Algorithms

Programming Assignment #1

Sorting

Introduction:

In this PA, you are required to implement various sorters that we learnt in the class. You can download the *PA1.tar* file from Ceiba website. Uncompress it using linux command,

```
tar -xvf PA1.tar
```

You can see the following directories after uncompressing it.

Name	Description
`bin/`	Directory of binary file
`doc/`	Directory of document
`inputs/`	Directory of unsorted data
`lib/`	Directory of library source code
`outputs/`	Directory of sorted data
`src/`	Directory of source code
`utility/`	Directory of checker

Input/output Files:

In the input file (`*.in`), the first two lines starting with '#' are just comments. Except comments, each line contains two numbers: index followed by the unsorted number. The range of unsorted number is between 0 and 1,000,000. Two numbers are separated by a space. For an example, the file `5.ac.in` contains five numbers

```
# 5 data points
# index number
0 16
1 13
2 0
3 6
4 7
```

The output file(`*.out`) is actually the same as the input file except that the numbers are sorted in *increasing* order. For example `5.ac.out` is like:

```
# 5 data points
# index number
0 0
1 6
2 7
3 13
4 16
```

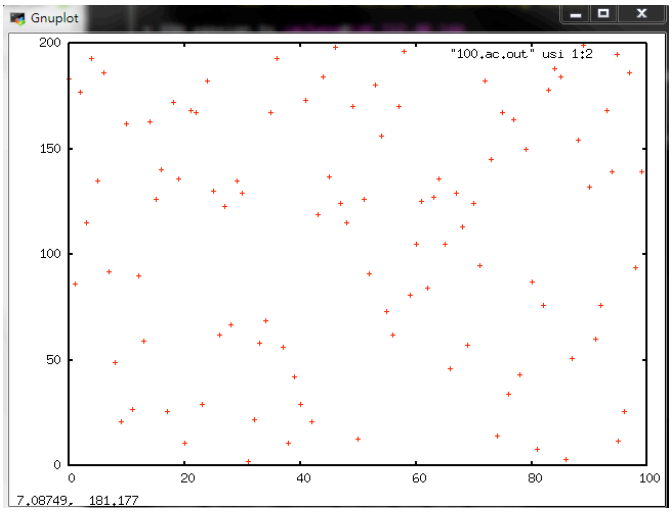
PLOT:

You can visualize your unsorted/sorted numbers by using the gnuplot tool by the command `gnuplot`. After that, please key in the following

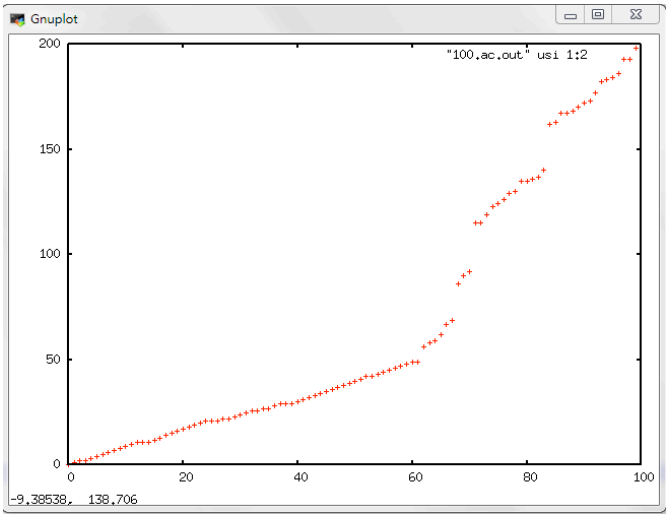
```
set xrange [0:5]
set yrange [0:20]
plot "5.ac.in" usi 1:2
plot "5.ac.out" usi 1:2

# if you want to save to png files
set terminal png
set output "5.ac.out.png"
replot
```

You need to allow X-window display to see the window if you are login remotely. For more gnuplot information, see
http://people.duke.edu/~hpgavin/gnuplot.html

There are two example "before" and "after" sort pictures with 100 numbers benchmark.
Before sort：



After sort：



Command line parameters:

In the command line, you are required to follow this format

```
NTU_sort -[IS|MS|HS] <input_file_name> <output_file_name>
```

where IS represents insertion sort, MS is merge sort, and HS is heap sort. The square bracket with vertical bar '[IS|MS|HS]' means that only one of the three versions is chosen.

