

Cumulative Report
CS 4230-01
Austin Lee, Jacob Mulroy, Conner Hundt, D'Angelo Abell
10-01-2020

Project Part One

Successes

Installation and Setup of MongoDB: This was fairly straightforward as the VM was mostly setup from the last semester. The main problem here was just cleaning up old data, and setting up the new database for this project. Following are the primary commands we used:

| Command | Function |
|--------------------------------|-----------------------------------|
| Mongo | Init MongoDB service |
| use Students | Access existing database |
| db.dropDatabase() | Remove existing data from project |
| use Hospital | Create new project database |
| db.createCollection("info1") | Create collections for project |
| db.createCollection("info2") | |

Modifying the HTML Page: We decided to use a default HTML form to construct our main page. All coding was done within the client/main.(html/js/css) forms.

The screenshot shows a web browser window with the title 'Project Part 1' and the address bar set to 'localhost:3000'. The page content is titled 'HTML Page' and 'Enter Student Data'. It features a form with the following fields and controls:

- Case ID:
- Hospital Code:
- Type of Admission:
- Severity of Illness:
- Visitors With Patient:
- Age:
- Admission Deposit:
- Stay:
- Sequential Upload:
- Browse...: (No file selected.)
- Bulk Upload:

Inserting Records (individually): Using the constructed HTML page, we were able to successfully insert records sequentially into our MongoDB info1 collection.

Enter Student Data

Terminal

Hospital Code

Hospital Type Code

City Code Hospital

Hospital Region Code

Available Beds

Department

Ward Type

Ward Facility Code

Bed Grade

Patient ID

City Code Patient

```
leeaus@leeaus-VirtualBox: ~/FinalExam/FinalAssign
File Edit View Search Terminal Help
"Stay" : "a"
}
{
  "_id" : ObjectId("5f762a594d5dca0be7a2849b"),
  "case_id" : "e",
  "Hospital_code" : "e",
  "Hospital_type_code" : "e",
  "City_Code_Hospital" : "e",
  "Hospital_region_code" : "e",
  "Available" : "e",
  "Department" : "e",
  "Ward_Type" : "e",
  "Ward_Facility_Code" : "e",
  "BedGrade" : "e",
  "patientID" : "e",
  "City_Code_Patient" : "e",
  "TypeOfAdmission" : "e",
  "SeverityOfIllness" : "e",
  "Visitors" : "e",
  "age" : "e",
  "Admission_Deposit" : "e",
  "Stay" : "e"
}
```

Main Issues

Inserting Records (bulk): We were unable to implement a bulk upload through the HTML page. Our main problem was getting the script to recognize the button click events to actually upload and process the file.

- Resolution: Using the following MongoDB terminal command, we were able to manually import the entire .csv file into the collection.

```
mongoimport --type csv -d Hospital -c info2--headerline --drop
products.csv
```

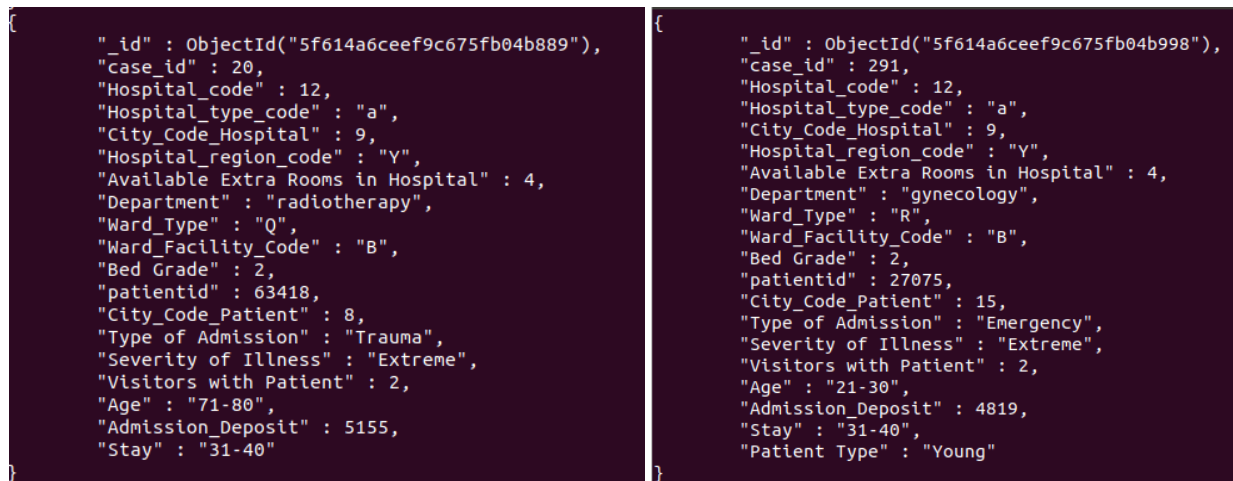
| Flag Explanation |
|---|
| –type: The input format to import: json, csv, or tsv. |
| –d: Specifies what database to use. |
| –c: Specifies what collection to use. |
| –headerline: Specifies that the first row in our csv file should be the field names. |
| –drop: Specifies that we want to drop the collection before importing documents. |
| last key is just the name of the csv file that you import |
| –type: The input format to import: json, csv, or tsv. We are using csv so that's what we specify. |

