National Taiwan University
Department of Electrical Engineering
Algorithms, Fall 2021

September 22, 2021
James Chien-Mo Li

# Algorithms

## Programming Assignment #1

Sorting

## Submission & Online Resources: on NTU COOL

## Introduction:

In this PA, you are required to implement various sorters that we learnt in class. You can download the *PA1.tar* file from NTU COOL 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 example, the file `5.case1.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.case1.out` is like:

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

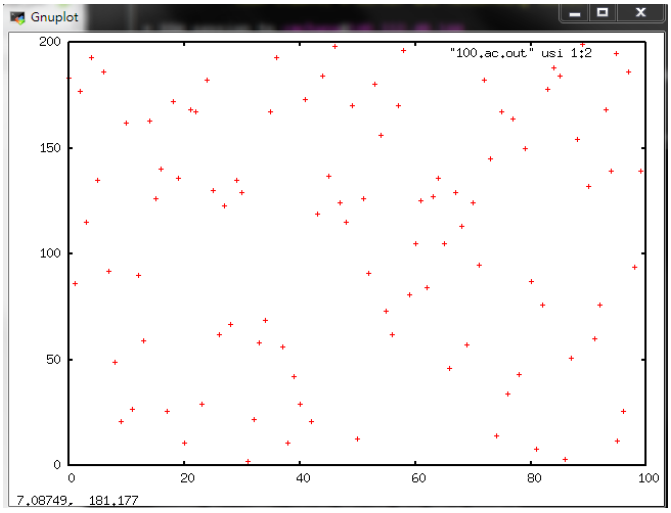
**PLOT:**

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

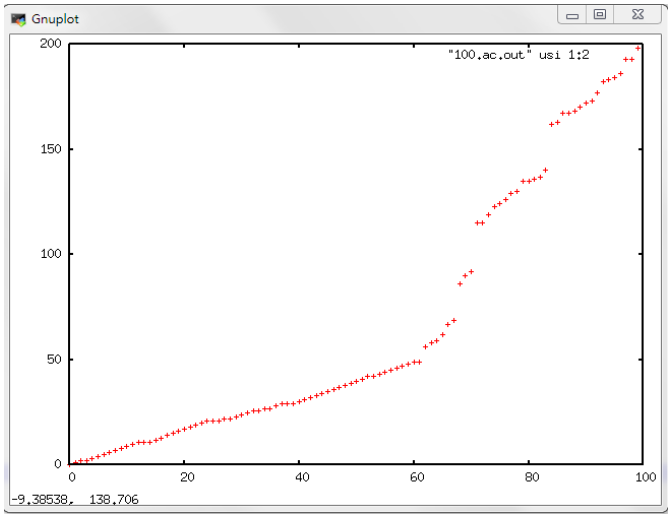
```
set xrange [0:5]
set yrange [0:20]
plot "5.case1.in" usi 1:2
plot "5.case1.out" usi 1:2

# if you want to save to png files
set terminal png
set output "5.case1.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|QS|HS] <input_file_name> <output_file_name>
```