The angle bracket <input_file_name> should be replaced by the name of the input file, `*.[ac|bc|wc].in`, where ac represents average case, bc is best case, and wc is worst case. For the best case, all the numbers are sorted in increasing order. For the worst case, all numbers are sorted in descending order. For the average case, numbers are in random order.

The output file names are `*.[ac|bc|wc].out`. Please note that you do NOT need to add '[|]' or '<>' in your command line. For example, the following command sorts *10000.ac.in* to *10000.ac.out* using insertion sort.

```
./bin/NTU_sort -IS  inputs/10000.ac.in  outputs/10000.ac.out
```

Source code files:

Please notice that all of the source code files have been already finished except *sort_tool.cpp*. You only need to complete the different sorting functions of class SortTool in *sort_tool.cpp*. You can still modify other source code files if you think it is necessary. The following will simply introduce the source code files.

main.cpp : main program for PA1

```
1.  // ************************************************************************
2.  //   File           [main.cpp]
3.  //   Author         [Yu-Hao Ho]
4.  //   Synopsis       [The main program of 2015 Spring Algorithm PA1]
5.  //   Modify         [2015/03/02 Yu-Hao Ho]
6.  // ************************************************************************
7.
8.  #include <cstring>
9.  #include <iostream>
10. #include <fstream>
11. #include "../lib/tm_usage.h"
12. #include "sort_tool.h"
13.
14. using namespace std;
15.
16. void help_message() {
17.     cout << "usage: mysort -[IS|MS|HS] <input_file> <output_file>" << endl;
18.     cout << "options:" << endl;
19.     cout << "      IS - Insertion Sort" << endl;
20.     cout << "      MS - Merge Sort" << endl;
21.     cout << "      HS - Heap Sort" << endl;
22. }
23.
24. int main(int argc, char* argv[])
25. {
26.     if(argc != 4) {
27.         help_message();
28.         return 0;
29.     }
30.     CommonNs::TmUsage tmusg;
31.     CommonNs::TmStat stat;
32.
33.     ////////////// read the input file //////////////
34.
35.     char buffer[200];
36.     fstream fin(argv[2]);
37.     fstream fout;
38.     fout.open(argv[3],ios::out);
39.     fin.getline(buffer,200);
40.     fin.getline(buffer,200);
41.     int junk,num;
42.     vector<int> data;
43.     while (fin >> junk >> num)
44.         data.push_back(num); // data[0] will be the first data.
45.                              // data[1] will be the second data and so on.
46.
47.     ////////////// the sorting part ////////////////
48.     tmusg.periodStart();
49.     SortTool NTUSortTool;
50.
51.     if(!strcmp(argv[1],"-IS")) {
52.         NTUSortTool.InsertionSort(data);
53.     }
54.     else if(!strcmp(argv[1],"-MS")) {
55.         NTUSortTool.MergeSort(data);
56.     }
57.     else if(!strcmp(argv[1],"-HS")) {
58.         NTUSortTool.HeapSort(data);
59.     }
60.     else {
61.         help_message();
62.         return 0;
63.     }
64.
65.     tmusg.getPeriodUsage(stat);
66.     cout <<"The total CPU time: " << (stat.uTime + stat.sTime) / 1000.0 << "ms" << endl;
67.     cout <<"memory: " << stat.vmPeak << "KB" << endl; // print peak memory
68.
69.     ////////////// write the output file ///////////
70.     fout << "# " << data.size() << " data points" <<endl;
71.     fout << "# index number" << endl;
72.     for (int i = 0; i < data.size(); i++)
73.         fout << i << " " <<data[i] << endl;
74.     fin.close();
75.     fout.close();
76.     return 0;
77. }
```

main.cpp

Line 34-44: parse unsorted data from input file and push them into the vector.

