

Programming Assignment #1: Indexing and Sorting **(due 6pm, October 23, 2016 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 credit. 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: <http://eda.ee.ntu.edu.tw/~lzw/alg16/submission/>

Online Resources: <http://eda.ee.ntu.edu.tw/~lzw/alg16/resource/pal.tar.gz>

Problem

You are asked to apply the **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, C++, or Java 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 followed by a list of words (sorted in ASCII-code order), and the number n denoting that it is the n -th **word** in the article (ignoring other symbols, such as numbers and blanks) as shown in the following table. Please do not try to sort the number 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:

[total word count]
[word] [the appearing-order number in the article]
...

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 .tar.gz or .zip file:

- (1) Source codes (e.g. insertion-sort.c; you can divide the four algorithms into four different programs or integrate them into one program with command-line parameters)
- (2) Executable binary
- (3) A text readme file describing how to compile and run your programs
- (4) A report (**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**. **4-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.**)

Language/Platform

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

File-name Rule

The submission filename should be <student_id>-<p1>.tar.gz or <student_id>-<p1>.zip (e.g. **b00901001-p1.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-v1.zip**, **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

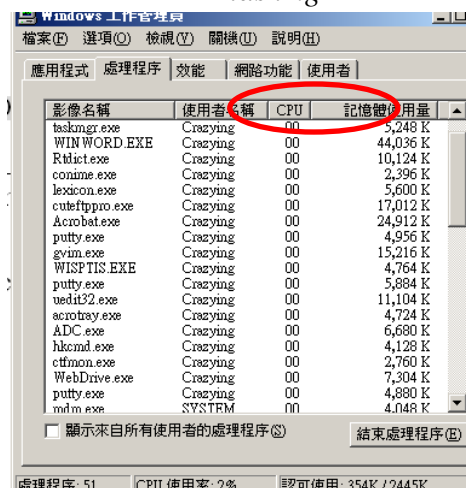
You can add a `getchar()` at the end of sorting to suspend the program. Then use “*top*” for Linux or *taskmgr* for WinXP to measure the memory usage.

Linux: use the *top* instruction.

PID	USER	PR	NI	VI RT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
32599	dovecot	16	0	3116	1508	2892	S	0.0	0.6	0:00.01	imap-logi n
32592	dovecot	16	0	3116	1508	2892	S	1.0	0.6	0:00.02	imap-logi n
32588	dovecot	16	0	3116	1508	2892	S	0.0	0.6	0:00.02	imap-logi n
32399	crazying	25	0	13300	9.8m	2360	S	0.0	3.9	0:06.72	Al gSort

The “VI RT” field measures the memory usage in the unit of Kbyte.

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



映像名稱	使用者名稱	CPU	記憶體用量
taskmgr.exe	Crazying	00	5,248 K
WINWORD.EXE	Crazying	00	44,036 K
Rtldll.exe	Crazying	00	10,124 K
conime.exe	Crazying	00	2,396 K
lexicon.exe	Crazying	00	5,600 K
cuteftp.exe	Crazying	00	17,012 K
Acrobat.exe	Crazying	00	24,912 K
putty.exe	Crazying	00	4,956 K
gvim.exe	Crazying	00	15,216 K
WISPTIS.EXE	Crazying	00	4,764 K
putty.exe	Crazying	00	5,884 K
tedi52.exe	Crazying	00	11,104 K
acrobay.exe	Crazying	00	4,724 K
ADC.exe	Crazying	00	6,680 K
hkcmd.exe	Crazying	00	4,128 K
ctfmon.exe	Crazying	00	2,760 K
WebDrive.exe	Crazying	00	7,504 K
putty.exe	Crazying	00	4,880 K
mlm.exe	SYSTEM	00	4,048 K

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.

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 *ith* string by ***AlgParser::QueryString()***, and the line number of *ith* string by ***AlgParser::QueryLineNumber()***.

The Timer Class: AlgTimer

Declaration:

```
class AlgTimer
{
    public:
        AlgTimer(void);
        // Start 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 argument 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 Zhi-Wen (植文) at lzw@eda.ee.ntu.edu.tw.