ARIZONA STATE UNIVERSITY

CSE 310 — Data Structures and Algorithms — Spring 2016

Project #1

Available 01/19/2016; milestone due 02/04/2016; complete project due 02/23/2016
See Blackboard for submission time requirements for your CSE 310 section

This project has two major goals:

1. The first is to implement various sorting and selection algorithms both in serial and in parallel.
2. The second is to run experiments comparing the performance of certain algorithms, to collect and plot data, and to interpret the results.

Note: This project is to be completed individually. Your implementation must use C/C++ and your code must run on the Linux machine `general.asu.edu`.

All dynamic memory allocation must be done yourself, i.e., using either `malloc()` and `free()`, or `new()` and `delete()`. You may not use any external libraries to implement any part of this project.

See §2 for full project requirements. A script will be used to check the correctness of your program. Therefore, absolutely no changes to these project requirements are permitted.

You should use a version control system as you develop your solution to this project, e.g., Dropbox or GitHub. Your code repository should be private to prevent anyone from plagiarizing your work.

1 Super Bowl 50

The 50th Super Bowl will be played in 2016. A mountain of statistics is gathered for each game, player category, player position, and team. In this project you will write a serial program that executes a sequence of commands that operate on game statistics over a number of years.

All input data is to be read from standard input (`stdin`). Of course, you may (and should) redirect `stdin` from a file. No file operations are allowed.

You may assume the format of the input data is correct.

The format of the input data is as follows:

- The number of years y (integer) of team game statistics in this data set.
- For each of y years, the following is provided:
  - The year for which statistics are provided (integer).
  - For each of 32 teams, numerous statistics are provided in a tab separated list. See the definition of `struct team_stats` in §1.1.

Upon reading the value of y, you must dynamically allocate an array of `struct annual_stats` of size y, and then initialize the array with the data following. Once the data structure is initialized, a sequence of commands is to be processed. The input continues with:

- The number of commands c (integer).
- c commands following the format in §1.2.

Each of the c commands must be processed as described in §1.2.

1.1 Format of Team Data

For each team, the structure team_stats contains numerous fields associated with the given team for the given year. The structure annual_stats holds team statistics for all teams in a given year. A file named defns.cc that contains these structure definitions is provided for you.

The data for this project is drawn from the team statistics page of the National Football League (NFL). See http://www.nfl.com/stats/team.

```
#define NO_TEAMS      32 // Number of NFL teams
#define TEAM_NAME_LEN 25 // Maximum team name string length
#define TOP_LEN        6 // Maximum time of possession string length

struct team_stats{
   char team_name[ TEAM_NAME_LEN ]; // Name of NFL team
   int games; // Number of games played in the season
   float pts_per_game; // Points per game
   int total_points; // Total points
   int scrimmage_plays; // Scrimmage plays
   float yds_per_game; // Yards per game
   float yds_per_play; // Yards per play
   float first_per_game; // First downs per game
   int third_md; // Third down conversions
   int third_att; // Third down attempts
   int third_pct; // Third down percentage
   int fourth_md; // Fourth down conversions
   int fourth_att; // Fourth down attempts
   int fourth_pct; // Fourth down percentage
   int penalties; // Number of penalties
   int pen_yds; // Penalty yards
   char top_per_game[ TOP_LEN ]; // Time of possession per game
   int fum; // Number of fumbles
   int lost; // Fumbles lost
   int to; // Turnover ratio
};

struct annual_stats{
   int year;
   struct team_stats teams[ NO_TEAMS ];
};
```

1.2 Commands

There are eight (8) commands to implement. The syntax of valid commands is:

- `bsort year field order`, use the Bubblesort algorithm to sort team data for the given `year` on the given `field` in the given order. The output is a list of team name, and field name, with appropriate headers on the list.
  - If the `year` given is not one of the years this is part of the input data, the error message: `Error: no such year.` should be output.
  - The `field` may be any field of `struct team_stats`. This structure contains fields of character strings, integers, and floating point numbers. Hence the Bubblesort algorithm should be able to sort any of these types.

  - The order may be either `incr` for increasing order, and `decr` for decreasing order. If there are multiple fields with the same order, then they should be sorted in alphabetic order by team name.

- `qsort year field order`, use the Quicksort algorithm to sort team data for the given `year` on the given `field` in the given `order`. Use the leftmost element as the pivot. See the command `bsort` for a description of the `year`, `field`, and `order`, as well as the format of the output.

