National Taiwan University
Department of Electrical Engineering
Algorithms, Fall 2024

Programming Assignment #1

Sorting

Submission Due: 12:00 noon, October 6 (Sunday), 2024

TA in charge: Meng-Chen Wu, r12943107@ntu.edu.tw,
or ask questions in Algorithms @ NTUEE on FB

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. Decompress 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 in given cases. 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
```

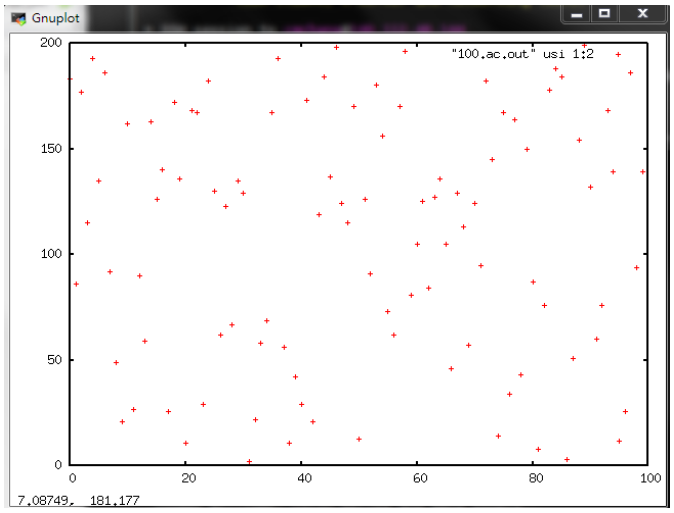
PLOT:

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

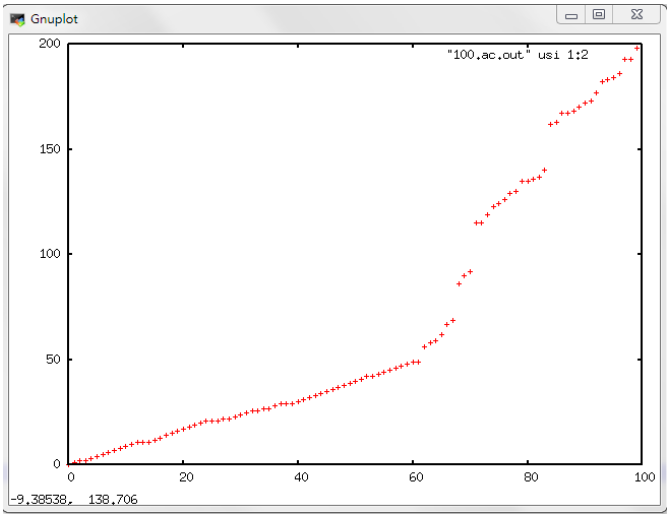
```
set xrange [0:5]
set yrange [0:20]
plot "5.case1.in" usi 1:2
plot "5.case1.out" usi 1:2

# 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|BMS|QS|RQS] <input_file_name> <output_file_name>
```

where IS represents insertion sort, MS is merge sort (top-down merge sort, as in textbook), BMS is bottom-up merge sort, QS is quick sort, and RQS is randomized quick sort. The square bracket with vertical bar '[IS|MS|BMS|QS|RQS]' means that only one of the five algorithms 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
```

You are suggested to refer to the pseudo code in our textbook for IS, MS, QS, and RQS. For BMS, you could refer to **Algorithm 1**, *BottomUpMergeSort*. *data* is the input unsorted array. *data.size* is the number of input data. We consider *data* as *data.size* groups of one data in each group. *numGroup* is the current number of groups. *groupMem* is the number of data in a group. For each iteration of the **while** loop at line 3, we merge every two nearby groups to one sorted group. At line 6, *data*[*x* : *y*] denotes all elements from *data*[*x*] to *data*[*y*]. At line 8, 'Merge' refers to the same procedure of top-down merge sort, which merge two sorted arrays to one sorted array. After merging, we should update the number of groups and number of data in one group at the end of one iteration, at line 12-13. When **while** loop at line 3 terminates, *data* itself is one sorted group.