Modifying Variable-Length Records: For the first demo, we were able to append an extra category to all records, but not just one with a specific matching condition.

- Resolution: There was a typo in the original command which was causing the query to fail because there was no field header with that case-sensitive name. After amending the typo, we used the following command:

```
db.info2.update({"Age":{"$eq": "21-30"}},
{$set:{"Patient Type":"Young"}},false, true)
```

Which successfully modified the database as shown below:



```
{
  "_id" : ObjectId("5f614a6ceef9c675fb04b889"),
  "case_id" : 20,
  "Hospital_code" : 12,
  "Hospital_type_code" : "a",
  "City_Code_Hospital" : 9,
  "Hospital_region_code" : "Y",
  "Available Extra Rooms in Hospital" : 4,
  "Department" : "radiotherapy",
  "Ward_Type" : "Q",
  "Ward_Facility_Code" : "B",
  "Bed Grade" : 2,
  "patientid" : 63418,
  "City_Code_Patient" : 8,
  "Type of Admission" : "Trauma",
  "Severity of Illness" : "Extreme",
  "Visitors with Patient" : 2,
  "Age" : "71-80",
  "Admission_Deposit" : 5155,
  "Stay" : "31-40"
}
```

```
{
  "_id" : ObjectId("5f614a6ceef9c675fb04b998"),
  "Case_id" : 291,
  "Hospital_code" : 12,
  "Hospital_type_code" : "a",
  "City_Code_Hospital" : 9,
  "Hospital_region_code" : "Y",
  "Available Extra Rooms in Hospital" : 4,
  "Department" : "gynecology",
  "Ward_Type" : "R",
  "Ward_Facility_Code" : "B",
  "Bed Grade" : 2,
  "patientid" : 27075,
  "City_Code_Patient" : 15,
  "Type of Admission" : "Emergency",
  "Severity of Illness" : "Extreme",
  "Visitors with Patient" : 2,
  "Age" : "21-30",
  "Admission_Deposit" : 4819,
  "Stay" : "31-40",
  "Patient Type" : "Young"
}
```

Left: record of Age: 71-80 with no added field; **right:** record of Age: 21-30 with added field header/value

Project Part Two

Successes

Quicksort: A python-based implementation of quicksort was able to successfully sort the DateSet.csv file by age. On average, this run took ~4200ms to complete sorting of the entire data set.

MongoDB sort: This was implemented in one of two ways.

- *Initial method* - Using the following command:

```
db.info2.find({}).sort({"Age":1}).explain("executionStats")
```

We were able to sort the collection by age. However, this was only a temporary sort method, and while searches were functional, the sort never persisted to the collection.

- *Second method* – Using the following command:

```
db.info2.find().sort({"Age":1}).forEach(function(
    e){db.info2.insert(e);})
```

Allowed us to sort the collection and persist the changes to the collection. Interestingly, by modifying the bolded portion of the above code (the destination collection name) we could also sort to an entirely new collection.