- `bfind year field item`, first uses Bubblesort to sort the team data for the given `year` on the given `field` in increasing order. Then, a selection is made based on `item`. Valid `items` for selection include:

  - `max`, prints the maximum value for the given `field` in the given year, along with the team name(s) achieving the maximum.
  - `min`, prints the minimum value for the given `field` in the given year, along with the team name(s) achieving the minimum.
  - `average`, prints the average value for the given `field` in the given `year` over all teams. Only an integer or floating point `field` will be given for this `item`.
  - `median`, prints the median value for the given `field` in the given `year` over all teams. Here, "median" refers to the lower median, i.e., the element at position $\lfloor (n+1)/2 \rfloor$, where n is the total number of elements.

- `qfind year field item`, first uses Quicksort to sort the team data for the given `year` on the given `field` in increasing order, using the leftmost element as the pivot. See the command `bfind` for a description of the `year`, `field`, and `item`, as well as the format of the output.

- `bsort range start-year end-year field order`, use the Bubblesort algorithm to sort team data for the range of years starting with `start-year` and ending with `end-year`, on the given `field` in the given `order`. There are two differences between this command and `bsort` for a single year.

  - The sort is to be performed on data for all years in the given range. You may assume that `start-year` < `end-year` and that all years exist in the input data provided.
  - Add the year as part of the output for clarity.

- `qsort range start-year end-year field order`, use the Quicksort algorithm to sort team data for the range of years starting with `start-year` and ending with `end-year`, on the given `field` in the given `order`. See the command `bsort` on a range of years for a description of the `start-year`, `end-year`, `field`, and `order`, as well as the format of the output.

- `pmax year field`, uses a parallel divide-and-conquer algorithm implemented using two threads in OpenMP to determine the maximum value for the given `field` in the given `year`, along with the team name(s) achieving the maximum. Hint: The presentation in the file `Parallel-Intro.pptx` will be reviewed in class; a copy of this presentation is also provided to you. It illustrates a parallel divide-and-conquer approach to sum values stored in an array. A similar idea may be used to compute the maximum element.

- `pmin year field`, uses a parallel divide-and-conquer algorithm implemented using two threads in OpenMP to determine the minimum value for the given `field` in the given `year`, along with the team name(s) achieving the minimum. (See also the Hint for the `pmax` command.)

Consider the following valid example command sequence. Comments are not part of the input, and are only included here for clarification.

```
8 // A total of 8 commands follow
bsort 2014 penalties decr // Bubblesort 2014 data on penalties in decreasing order
qsort 2015 pts_per_game decr // Quicksort 2015 data on points per game in decreasing order
bfind 2015 yds_per_game max // Return maximum yards per game after Bubblesorting 2015 data
bfind 2014 fum average // Return average of number of fumbles after Bubblesorting 2014 data
qfind 2015 top_per_game min // Return minimum time of possession per game after Quicksorting 2015 data
bsort range 2014 2015 penalties incr // Bubblesort 2014-2015 data on penalties in increasing order
pmax 2015 yds_per_play // Use parallel algorithm to return maximum yards per play in 2015
pmin 2014 fourth_pct // Use parallel algorithm to return minimum fourth down percentage in 2014
```

2 Program Requirements for Project #1

1. Write a C/C++ program that implements all of the commands described in §1.2 on data in the format described in §1. You must use OpenMP to implement the commands `pmax` and `pmin` in parallel using a divide-and-conquer approach with two threads; all other commands are to be implemented in serial.
2. Design experiments that exercise your program to answer the questions in §3. A brief report with figures and data to support your answers is expected.
3. Provide a `Makefile` that compiles your program into an executable named `p1`. This executable must be able to run commands read from standard input directly, or from a script file redirected from standard input (this should require no change to your program).

Sample input files that adhere to the format described in §1 will be provided on Blackboard; use them to test the correctness of your programs.

3 Experimentation

1. Plot the run time of your Bubblesort algorithm and your Quicksort algorithm on arrays of integers of size $n = \{10^1, 10^2, 10^3, 10^4, 10^5, 10^6\}$ (or larger if you wish). For this you will need to write a separate program that simply calls the Bubblesort and Quicksort algorithms you have implemented to meet the program requirements of Project #1, i.e., use your solution to the `bsort` and `qsort` commands. The `clock` function is one function you may use to obtain the processor time used by your algorithm. A program to generate random integer data will be provided.
   - Do you observe a cross-over point? That is, can you recommend when you should use one algorithm over the other?
2. Plot the run time of maximum finding on arrays of integers of size $n = \{10^1, 10^2, 10^3, 10^4, 10^5, 10^6\}$ (or larger if you wish) using three techniques: Bubblesort the array in increasing order and then select the maximum, Quicksort the array in increasing order and then select the maximum, and finally use your OpenMP parallel divide-and-conquer maximum finding algorithm. Specifically, you should be able to use your solutions to `bfind` and `qfind` on `item=max`, and `pmax`, respectively, for this purpose. Use random integer data for this purpose.
   - Do you observe any cross-over points? That is, is there a size at which the parallel algorithm overtakes either serial algorithm? Is there a point the recursive Quicksort algorithm overtakes the iterative Bubblesort?
   - Can you conclude anything about when to use the iterative, recursive divide-and-conquer, or parallel divide-and-conquer algorithms?