Algorithm 1 $BottomUpMergeSort(data)$

1: $numGroup = data.size$
2: $groupMem = 1$
3: **while** $numGroup > 1$ **do**
4: $\quad i = 1$
5: $\quad$ **while** $i \leq data.size$ **do**
6: $\quad\quad a = data[i : i + groupMem - 1]$
7: $\quad\quad b = data[i + groupMem : i + groupMem * 2 - 1]$
8: $\quad\quad$ Merge a and b to one sorted group c
9: $\quad\quad data[i : i + groupMem * 2 - 1] = c$
10: $\quad\quad i = i + groupMem * 2$
11: $\quad$ **end while**
12: $\quad$ Update $numGroup$ to current number of groups
13: $\quad groupMem = groupMem * 2$
14: **end while**

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 2024 fall Algorithm PA1]
5.  //   Modify         [2020/9/15 Mu-Ting Wu]
6.  //   Modify         [2023/9/12 Ming-Bang Fan]
7.  //   Modify         [2024/9/2 Meng-Chen Wu]
8.  // ****************************************************************************
9.
10. #include <cstring>
11. #include <iostream>
12. #include <fstream>
13. #include "../lib/tm_usage.h"
14. #include "sort_tool.h"
15.
16. using namespace std;
17.
18. void help_message() {
19.     cout << "usage: NTU_sort -[IS|MS|BMS|QS|RQS|HS] <input_file> <output_file>" << endl;
20.     cout << "options:" << endl;
21.     cout << "    IS    - Insersion Sort" << endl;
22.     cout << "    MS    - Merge Sort" << endl;
23.     cout << "    BMS - Bottom-up Merge Sort" << endl;
24.     cout << "    QS    - Quick Sort" << endl;
25.     cout << "    RQS - Randomized Quick Sort" << endl;
26.     cout << "    HS    - Heap Sort" << endl;
27. }
28.
29. int main(int argc, char* argv[])
30. {
31.     if(argc != 4) {
32.         help_message();
33.         return 0;
34.     }
35.     CommonNs::TmUsage tmusg;
36.     CommonNs::TmStat stat;
37.
38.     //////////// read the input file ////////////
39.
40.     char buffer[200];
41.     fstream fin(argv[2]);
42.     fstream fout;
43.     fout.open(argv[3],ios::out);
44.     fin.getline(buffer,200);
45.     fin.getline(buffer,200);
46.     int junk,num;
47.     vector<int> data;
48.     while (fin >> junk >> num)
49.         data.push_back(num); // data[0] will be the first data.
50.                              // data[1] will be the second data and so on.
51.
52.     //////////// the sorting part ////////////////
53.     tmusg.periodStart();
54.     SortTool NTUSortTool;
55.
56.     string mode(argv[1]);
57.     if(mode == "-IS") {
58.         NTUSortTool.InsertionSort(data);
59.     }
60.     else if(mode == "-MS") {
61.         NTUSortTool.MergeSort(data);
62.     }
63.     else if(mode == "-BMS") {
64.         NTUSortTool.BottomUpMergeSort(data);
65.     }
66.     else if(mode == "-QS") {
67.         NTUSortTool.QuickSort(data, 0);
68.     }
69.     else if(mode == "-RQS") {
70.         NTUSortTool.QuickSort(data, 1);
71.     }
72.     else if(mode == "-HS") {
73.         NTUSortTool.HeapSort(data);
74.     }
75.     else {
76.         help_message();
77.         return 0;
78.     }
79.     tmusg.getPeriodUsage(stat);
80.     cout <<"The total CPU time: " << (stat.uTime + stat.sTime) / 1000.0 << "ms" << endl;
81.     cout <<"memory: " << stat.vmPeak << "KB" << endl; // print peak memory
82.
83.     //////////// write the output file //////////
84.     fout << "# " << data.size() << " data points" <<endl;
85.     fout << "# index number" << endl;
86.     for (int i = 0; i < data.size(); i++)
87.         fout << i << " " <<data[i] << endl;
88.     fin.close();
89.     fout.close();
90.     return 0;
91. }
```

main.cpp

Line 40-50: parse unsorted data from input file and push them into the vector.

Line 56-78: call different function depending on given command.

Line 84-87: 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.  //   Modify         [2023/9/12 Ming-Bang Fan]
7.  //   Modify         [2024/9/2 Meng-Chen Wu]
8.  // ************************************************************************
9.
10. #ifndef _SORT_TOOL_H
11. #define _SORT_TOOL_H
12.
13. #include <vector>
14. using namespace std;
15.
16. class SortTool {
17.     public:
18.                             SortTool(); // constructor
19.             void            InsertionSort(vector<int>&); // sort data using insertion sort
20.             void            MergeSort(vector<int>&); // sort data using merge sort
21.             void            BottomUpMergeSort(vector<int>&); // sort data using bottom-up merge sort
22.             void            QuickSort(vector<int>&, int); // sort data using quick sort
23.             void            HeapSort(vector<int>&); // sort data using heap sort
24.     private:
25.             void            QuickSortSubVector(vector<int>&, int, int, const int); // quick sort subvector
26.             int             RandomizedPartition(vector<int>&, int, int); // randomized partition the subvector
27.             int             Partition(vector<int>&, int, int); // partition the subvector
28.             void            MergeSortSubVector(vector<int>&, int, int); // merge sort subvector
29.             void            Merge(vector<int>&, int, int, int, int); // merge two sorted subvector
30.             void            MaxHeapify(vector<int>&, int); // make tree with given root be a max-heap
31.                                                                 //if both right and left sub-tree are max-heap
32.             void            BuildMaxHeap(vector<int>&); // make data become a max-heap
33.             int             heapSize; // heap size used in heap sort
34.
35. };
36.
37. #endif
```