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

The angle bracket <input_file_name> should be replaced by the name of the input file, `*.[case1|case2|case3].in`, where case1 represents test case in random order, case2 is test case in increasing order, and case3 is test case in reverse order. 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 `*.[case1|case2|case3].out`. Please note that you do NOT need to add '[|]' or '<>' in your command line. For example, the following command sorts *1000.case1.in* to *1000.case1.out* using insertion sort.

```
./bin/NTU_sort -IS inputs/1000.case1.in outputs/1000.case1.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 2019 fall Algorithm PA1]
5.  //   Modify         [2020/9/15 Mu-Ting Wu]
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: NTU_sort -[IS|MS|QS|HS] <input_file> <output_file>" << endl;
18.     cout << "options:" << endl;
19.     cout << "      IS - Insersion Sort" << endl;
20.     cout << "      MS - Merge Sort" << endl;
21.     cout << "      QS - Quick Sort" << endl;
22.     cout << "      HS - Heap Sort" << endl;
23. }
24.
25. int main(int argc, char* argv[])
26. {
27.     if(argc != 4) {
28.         help_message();
29.         return 0;
30.     }
31.     CommonNs::TmUsage tmusg;
32.     CommonNs::TmStat stat;
33.
34.     /////////// read the input file //////////
35.
36.     char buffer[200];
37.     fstream fin(argv[2]);
38.     fstream fout;
39.     fout.open(argv[3],ios::out);
40.     fin.getline(buffer,200);
41.     fin.getline(buffer,200);
42.     int junk,num;
43.     vector<int> data;
44.     while (fin >> junk >> num)
45.         data.push_back(num); // data[0] will be the first data.
46.                              // data[1] will be the second data and so on.
47.
48.     /////////// the sorting part ////////////////
49.     tmusg.periodStart();
50.     SortTool NTUSortTool;
51.
52.     if(!strcmp(argv[1],"-QS")) {
53.         NTUSortTool.QuickSort(data);
54.     }
55.     else if(!strcmp(argv[1],"-IS")) {
56.         NTUSortTool.InsertionSort(data);
57.     }
58.     else if(!strcmp(argv[1],"-MS")) {
59.         NTUSortTool.MergeSort(data);
60.     }
61.     else if(!strcmp(argv[1],"-HS")) {
62.         NTUSortTool.HeapSort(data);
63.     }
64.     else {
65.         help_message();
66.         return 0;
67.     }
68.
69.     tmusg.getPeriodUsage(stat);
70.     cout <<"The total CPU time: " << (stat.uTime + stat.sTime) / 1000.0 << "ms" << endl;
71.     cout <<"memory: " << stat.vmPeak << "KB" << endl; // print peak memory
72.
73.     /////////// write the output file //////////
74.     fout << "# " << data.size() << " data points" <<endl;
75.     fout << "# index number" << endl;
76.     for (int i = 0; i < data.size(); i++)
77.         fout << i << " " <<data[i] << endl;
78.     fin.close();
79.     fout.close();
80.     return 0;
81. }
```

**main.cpp**

Line 36-46: parse unsorted data from input file and push them into the vector.

Line 52-67: call different function depending on given command.