Sorting to the same database took ~282,040ms to complete, while sorting to a new (empty) collection took ~69,920ms to complete. We verified these times by using the appendix **.explain("executionStats")** to our sort call, which displayed the processing time in millis.

```

    },
    "executionStats" : {
      "executionSuccess" : true,
      "nReturned" : 318438,
      "executionTimeMillis" : 2442,
      "totalKeysExamined" : 0,
      "totalDocsExamined" : 318438,
      "executionStages" : {
        "stage" : "SORT",
        "nReturned" : 318438,
        "executionTimeMillisEstimate" : 2311,
        "works" : 636880,
        "advanced" : 318438,
        "needTime" : 318441,

```

Sample display from using **.explain()**

Bubblesort: Our initial C# implementation had some major issues upon submission. Fortunately, as of demo, we were able to implement it in both python and Ruby. However, there were some interesting observations we noted.

- *Python* – The program was able to sort smaller versions of the dataset, at an almost exponential level of growth re: processing time. For example:

| Number Records | Time (sec) |
|----------------|------------|
| 100 | 0.023 |
| 1000 | 0.226 |
| 5000 | 2.743 |
| 10000 | 10.251 |
| 50000 | 339.715 |
| 100000 | ??? |

When processing 5000 and fewer records, the program completed with expected times. However, as the number of records increased, the processing time increased in massive

leaps, with 100,000 records taking in excess of 1.25 hours to complete (we actually stopped the program mid-run because of time constraints.)

This program was run on: AMD Ryzen 5 3600 6-Core Processor, 16.0GB RAM, Win10 x64, using the Python3 installation.

- *Ruby* - This program did perform similarly to python with regards to performance times. However, we let this program run overnight just to see if it would finish sorting. Approximately 7 hours later, it had managed to successfully complete sorting of the entire data set.

This program was run on: Intel i5 6600k OC 4.6 GHz, 16gb 2400MHz ram, gtx 1070, Intel SSDPEKKW256G7 ssd, 4tbh 4HDD Raid 0 hdd array.

Main Issues

The major issue with this portion was with MongoDB. While running the Initial Method of sorting, the following error message appeared:

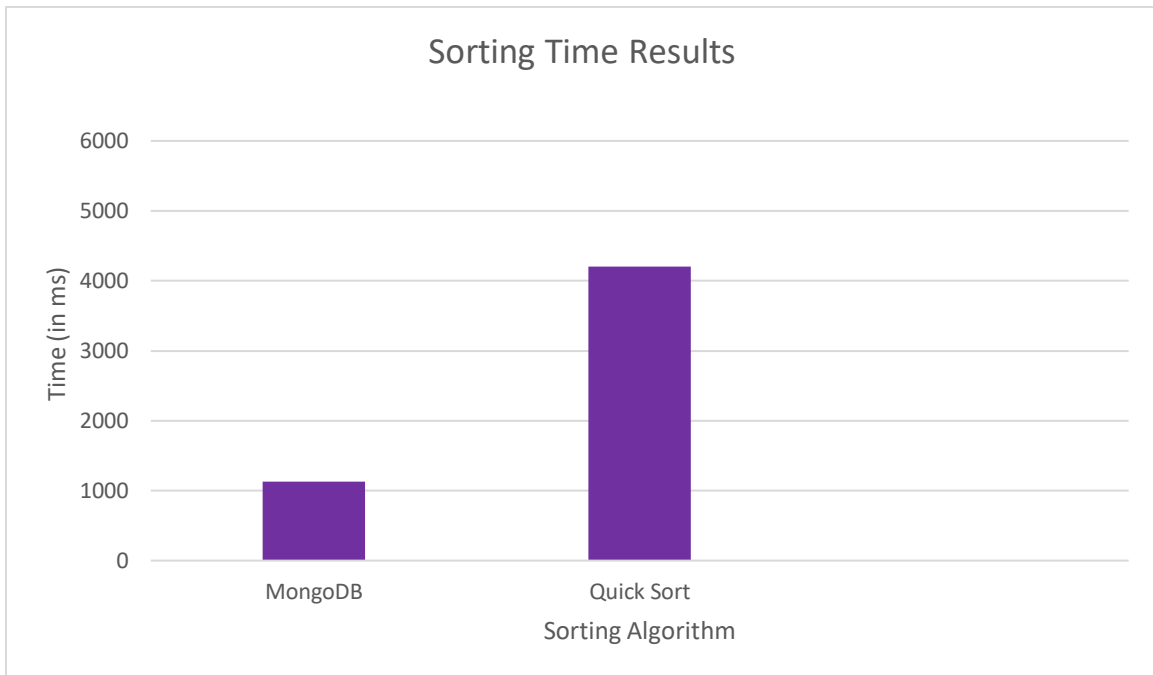
```
"executionSuccess" : false,  
  "errorMessage" : "Exec error resulting in state FAILURE :: caused by :: errmsg:  
  \Sort operation used more than the maximum 33554432 bytes of RAM. Add an index, or specify a  
  smaller limit.\",
```