sort_tool.h

Line 19-23: Sort function which will be called in *main.cpp.*

Line 25: 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 26-27: These functions will be used in quick sort and should be implemented to partition the sub vector.

Line 28: 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 29: This function will be used in merge sort and should be implemented to merge two sorted sub vectors.

Line 30: 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 32: 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.  //   Modify         [2023/9/12 Ming-Bang Fan]
7.  //   Modify         [2024/9/2 Meng-Chen Wu]
8.  // ************************************************************************
9.
10. #include "sort_tool.h"
11. #include<iostream>
12. // Constructor
13. SortTool::SortTool() {}
14.
15. // Insertsion sort method
16. void SortTool::InsertionSort(vector<int>& data) {
17.     // Function : Insertion sort
18.     // TODO : Please complete insertion sort code here
19. }
20.
21. // Quick sort method
22. void SortTool::QuickSort(vector<int>& data,int f){
23.     QuickSortSubVector(data, 0, data.size() - 1, f);
24. }
25. // Sort subvector (Quick sort)
26. void SortTool::QuickSortSubVector(vector<int>& data, int low, int high, const int flag) {
27.     // Function : Quick sort subvector
28.     // TODO : Please complete QuickSortSubVector code here
29.     // Hint : recursively call itself
30.     //            Partition function is needed
31.     // flag == 0 -> normal QS
32.     // flag == 1 -> randomized QS
33. }
34. int SortTool::RandomizedPartition(vector<int>& data, int low, int high){
35.     // Function : RQS's Partition the vector
36.     // TODO : Please complete the function
37. }
38. int SortTool::Partition(vector<int>& data, int low, int high) {
39.     // Function : Partition the vector
40.     // TODO : Please complete the function
41. }
42.
43. // Merge sort method
44. void SortTool::MergeSort(vector<int>& data){
45.     MergeSortSubVector(data, 0, data.size() - 1);
46. }
47.
48. // Sort subvector (Merge sort)
49. void SortTool::MergeSortSubVector(vector<int>& data, int low, int high) {
50.     // Function : Merge sort subvector
51.     // TODO : Please complete MergeSortSubVector code here
52.     // Hint : recursively call itself
53.     //            Merge function is needed
54. }
55.
56. // Merge
57. void SortTool::Merge(vector<int>& data, int low, int middle1, int middle2, int high) {
58.     // Function : Merge two sorted subvector
59.     // TODO : Please complete the function
60. }
61.
62. // Bottom-up merge sort method
63. void SortTool::BottomUpMergeSort(vector<int>& data)
64. {
65.     // Function : Bottom-up merge sort, sorting is done in this function only
66.     // TODO :
67.     // Implement merge sort in bottom-up style, in other words,
68.     // without recursive function calls.
69.     // Hint:
70.     // 1. Divide data to n groups of one data each group
71.     // 2. Iteratively merge each pair of 2 neighbor groups into one larger group
72.     // 3. Finally we obtain exactly one sorted group
73. }
```

sort_tool.cpp

Line 16-19: please complete the function of insertion sort here.

Line 22-24: the function of quick sort will call function of Sorting sub-vector and give initial lower/upper bound.

Line 26-33: please complete the function of sorting sub-vector using quick sort algorithm here.

Line 34-37: please complete the function of randomized quick sort's partition here.

Line 38-41: please complete the function of partition here.

Line 44-46: the function of merge sort will call function of sorting sub-vector and give initial lower/upper bound.

Line 49-54: please complete the function of sorting sub-vector using merge sort algorithm here.

Line 57-60: please complete the function of merging two sorted sub-vector here.

Line 63-73: please complete the function of bottom-up merge sort here.

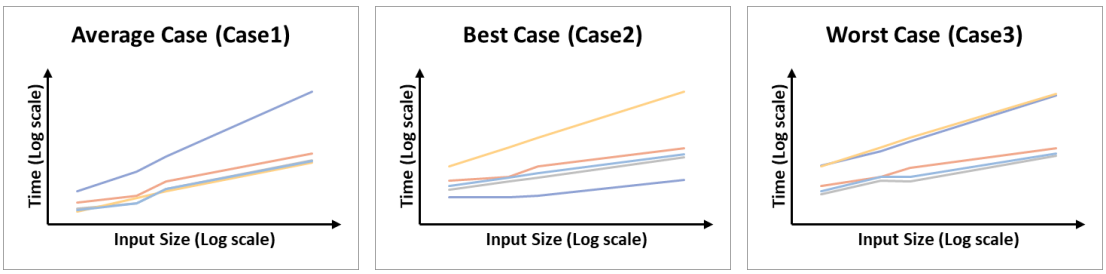
Requirements:

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 five sorting algorithms of different input sizes. Please fill in the following table. Please use –O2 optimization and turn off all debugging message. You should specify where you run your code to obtain the data, on EDA union lab machines or your local terminal.

Input size	IS		MS		BMS		QS		RQS	
	CPU time (s)	Memory (KB)	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, plot the trendline of five sorting algorithms 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. If your trendline is different from following example, you should give reasonable explanation. For example: You may find out quick sort have same time tendency as insertion sort in worst case, why? How to solve this?

Your figures should be clear and easy to distinguish the curves.



5. There is heap sort (HS) function structure in provided source code, but be aware that you are only required to finish IS, MS, BMS, QS and RQS.
6. 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|BMS|QS|RQS] <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 recursive function calls, please set the stack size to 256MB using the following Linux command:

```
ulimit -s 262144
```

```
# CC and CFLAGS are varilables
CC = g++
CFLAGS = -c
AR = ar
ARFLAGS = rcv
# -c option ask g++ to compile the source files, but do not link.
# -g option is for debugging version
# -O2 option is for optimized version
DBGFLAGS = -g -D_DEBUG_ON_
OPTFLAGS = -O2
# make all
all: bin/NTU_sort
	@echo -n ""

# optimized version
bin/NTU_sort: sort_tool_opt.o main_opt.o lib
	$(CC) $(OPTFLAGS) sort_tool_opt.o main_opt.o -ltm_usage -Llib -o bin/NTU_sort
main_opt.o: src/main.cpp lib/tm_usage.h
	$(CC) $(CFLAGS) $< -Ilib -o $@
sort_tool_opt.o: src/sort_tool.cpp src/sort_tool.h
	$(CC) $(CFLAGS) $(OPTFLAGS) $< -o $@

