seconds" << endl;

    return 1;
}
```

If there is any question, please send an email to Yu-Sheng (祐昇) at yslu@eda.ee.ntu.edu.tw.