The assignment is to be turned in before Midnight (by 11:59pm) on February 1, 2018. You should turn in the solutions to the first question as a pdf file through the TEACH website. You should turn in your source code for the second question in a single C++ source code file with cpp extension through the TEACH website.

1: Query processing (3 points)

Consider the natural join of the relation R and S on attribute A. Neither relations have any indexes built on them. Assume that R and S have 80000 and 20000 blocks, respectively. The cost of a join is the number of its block I/Os accesses.

1. Assume that there are 300 buffer blocks available in the main memory. We would like to have the output of join sorted according to attribute A. What is the fastest join algorithm for computing the join of R and S? What is the cost of this algorithm?

   If we were just joining the two then a hash join would be fastest because we wouldn't be able to use the optimized sort-merge but since the output must be sorted it is faster to use sort-merge. The cost of this algorthm is: sorting + 2B(R) + 2B(S). This comes out to:

   $$(80000)(log_{300}(80000)) + (20000)(log_{300}(20000)) + 2(20000 + 80000) = 393074$$

2. Assume that there are 40 buffer blocks available in the main memory. What is the fastest join algorithm to compute the join of R and S? What is the cost of this algorithm?

   Since both of the relations R and S are more the M^2 ($40^2 = 1600$) we can't use sort merge or hash join. Instead we must use a Nested block join. The cost of this is B(S) + B(R)*B(S)/M which comes out to 40,020,000.

3. Assume that there are 200 buffer blocks available in the main memory. What is the fastest join algorithm to compute the join of R and S? What is the cost of this algorithm?

   Similar to 1.1, we can not use the optimized sort merge but we can use a hash because 200^2=40000 which is more than our lowest relation of 20000. Again, the cost is going to be 3B(R)+3B(S) which comes out to 3(20000 + 80000) = 300000.

2: Query processing (4 points)

Consider the following relations:

```
Dept (did (integer), dname (string), budget (double), managerid (integer))
Emp (eid (integer), ename (string), age (integer), salary (double))
```

Fields of types *integer*, *double*, and *string* occupy 4, 8, and 40 bytes, respectively. Each block can fit at most one tuple of an input relation. There are at most 22 blocks available to the join algorithm in the main memory. Implement the optimized sort-merge join algorithm for $Dept \bowtie_{Dept.managerid=Emp.eid} Emp$ in C++.

- Each input relation is stored in a separate CSV file, i.e., each tuple is in a separate line and fields of each record are separated by commas.

- The result of the join must be stored in a new CSV file. The files that store relations Dept and Emp are Dept.csv and Emp.csv, respectively.
- Your program must assume that the input files are in the current working directory, i.e., the one from which your program is running.
- The program must store the result in a new CSV file with the name join.csv in the current working directory.
- Your program must run on Linux. Each student has an account on *voltdb1.eecs.oregonstate.edu* server, which is a Linux machine. You may use this machine to test your program if you do not have access to any other Linux machine. You can use the following *bash* command to connect to *voltdb1*:

  ```
  > ssh your_onid_username@voltdb1.eecs.oregonstate.edu
  ```

  Then it asks for your ONID password and probably one another question. You can only access this server on campus.
- You can use following commands to compile and run C++ code:

  ```
  > g++ main.cpp -o main.out
  > main.out
  ```