Line 74-77: 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        [2020/9/15 Mu-Ting Wu]
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.                         SortTool(); // constructor
17.         void            InsertionSort(vector<int>&); // sort data using insertion sort
18.         void            MergeSort(vector<int>&); // sort data using merge sort
19.         void            QuickSort(vector<int>&); // sort data using quick sort
20.         void            HeapSort(vector<int>&); // sort data using heap sort
21.     private:
22.         void            QuickSortSubVector(vector<int>&, int, int); // quick sort subvector
23.         int             Partition(vector<int>&, int, int); // partition the subvector
24.         void            MergeSortSubVector(vector<int>&, int, int); // merge sort subvector
25.         void            Merge(vector<int>&, int, int, int, int); // merge two sorted subvectors
26.         void            MaxHeapify(vector<int>&, int); // make tree with given root be a max-heap
27.                                                         //if both right and left sub-tree are max-heap
28.         void            BuildMaxHeap(vector<int>&); // make data become a max-heap
29.         int             heapSize; // heap size used in heap sort
30.
31. };
32.
33. #endif
```

**sort_tool.h**

Line 17-20: sort function which will be called in *main.cpp.*

Line 22: This function will be used in quick sort. It will sort sub vector with given lower and upper bound. This function should be implemented to partition the sub vector and recursively call itself.

Line 23: This function will be used in quick sort and should be implemented to partition the sub vector.

Line 24: 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 25: This function will be used in merge sort and should be implemented to merge two sorted sub vectors.

Line 26: 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 28: 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         [2020/9/15 Mu-Ting Wu]
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. // Quick sort method
21. void SortTool::QuickSort(vector<int>& data){
22.     QuickSortSubVector(data, 0, data.size() - 1);
23. }
24. // Sort subvector (Quick sort)
25. void SortTool::QuickSortSubVector(vector<int>& data, int low, int high) {
26.     // Function : Quick sort subvector
27.     // TODO : Please complete QuickSortSubVector code here
28.     // Hint : recursively call itself
29.     //              Partition function is needed
30. }
31.
32. int SortTool::Partition(vector<int>& data, int low, int high) {
33.     // Function : Partition the vector
34.     // TODO : Please complete the function
35.     // Hint : Textbook page 171
36. }
37.
38. // Merge sort method
39. void SortTool::MergeSort(vector<int>& data){
40.     MergeSortSubVector(data, 0, data.size() - 1);
41. }
42.
43. // Sort subvector (Merge sort)
44. void SortTool::MergeSortSubVector(vector<int>& data, int low, int high) {
45.     // Function : Merge sort subvector
46.     // TODO : Please complete MergeSortSubVector code here
47.     // Hint : recursively call itself
48.     //              Merge function is needed
49. }
50.
51. // Merge
52. void SortTool::Merge(vector<int>& data, int low, int middle1, int middle2, int high) {
53.     // Function : Merge two sorted subvector
54.     // TODO : Please complete the function
55. }
56.
57. // Heap sort method
58. void SortTool::HeapSort(vector<int>& data) {
59.     // Build Max-Heap
60.     BuildMaxHeap(data);
61.     // 1. Swap data[0] which is max value and data[i] so that the max value will be in correct location
62.     // 2. Do max-heapify for data[0]
63.     for (int i = data.size() - 1; i >= 1; i--) {
64.         swap(data[0],data[i]);
65.         heapSize--;
66.         MaxHeapify(data,0);
67.     }
68. }
69.
70. //Max heapify
71. void SortTool::MaxHeapify(vector<int>& data, int root) {
72.     // Function : Make tree with given root be a max-heap if both right and left sub-tree are max-heap
73.     // TODO : Please complete max-heapify code here
74. }
75.
76. //Build max heap
77. void SortTool::BuildMaxHeap(vector<int>& data) {
78.     heapSize = data.size(); // initialize heap size
79.     // Function : Make input data become a max-heap
80.     // TODO : Please complete BuildMaxHeap code here
81. }
```

**sort_tool.cpp**

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

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

Line 25-30: please complete the function of sorting sub-vector using quick sort algorithm here.

Line 32-36: please complete the function of partition here.

Line 39-41: the function of merge sort will call function of Sorting sub-vector and give initial lower/upper bound.

Line 44-49: please complete the function of sorting sub-vector using merge sort algorithm here.

Line 52-55: please complete the function of merging two sorted sub-vector here.

Line 58-68: the function of heap sort will build max-heap first. And then, exchange data iteratively.

Line 71-74: 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.

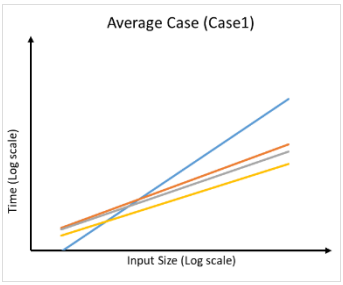
Line 77-81: please complete the function of building max-heap with given input data.

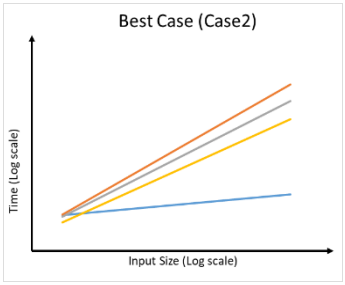
## Requirements:

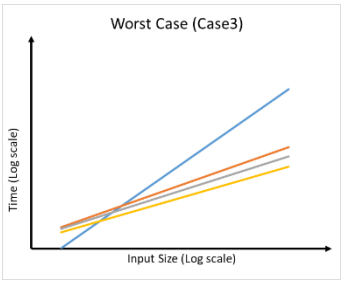
1. Please check the source code files under the src directory. You may need to complete the functions of class SortTool in *sort_tool.cpp*. You can also modify *main.cpp* and *sort_tool.h* if you think it is necessary.
2. Your source code must be written in C or C$^{++}$. The code must be executable on EDA union lab machines.
3. In your report, compare the running time of four 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 | | QS | | HS | |
|---|---|---|---|---|---|---|---|---|
| | CPU time (s) | Memory (KB) | CPU time (s) | Memory (KB) | CPU time (s) | Memory (KB) | CPU time (s) | Memory (KB) |
| 4000.case2 | | | | | | | | |
| 4000.case3 | | | | | | | | |
| 4000.case1 | | | | | | | | |
| 16000.case2 | | | | | | | | |
| 16000.case3 | | | | | | | | |
| 16000.case1 | | | | | | | | |
| 32000.case2 | | | | | | | | |
| 32000.case3 | | | | | | | | |
| 32000.case1 | | | | | | | | |
| 1000000.case2 | | | | | | | | |
| 1000000.case3 | | | | | | | | |
| 1000000.case1 | | | | | | | | |

4. In your report, draw figures to show the growth of run time as a function of input size and try to analyze the slopes of the curves as well as their relation (as the following example, where each curve represents an algorithm.) Please note that you should transfer the run time and input size to log scale first, then draw the figures. Your figures should be clear and easy to distinguish the curves.




You can skip the test case if the run time is more than 3 minutes.

5. Notice: You are not allowed to include the header <algorithm> or <queue> in STL!

## Compile

We expect 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|QS|HS] <input_file_name> <output_file_name>
```

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

## Control the stack size

To prevent stack overflow cause by the recursion function calls, please set the stack size to 256MB using the following Linux comment:

```
ulimit -s 262144
```

```
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.

Makefile Tutorial By Example

---

## Validation:

You can verify your answer very easily by comparing your output with case2 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 <input_file> <your_output_file>
```

Please notice that it 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:

You need to create a directory named **<student_id>_pa1/** (e.g. b09901000_pa1/) (**student id should start with a lowercase letter**) which must contain the following materials:

1. A directory named **src/** contains your source codes: only `*.h, *.hpp, *.c, *.cpp` are allowed in src/, and no directories are allowed in src/;
2. A directory named **bin/** containing your executable binary named **NTU_sort**;
3. A directory named **doc/** containing your report;
4. A makefile named **makefile** that produces an executable binary from your source codes by simply typing “make”: the binary should be generated under the directory <student_id>_pa1/bin/;
5. A text readme file named **README** describing how to compile and run your program;
6. A report named **report.pdf** on the data structures used in your program and your findings in this programming assignment.

We will use our own test cases, so do NOT include the input files.

In summary, you should at least have the following items in your *.tgz file.

src/<all your source code>
lib/<library file>
bin/NTU_sort
doc/report.pdf
makefile
README

The submission filename should be compressed in a single file <student_id>_pa1.tgz. (e.g. b09901000_pa1.tgz). You can use the following command to compress a whole directory:

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

For example, go to the same level as PA1 directory, and type

```
tar -zcvf b09901000_pa1.tgz b09901000_pa1/
```

Please submit a single *.*tgz* file to NTU COOL system before **10/24(Sun.) 13:00**.

You are required to run the checksubmitPA1 script to check if your .tgz submission file is correct. Suppose you are in the same level as PA1 directory

```
bash ./PA1/utility/checkSubmitPA1.sh b09901000_pa1.tgz
```

Please note the path must be correct. If you are located in the ~/ directory, then './PA1/utility/checkSubmitPA1.sh' means the path ~/PA1/utility/checkSubmitPA1.sh and b09901000_pa.tgz means the path ~/b99901000_pa1.tgz

Your program will be graded by automatic grading script. Any mistake in the submission will cost at least 20% penalty of your score. Please be very careful in your submission.

## Grading:

70% correctness (including submission correctness and implementation correctness)
20% file format and location
10% report

## NOTE:

1. TA will check your source code carefully. Copying other source code can result in zero grade for all students involved.
2. Implementation correctness means to follow the guideline on the handout to write the codes. Wrong implementation will result in penalty even if the output is correct.