Line 51-63: call different function depending on given command.

Line 70-73: write the sorted data file.

sort_tool.h : the header file for the SortTool Class

```
1.  // ************************************************************************
2.  //   File           [sort_tool.h]
3.  //   Author         [Yu-Hao Ho]
4.  //   Synopsis       [The header file for the SortTool Class]
5.  //   Modify         [2015/03/02 Yu-Hao Ho]
6.  // ************************************************************************
7.
8.  #ifndef _SORT_TOOL_H
9.  #define _SORT_TOOL_H
10.
11. #include<vector>
12. using namespace std;
13.
14. class SortTool {
15.       public:
16.
17.                              SortTool(); // constructor
18.             void             InsertionSort(vector<int>&); // sort data using insertion sort
19.             void             MergeSort(vector<int>&); // sort data using merge sort
20.             void             HeapSort(vector<int>&); // sort data using heap sort
21.       private:
22.
23.             void             SortSubVector(vector<int>&, int, int); // sort subvector
24.             void             Merge(vector<int>&, int, int, int, int); // merge two sorted subvector
25.             void             Max_Heapify(vector<int>&, int); // make tree with given root be a max-heap
26.                                                                 // if both right and left sub-tree are max-heap
27.             void             Build_Max_Heap(vector<int>&); // make data become a max-heap
28.             int              heapSize; // heap size used in heap sort
29.
30. };
31.
32. #endif
```

sort_tool.h

Line 18-20: sort function which will be called in *main.cpp.*

Line 23: This function will be used in merge sort. It will sort sub vector with given lower and upper bound. This function should be implemented to call itself for splitting and merging the sub vector.

Line 24: This function will be used in merge sort and should be implemented to merge two sorted sub vector.

Line 25: This function will be used in heap sort and should be implemented to make the tree with given root be a max-heap if both of its right subtree and left subtree are max-heap.

Line 27: This function will be used in heap sort and should be implemented to make input data be a max-heap.

sort_tool.cpp : the implementation of the SortTool Class

```
1.  // ************************************************************************
2.  //   File           [sort_tool.cpp]
3.  //   Author         [Yu-Hao Ho]
4.  //   Synopsis       [The implementation of the SortTool Class]
5.  //   Modify         [2015/03/02 Yu-Hao Ho]
6.  // ************************************************************************
7.
8.  #include "sort_tool.h"
9.  #include<iostream>
10.
11. // Constructor
12. SortTool::SortTool() {}
13.
14. // Insertsion sort method
15. void SortTool::InsertionSort(vector<int>& data) {
16.       // Function : Insertion sort
17.       // TODO : Please complete insertion sort code here
18. }
19.
20. // Merge sort method
21. void SortTool::MergeSort(vector<int>& data){
22.       SortSubVector(data, 0, data.size() - 1);
23. }
24.
25. // Sort subvector
26. void SortTool::SortSubVector(vector<int>& data, int low, int high) {
27.       // Function : Sort subvector
28.       // TODO : Please complete SortSubVector code here
29.       // Hint : recursively call itself
30.       //              Merge function is needed
31. }
32.
33. // Merge
34. void SortTool::Merge(vector<int>& data, int low, int middle1, int middle2, int high) {
35.       // Function : Merge two sorted subvector
36.       // TODO : Please complete the function
37. }
38.
39. // Heap sort method
40. void SortTool::HeapSort(vector<int>& data) {
41.       // Build Max-Heap
42.       Build_Max_Heap(data);
43.       // 1. Swap data[0] which is max value and data[i] so that the max value will be in correct location
44.       // 2. Do max-heapify for data[0]
45.       for (int i = data.size() - 1; i >= 1; i--) {
46.             swap(data[0],data[i]);
47.             heapSize--;
48.             Max_Heapify(data,0);
49.       }
50. }
51.
52. //Max heapify
53. void SortTool::Max_Heapify(vector<int>& data, int root) {
54.       // Function : Make tree with given root be a max-heap if both right and left sub-tree are max-heap
55.       // TODO : Please complete max-heapify code here
56. }
57.
58. //Build max heap
59. void SortTool::Build_Max_Heap(vector<int>& data) {
60.       heapSize = data.size(); // initialize heap size
61.       // Function : Make input data become a max-heap
62.       // TODO : Please complete Build_Max_Heap code here
63. }
```

sort_tool.cpp

Line 15-18: please complete the function of insertion sort here.