Informing us that the sort operation had exceeded its memory. To solve this, we used the following command:

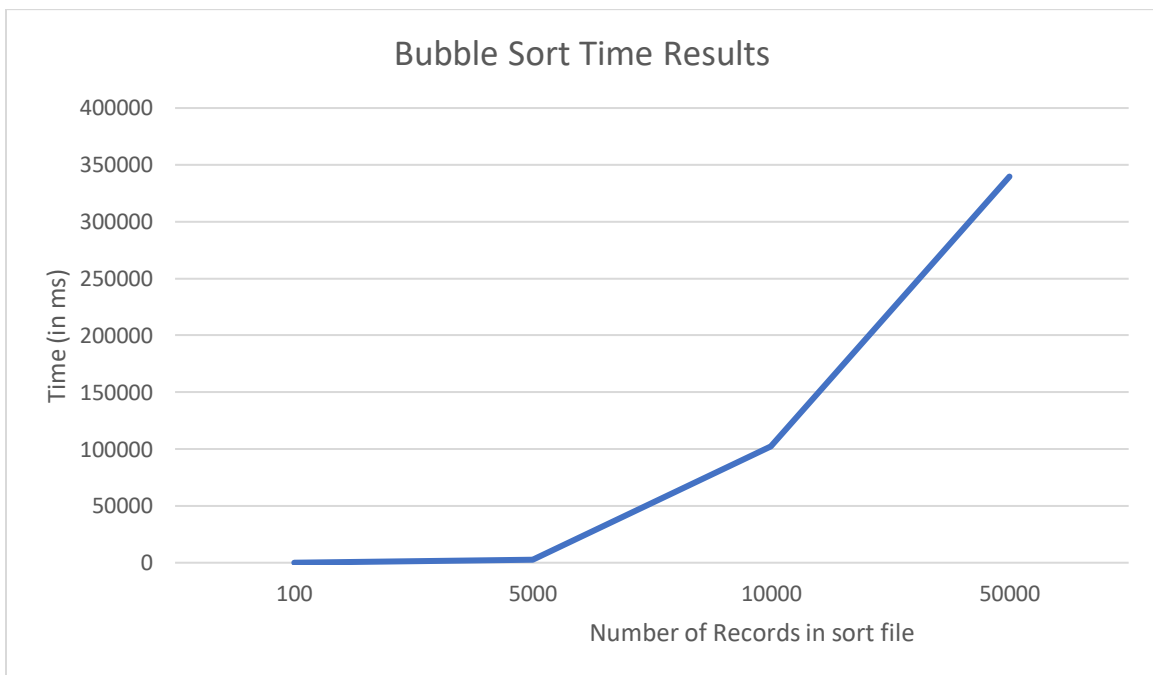
```
db.adminCommand({setParameter: 1,  
  internalQueryExecMaxBlockingSortBytes: 335544320})
```

Which allowed us to override the error and allocate more virtual space for the sorting process to complete. We also attempted a secondary method using the aggregate function of MongoDB, and while the terminal said that it was successful, the changes were not reflected in the collection. Therefore, we decided to opt for the prior command sequence instead.

Graphs



Bubble sort excluded from group due to indeterminable processing times



Average bubble sort processing times on data sets of up to 50000 records

Project Part Three

Successes

Binary Search: A python-based implementation of Binary search was used to scan the file for the value. On average, this took ~0.234ms to complete successfully, and was by far the fastest search function used.

Sequential Search: A Python-based implementation of Sequential search was used to scan the file for the value. If the data file was unsorted, this took an average of ~4ms to complete successfully. When used on a sorted file, it took longer: an average of ~40ms.

