**Algorithms**

Programming Assignment #1

Sorting

**Submission URL & Online Resources:**

<https://cool.ntu.edu.tw/courses/5037>

**Introduction:**

In this PA, you are required to implement various sorters that we learnt in class. You can download the *PA1.tgz* file from NTU COOL website. Decompress it using Linux command.

tar zxvf PA1.tgz

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 |
| 66 | 77 |

**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 10000.case1.into 10000.case1.outusing insertion sort.

./bin/NTU\_sort -IS inputs/10000.case1.in outputs/10000.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  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81 | // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // File [main.cpp]  // Author [Yu-Hao Ho]  // Synopsis [The main program of 2019 fall Algorithm PA1]  // Modify [2020/9/15 Mu-Ting Wu]  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  #include <cstring>  #include <iostream>  #include <fstream>  #include "../lib/tm\_usage.h"  #include "sort\_tool.h"  using namespace std;  void help\_message() {  cout << "usage: NTU\_sort -[IS|MS|QS|HS] <input\_file> <output\_file>" << endl;  cout << "options:" << endl;  cout << " IS - Insersion Sort" << endl;  cout << " MS - Merge Sort" << endl;  cout << " QS - Quick Sort" << endl;  cout << " HS - Heap Sort" << endl;  }  int main(int argc, char\* argv[])  {  if(argc != 4) {  help\_message();  return 0;  }  CommonNs::TmUsage tmusg;  CommonNs::TmStat stat;  //////////// read the input file /////////////    char buffer[200];  fstream fin(argv[2]);  fstream fout;  fout.open(argv[3],ios::out);  fin.getline(buffer,200);  fin.getline(buffer,200);  int junk,num;  vector<int> data;  while (fin >> junk >> num)  data.push\_back(num); // data[0] will be the first data.  // data[1] will be the second data and so on.    //////////// the sorting part ////////////////  tmusg.periodStart();  SortTool NTUSortTool;  if(!strcmp(argv[1],"-QS")) {  NTUSortTool.QuickSort(data);  }  else if(!strcmp(argv[1],"-IS")) {  NTUSortTool.InsertionSort(data);  }  else if(!strcmp(argv[1],"-MS")) {  NTUSortTool.MergeSort(data);  }  else if(!strcmp(argv[1],"-HS")) {  NTUSortTool.HeapSort(data);  }  else {  help\_message();  return 0;  }  tmusg.getPeriodUsage(stat);  cout <<"The total CPU time: " << (stat.uTime + stat.sTime) / 1000.0 << "ms" << endl;  cout <<"memory: " << stat.vmPeak << "KB" << endl; // print peak memory  //////////// write the output file ///////////  fout << "# " << data.size() << " data points" <<endl;  fout << "# index number" << endl;  for (int i = 0; i < data.size(); i++)  fout << i << " " <<data[i] << endl;  fin.close();  fout.close();  return 0;  } | |
| 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  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // File [sort\_tool.h]  // Author [Yu-Hao Ho]  // Synopsis [The header file for the SortTool Class]  // Modify [2020/9/15 Mu-Ting Wu]  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  #ifndef \_SORT\_TOOL\_H  #define \_SORT\_TOOL\_H  #include<vector>  using namespace std;  class SortTool {  public:  SortTool(); // constructor  void InsertionSort(vector<int>&); // sort data using insertion sort  void MergeSort(vector<int>&); // sort data using merge sort  void QuickSort(vector<int>&); // sort data using quick sort  void HeapSort(vector<int>&); // sort data using heap sort  private:  void QuickSortSubVector(vector<int>&, int, int); // quick sort subvector  int Partition(vector<int>&, int, int); // partition the subvector  void MergeSortSubVector(vector<int>&, int, int); // merge sort subvector  void Merge(vector<int>&, int, int, int, int); // merge two sorted subvector  void MaxHeapify(vector<int>&, int); // make tree with given root be a max-heap  //if both right and left sub-tree are max-heap  void BuildMaxHeap(vector<int>&); // make data become a max-heap  int heapSize; // heap size used in heap sort    };  #endif | |
| 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 vector. |