# DEBUG Version
dbg: bin/NTU_sort_dbg
	@echo -n ""

bin/NTU_sort_dbg: sort_tool_dbg.o main_dbg.o lib
	$(CC) $(DBGFLAGS) sort_tool_dbg.o main_dbg.o -ltm_usage -Llib -o bin/NTU_sort_dbg
main_dbg.o: src/main.cpp lib/tm_usage.h
	$(CC) $(CFLAGS) $< -Ilib -o $@
sort_tool_dbg.o: src/sort_tool.cpp src/sort_tool.h
	$(CC) $(CFLAGS) $(DBGFLAGS) $< -o $@

lib: lib/libtm_usage.a

lib/libtm_usage.a: tm_usage.o
	$(AR) $(ARFLAGS) $@ $<
tm_usage.o: lib/tm_usage.cpp lib/tm_usage.h
	$(CC) $(CFLAGS) $<

# clean all the .o and executable files
clean:
	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**, including all contents mentioned in **Grading** section.

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 **10/6(Sun.) 12:00 noon**.

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

If you see "Permission denied", you should use following command to fix the problem.

```
chmod +x <checker name>
```

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:

- Correctness (60%)
  - Including output result correctness and implementation correctness. 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.
  - There will be hidden test cases for grading. Number of unsorted data is between 1 and 1,000,000. Range of data is between -2,147,483,648 and 2,147,483,647.
  - Runtime limit for each test cases depend on the tested algorithm. We will kill your job if its runtime reaches the limit.

    IS, QS: 20 minutes

    MS, BMS, RQS: 1 minute
  - TA will check your source code carefully. Copying other source code can result in zero grade for all students involved.
- File format and location (20%)
- Report (20%)
  - Table of runtime and memory usage of five sorting algorithms. Trendline plot with slope calculation. Compare your slope with the complexity in the textbook. Please explain why or why not they match. (5%)
  - Comparison between MS and BMS, including runtime difference and explanation. (5%)
  - Comparison between QS and RQS, including runtime difference and explanation. (5%)
  - Data structure used and other findings in this programming assignment. (5%)

Frequently Asked Questions:

Q: Should we take care of the error handling issue of the testbench?

A: The testbench for this PA is correct. However, it is always better to have your error handling protection.

Q: Where is my EDA Union account and password?

A: NTU COOL > 成績 > Account & Password > 評語.

Q: How to log in to the server?

A: Please see the week 1 recitation file on NTU COOL.
Someone cannot log in by
"ssh alg24fxxx@edaunion.ee.ntu.edu.tw -p <port>"
can try another command
"ssh -p <port> alg24fxxx@edaunion.ee.ntu.edu.tw".

Your EDA Union account (a.k.a Web ID) is "alg24fXXX" starts with the lower letter "a".

Q: Can I use C++11? Can I use Mac OS? Can I use …?

A: Yes, you can. However, we will use and only use the EDA Union server for evaluation. So whatever platform/setting you use, test on the EDA Union server before you submit your final version. The submission that fails to compile and/or run on the EDA Union server will be severely penalized.

Q: Runtime warning message: cannot get memory usage

A: Our code does not support checking memory usage on Mac OS. To check memory usage, compile and run the code on the EDA Union server.

Q: Quicksort runtime error message: segmentation fault

A: If the code is correct, this might be caused by stack overflow. Also, other memory-related issues might be reported in the error message. Enlarging the stack size would help. See section "Control the stack size" in the pdf.

Q: Do the submission files include the lib/ folder?

A: Yes. You are also allowed to modify the codes in lib/, so it's recommended that you attach them in your submission.

Q: Permission denied when using PA1 checker.

A: Type "chmod u+x ./utility/PA1_result_checker" and run again.

Q: If there is any compile error (e.g. "Error: expecting string instruction after `rep'")

A: "Google" it first or try to compile it on another server.