Line 21-23: the function of merge sort will call function of Sorting sub-vector and give initial lower/upper bound.

Line 26-31: please complete the function of sorting sub-vector here.

Line 34-37: please complete the function of merging two sorted sub-vector here.

Line 40-50: the function of heap sort will build max-heap first. And then, exchange data iteratively.

Line 53-56: please complete the function of max-heapify which makes the tree with given root be a max-heap if its right and left sub-tree are both max-heap.

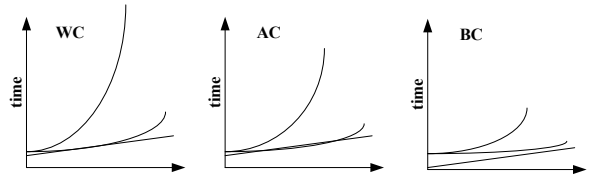
Line 59-63: please complete the function of building max-heap with given input data.

Requirements:

1. Please check the source code files under the src directory. You may need to complete the functions of class SortTool in *sort_tool.cpp*. You can also modify *main.cpp* and *sort_tool.h* if you think it is necessary.
2. Your source code must be written in C or C^{++}. The code must be executable on EDA union lab machines.
3. In your report, compare the running time of three versions of different input sizes. Please fill in the following table. Please use –O2 optimization and turn off all debugging message.

Input size	IS		MS		HS	
	CPU time (s)	Memory (KB)	CPU time (s)	Memory (KB)	CPU time (s)	Memory (KB)
4000.bc						
4000.wc						
4000.ac						
16000.bc						
16000.wc						
16000.ac						
32000.bc						
32000.wc						
32000.ac						
1000000.bc						
1000000.wc						
1000000.ac						

4. Draw figures to show the growth of running time as a function of input size (as the following example, where each curve represents an algorithm.)



You can skip the test case if the run time is more than 10 minutes.

5. Notice: You are not allowed to use “sort” or “priority_queue” in STL!!

Compile

We expected your code can compile and run in this way.

Type the following commands under <student_id>_pa1 directory,

```
make
cd bin
./NTU_sort -[IS|MS|HS] <input_file_name> <output_file_name>
```

We provide the sample makefile, **please modify into yours if needed**.

```
1.  # CC and CFLAGS are varilables
2.  CC = g++
3.  CFLAGS = -c
4.  AR = ar
5.  ARFLAGS = rcv
6.  # -c option ask g++ to compile the source files, but do not link.
7.  # -g option is for debugging version
8.  # -O2 option is for optimized version
9.  DBGFLAGS = -g -D_DEBUG_ON_
10. OPTFLAGS = -O2
11. # make all
12. all: bin/NTU_sort
13.     @echo -n ""
14.
15. # optimized version
16. bin/NTU_sort: sort_tool_opt.o main_opt.o lib
17.     $(CC) $(OPTFLAGS) sort_tool_opt.o main_opt.o -ltm_usage -Llib -o bin/NTU_sort
18. main_opt.o: src/main.cpp lib/tm_usage.h
19.     $(CC) $(CFLAGS) $< -Ilib -o $@
20. sort_tool_opt.o: src/sort_tool.cpp src/sort_tool.h
21.     $(CC) $(CFLAGS) $(OPTFLAGS) $< -o $@
22.
23. # DEBUG Version
24. dbg: bin/NTU_sort_dbg
25.     @echo -n ""
26.
27. bin/NTU_sort_dbg: sort_tool_dbg.o main_dbg.o lib
28.     $(CC) $(DBGFLAGS) sort_tool_dbg.o main_dbg.o -ltm_usage -Llib -o bin/NTU_sort_dbg
29. main_dbg.o: src/main.cpp lib/tm_usage.h
30.     $(CC) $(CFLAGS) $< -Ilib -o $@
31. sort_tool_dbg.o: src/sort_tool.cpp src/sort_tool.h
32.     $(CC) $(CFLAGS) $(DBGFLAGS) $< -o $@
33.
34. lib: lib/libtm_usage.a
35.
36. lib/libtm_usage.a: tm_usage.o
37.     $(AR) $(ARFLAGS) $@ $<
38. tm_usage.o: lib/tm_usage.cpp lib/tm_usage.h
39.     $(CC) $(CFLAGS) $<
40.
41. # clean all the .o and executable files
42. clean:
43.     rm -rf *.o lib/*.a bin/*
```