4 Submission Instructions

All submissions are electronic. This project has two submission deadline dates. The time each is due depends on your CSE 310 section; see Blackboard for details.

1. The milestone deadline date is on Thursday, 02/04/2016.
2. The complete project deadline date is on Tuesday, 02/23/2016.

It is your responsibility to submit your project well before the time deadline!!! Late projects will not be accepted. Do not expect the clock on your machine to be synchronized with the one on Blackboard!

Multiple submissions are allowed. The last submission will be graded.

4.1 Requirements for Milestone Deadline

By the milestone deadline, your project must implement the following commands as a minimum: `bsort year`, `bfind year`, `bsort range`.

You should also start collecting run times for the Bubblesort algorithm as described in §3.

Submit electronically, on Thursday, 02/04/2016 using the submission link on Blackboard for the Project #1 milestone, a zip[1] file named `yourFirstName-yourLastName.zip` containing the following:

Project State (5%): In a folder (directory) named State provide a brief report (.txt, .doc, .docx, .pdf) that addresses the following:

1. Describe any problems encountered in your implementation for this project milestone.
2. Describe any known bugs and/or incomplete command implementation for the project milestone.
3. While this project is to be complete individually, describe any significant interactions with anyone (peers or otherwise) that may have occurred.
4. Cite any external code bases, books, and/or websites used or referenced.

Implementation (50%): In a folder (directory) named Code provide:

1. In one or more files, your well documented C/C++ source code implementing the commands required for this project milestone.
2. A `Makefile` that compiles your program to an executable named `p1` on the Linux machine `general.asu.edu`. Our TA will write a script to compile and run all student submissions on `general.asu.edu`; therefore executing the command `make p1` in the Code directory must produce the executable `p1` also located in the Code directory.

Correctness (45%): The correctness of your program will be determined by running a series of commands on a variety of NFL team data, some of which will be provided to you on Blackboard prior to the deadline for testing purposes. For the milestone deadline, the scripts will only contain the subset of the commands described in §1.2 and listed above. As described in §2, your program must take input from standard input directly, or from a file redirected from standard input. You must not use file operations to read the input!

The milestone is worth 30% of the total project grade.

[1] Do not use any other archiving program except `zip`.

4.2 Requirements for Complete Project Deadline

Submit electronically, on Tuesday, 02/23/2016 using the submission link on Blackboard for the complete Project #1, a zip[2] file named `yourFirstName-yourLastName.zip` containing the following:

Project State (5%): Follow the same instructions for Project State as in §4.1.

Experimentation and Report (10%): In a folder (directory) named Report provide a brief report (.txt, .doc, .docx, .pdf) that addresses the following:

1. Describe the experiments you ran.
2. Present figures plotting the results of experimentation as requested in §3 along with your answers to the questions found there. There is no single correct answer! You should just try to explain the results you obtained.

Implementation (40%): Follow the same instructions for Implementation as in §4.1.

Correctness (45%): The same instructions for Correctness as in §4.1 apply except that the input files will exercise all commands from §1.2 rather than a subset of them.

5 Marking Guide

The project milestone is out of 100 marks.

Project State (5%): Summary of project state, use of a zip file, and directory structure required (i.e., a folder/directory named State and Code is provided).

Implementation (50%): 40% for the quality of implementation in your code including proper memory management, 10% for a correct `Makefile`.

Correctness (45%): 40% for correct output on several files of sample input, 5% for redirection from standard input.

The full project is out of 100 marks.

Project State (5%): Summary of project state, use of a zip file, and directory structure required (i.e., a folder/directory named State, Report, and Code is provided).

Experimentation and Report (10%): Experiment design, results of experimentation, and answers to questions.

Implementation (40%): 30% for the quality of implementation in your code including proper memory management, 10% for a correct `Makefile`.

Correctness (45%): 40% for correct output on several files of sample input, 5% for redirection from standard input.

[2] Do not use any other archiving program except `zip`.