MongoDB find(): This was implemented using the following command:

```
db.info2.findOne({"Age":"21-30"})
```

Using the .explain() command, we were able to determine that this took an average of ~515ms to complete, making it the slowest search function used. We verified the results using a sorted version of the data set:

```
> db.info3.findOne({"Age":"21-30"})
{
  "_id" : ObjectId("5f614a6ceef9c675fb04b8cd"),
  "case_id" : 88,
  "Hospital_code" : 11,
  "Hospital_type_code" : "b",
  "City_Code_Hospital" : 2,
  "Hospital_region_code" : "Y",
  "Available Extra Rooms in Hospital" : 3,
  "Department" : "anesthesia",
  "Ward_Type" : "Q",
  "Ward_Facility_Code" : "D",
  "Bed Grade" : 3,
  "patientid" : 88451,
  "City_Code_Patient" : 8,
  "Type of Admission" : "Trauma",
  "Severity of Illness" : "Moderate",
  "Visitors with Patient" : 4,
  "Age" : "21-30",
  "Admission_Deposit" : 2816,
  "Stay" : "0-10",
  "Patient Type" : "Young"
}
```

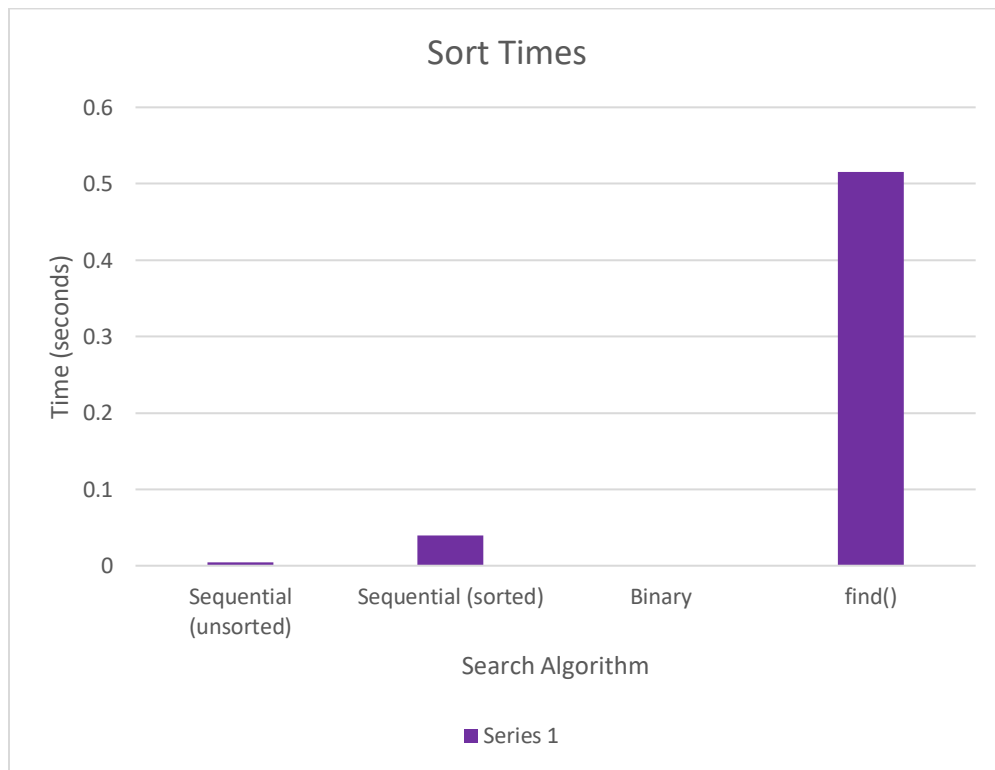
| | A | B | C | D | E | F | G | H | I | J |
|-----|---------|---------------|------|----------|----------|-----------|--------------|-----------|--------------------|-----------|
| 1 | case_id | Hospital_code | Hosp | City_Cod | Hospital | Available | Department | Ward_Type | Ward_Facility_Code | Bed Grade |
| 89 | 88 | 11 b | | 2 Y | | 3 | anesthesia | Q | D | 3 |
| 90 | 89 | 19 a | | 7 Y | | 4 | gynecology | Q | C | 2 |
| 91 | 90 | 19 a | | 7 Y | | 4 | gynecology | Q | C | 2 |
| 92 | 91 | 9 d | | 5 Z | | 3 | radiotherapy | Q | F | 3 |
| 93 | 92 | 19 a | | 7 Y | | 6 | gynecology | Q | C | 1 |
| 124 | 123 | 6 a | | 6 X | | 5 | gynecology | Q | F | 3 |
| 125 | 124 | 23 a | | 6 X | | 5 | gynecology | Q | F | 4 |
| 126 | 125 | 26 b | | 2 Y | | 4 | gynecology | Q | D | 2 |
| 127 | 126 | 11 b | | 2 Y | | 3 | gynecology | Q | D | 2 |
| 128 | 127 | 14 a | | 1 X | | 3 | gynecology | R | E | 3 |

