IITM-CS2710 : Programming & Data Structures Lab (Jul-Nov 2018) Lab Session 5

Aug 27, 2018 **Sorting Algorithms: MergeSort and QuickSort** Mode : On Hacker Rank

Objectives:

- Understand and implement MergeSort, Quicksort
- Gain a perspective into these sorting algorithms and when and where they are preferred over other sorting algorithms.

Overview:

In this assignment, you are expected to write three classes *City, State and Country* and look to arrange states by their populations etc. You are free to define any new private variables and private functions.

Detailed Structure of Assignment Task:

This assignment has two components. Both components carry 30 marks each; 25 marks for program correctness (which will be judged by hacker rank test cases) + 5 marks for good coding practice.

Part I [InLab] : Designing the Classes "City" and "State": The structure of each class is defined below.

- The `City` class should have two private data members, `city_id` of int-type and `city_population` of long long int-type. It should have the following `public` functions
  - `City()`: An empty constructor
  - `City(int c_id, long long int c_p)`: A parametrised constructor to initialize the private fields `city_id` and `city_population`.
  - `int get_city_id()` : It returns the `city_id`.
  - `long long int get_city_population()` : It returns `city_population`.
- The `State` class should have three private data members `int state_id`, array `cities` (which is an array of `City` objects) for all cities under it and `int num_cities` (the size of the array). It should have the following `public` functions:
  - `State()`: An empty constructor
  - `State(int s_id, int n_c)`: A parametrised constructor which initialises `state_id`, `num_cities` and allocates the space for array `cities` to hold `num_cities City` objects.
  - A destructor to de-allocate the memory allocated to the *cities* array.
  - `int get_state_id()` : It returns the `state_id`.
  - `int get_num_cities()` : It returns the `num_cities`.

  - `read_state_details()` : It reads the `city_id` and `population` of each city from the user.
  - `City* sort_cities_of_state()`: It sorts the `cities` array in the decreasing order of their population using `Merge Sort` and returns the sorted cities array.

Part II [Take Home] :Merging arrays and Quick Sort: (You are advised to complete Merging arrays part first and then start the Quick Sort part) Modify "State" class and design a new class "Country".

- The `State` class should have one more private variable `total_state_population`. It should also have an additional public function
  - `void set_total_state_population()`: The function sums up the `population` of all the cities and updates `total_state_population`.
- The `Country` class should have two private data members array `states` (which is an array of `State` objects) and `int num_states` (the size of the array).
- It should have the following `public` functions:
  - `Country(int n_s)`: A parametrised constructor which initialises `num_states` and allocates the space for array `states` to hold `num_states State` objects.
  - A destructor to de-allocate the memory allocated to the *states* array.
  - `read_input()`: Reads input states, cities belonging to each state and their population.
  - `void sort_all_cities()`: It should sort all cities in the country, in the decreasing order of their population. It should do so only by merging the arrays returned by `sort_cities_of_state()` of each state. It should then print the `state_id` and `city_id` of the cities in sorted order.
  - `void sort_states()`: It first finds the `total_state_population` of all the states in the `Country` (You shall have a private function for this) and then sorts the states in the decreasing order of `total_state_population` using quick sort. In Quicksort, when partitioning, always choose last elements as the pivot and print the `state_id` of pivot in each step. Once sorting is finished, print the state_ids in sorted order.
- After defining the class, you should write a driver program that reads in the input as specified below and generates the corresponding output as specified. The driver performs the basic IO in C++.

<u>**Input-Output:**</u>

Part I
Input format
*state_id*1   *num_cities*
*city_id*1   *population*1
*city_id*2   *population*2
.
.

.

Output format

(In decreasing order of city populations)

City_idx

City_idy

.

.

.

Part II

Input format

num_states

*state_id*1 *num_cities*

*city_id*1 *population*1

*city_id*2 *population*2

.

.

.

*state_id*2 *num_cities*

*city_id*3 *population*3

*city_id*4 *population*4

.

.

.

Output format

(In decreasing order of city populations)

State_idx City_idx

State_idy City_idy

.

.

.

(In decreasing order of state populations)

*pivot*1 *pivot*2 *pivot*3....... *pivotN*

State_idx

State_idy

.

.

.