makefile

Line 38-39: compile the object file *tm_usage.o* from *tm_usage.cpp* and *tm_usage.h*

Line 36-37: archive *tm_usage.o* into a static library file *libtm_usage.a*. Please note that library must start with *lib* and ends with *.a*.

Line 37: this small library has only one objet file. In a big library, more than one objective files can be archived into a single *lib*.a* file like this

```
ar rcv libx.a file1.o [file2.o …]
```

Lines 12-21: When we type '`make`' without any option the makefile will do the first command (line.12 in this sample). Thus, we can compile the optimization version when we type '`make`'. This version invokes options '-O2' for speed improvement. Also '_DEBUG_ON_' is not defined to disable the printing of arrays in *sort_tool.cpp*.

Lines 23-32: Compile the debug version when we type '`make dbg`'. This version invokes options '-g' (for DDD debugger) and also '-D_DEBUG_ON_' to enable the printing of arrays in *sort_tool.cpp*.

Lines 13,25: `@echo -n "″` will print out the message in "″. In this sample we print nothing.

Notice: $< represent the first dependency.

$@ represent the target itself.

Example: a.o : b.cpp b.h

```
$(CC) $(CFLAGS) $(DBGFLAGS) $< -o $@
```

$< = b.cpp $@ = a.o

You can find some useful information here.

http://mrbook.org/tutorials/make/

Validation:

You can verify your answer very easily by comparing your output with .bc (best case) which is the sorted input. Or you can see the gnuplot and see if there is any dot that is not sorted in order.

Also, you can use our result checker which is under utility directory to check whether your result is correct or not. To use this checker, simply type

```
./PA1_Result_Checker <filename>
```

Please notice that it will only check your data order and will not check whether the format of result file is correct or not. You have to check the format by yourself if you modify the part of writing output file in *main.cpp*.

Submission:

Please submit a hardcopy of your report in class. Also, please submit a single **.tgz* file to CEIBA system. Your submission must contain:

1. <student_id>_pa1 directory contains your source code in *src* directory. By simply typing "make" can compile.
2. a report in the *doc* directory. <student_id>_pa1_report.doc.
3. a README file that explains your files.
4. We will use our own test cases so do NOT include the input files.
5. In summary, you should at least have the following item in your *.tgz file.
   src/<all your source code>

lib/<library file>
bin/NTU_sort
doc/<student_id>_pa1_report.doc
makefile
README

The submission filename should be compressed in a single file <student_id>_pa1.tgz. (e.g. b90901000_pa1.tgz). You can use the following command to compress a whole directory:

```
tar -zcvf <filename>.tgz <dir>
```

6. You are required to run the checksubmitPA1 script to check if your .tgz submission file is correct. To use this script, simply type

```
./checkSubmitPA1.sh  b99901000_pa1.tgz
```

We have so many students in the class so we need automatic grading. Any mistake in the submission will result in cost 20% off your score. Please be very careful in your submission.

Grading:

60% correctness (including submission correctness)
20% file format and location
20% report

Bonus :

5% bonus will be given to any extra features that you add to make your sorter faster. Please write clearly in your report.

NOTE:

TA will check your source code carefully. Copying other source code can result in zero grade for all students involved.