| 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  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83 | // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // File [sort\_tool.cpp]  // Author [Yu-Hao Ho]  // Synopsis [The implementation of the SortTool Class]  // Modify [2021/2/26 Hsien-Chia Chen]  // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  #include "sort\_tool.h"  #include<iostream>  // Constructor  SortTool::SortTool() {}  // Insertsion sort method  void SortTool::InsertionSort(vector<int>& data) {  // Function : Insertion sort  // TODO : Please complete insertion sort code here  }  // Quick sort method  void SortTool::QuickSort(vector<int>& data){  QuickSortSubVector(data, 0, data.size() - 1);  }  // Sort subvector (Quick sort)  void SortTool::QuickSortSubVector(vector<int>& data, int low, int high) {  // Function : Quick sort subvector  // TODO : Please complete QuickSortSubVector code here  // Hint : recursively call itself  // Partition function is needed  }  int SortTool::Partition(vector<int>& data, int low, int high) {  // Function : Partition the vector  // TODO : Please complete the function  // Hint : Textbook page 171  }  // Merge sort method  void SortTool::MergeSort(vector<int>& data){  MergeSortSubVector(data, 0, data.size() - 1);  }  // Sort subvector (Merge sort)  void SortTool::MergeSortSubVector(vector<int>& data, int low, int high) {  // Function : Merge sort subvector  // TODO : Please complete MergeSortSubVector code here  // Hint : recursively call itself  // Merge function is needed  }  // Merge  void SortTool::Merge(vector<int>& data, int low, int middle1, int middle2, int high) {  // Function : Merge two sorted subvector  // TODO : Please complete the function  }  // Heap sort method  void SortTool::HeapSort(vector<int>& data) {  // Build Max-Heap  BuildMaxHeap(data);  // 1. Swap data[0] which is max value and data[i] so that the max value will be in correct location  // 2. Do max-heapify for data[0]  for (int i = data.size() - 1; i >= 1; i--) {  int tmp = data[i];  data[i] = data[0];  data[0] = tmp;  heapSize--;  MaxHeapify(data,0);  }  }  //Max heapify  void SortTool::MaxHeapify(vector<int>& data, int root) {  // Function : Make tree with given root be a max-heap if both right and left sub-tree are max-heap  // TODO : Please complete max-heapify code here  }  //Build max heap  void SortTool::BuildMaxHeap(vector<int>& data) {  heapSize = data.size(); // initialize heap size  // Function : Make input data become a max-heap  // TODO : Please complete BuildMaxHeap code here  } | |
| Line 15-18: | | (TO-DO) The function of insertion sort*.* |
| Line 21-23: | | The function of quick sort will call function of Sorting sub-vector and give initial lower/upper bound. |
| Line 25-30: | | (TO-DO) The function of sorting sub-vector using quick sort algorithm. |
| Line 32-36: | | (TO-DO) The function of partition. |
| Line 39-41: | | The function of merge sort will call function of Sorting sub-vector and give initial lower/upper bound. |
| Line 44-49: | | (TO-DO) The function of sorting sub-vector using merge sort algorithm. |
| Line 52-55: | | (TO-DO) The function of merging two sorted sub-vector. |
| Line 58-70: | | The function of heap sort will build max-heap first. And then, exchange data iteratively. |
| Line 73-76: | | (TO-DO) 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 79-83: | | (TO-DO) 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 running time as a function of input size and try to analyze the curve (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 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

*makefile*:

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | # 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  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 lib/\*.o bin/\* | |
| 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 object 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 …] |
| Line 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*. |
| Line 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*. |
| Line13, 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 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*** describing 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 **3/28 (Sun.) 13:00**.

You are required to run the *checksubmitPA1.sh* 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.