Main Issues

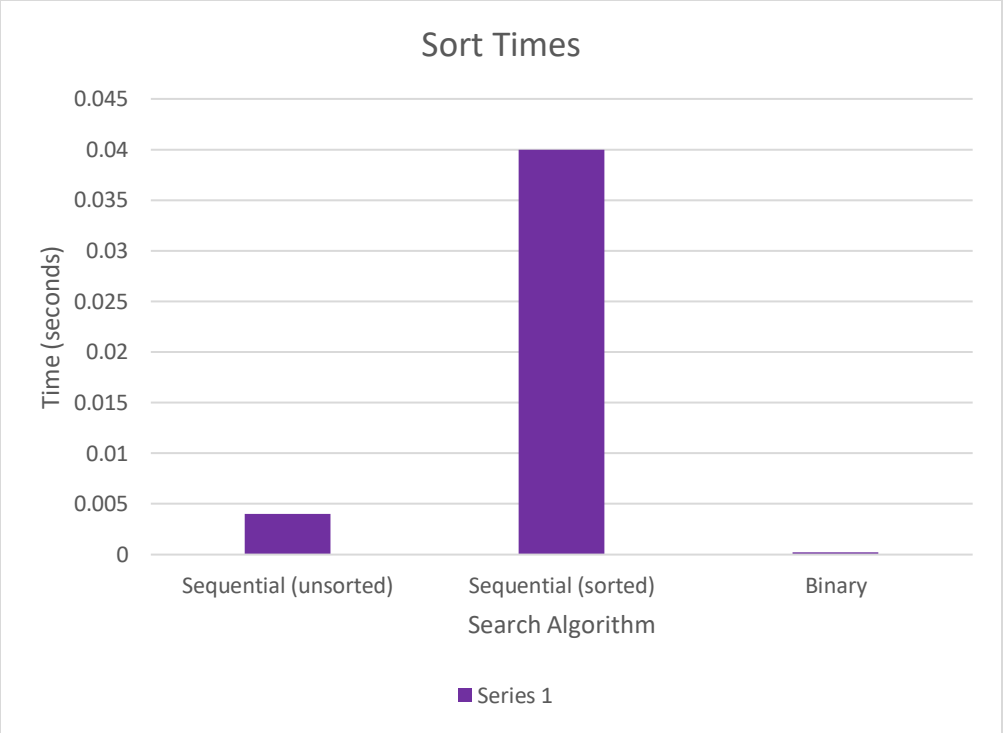
N/A: We actually did not encounter any significant issues with this project part related to either the search algorithms or MongoDB.

Graphs

| Method | Time (sec) |
|----------------------|------------|
| Sequential(unsorted) | 0.004 |
| Sequential (sorted) | 0.04 |
| Binary | 0.000234 |
| find() | 0.515 |



All sort times together



Sort times excluding find() for clarity around smaller values

Miscellaneous

Ruby Source Code

@author Jacob Mulroy

Creating the Record Class:

```
class Record
  attr_accessor :case_id
  attr_accessor :Hospital_code
  attr_accessor :Hospital_type_code
  attr_accessor :City_Code_Hospital
  attr_accessor :Hospital_region_code
  attr_accessor :Available_Extra_Rooms_in_Hospital
  attr_accessor :Department
  attr_accessor :Ward_Type
  attr_accessor :Ward_Facility_Code
  attr_accessor :Bed_Grade
  attr_accessor :patient_Id
  attr_accessor :City_Code_Patient
  attr_accessor :Type_of_Admission
  attr_accessor :Severity_of_Illness
  attr_accessor :Visitors_with_Patient
  attr_accessor :Age
  attr_accessor :Admission_Deposit
  attr_accessor :Stay

  def initialize (a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r)
    @case_id = a
    @Hospital_code = b
    @Hospital_type_code = c
    @City_Code_Hospital = d
    @Hospital_region_code = e
    @Available_Extra_Rooms_in_Hospital = f
    @Department = g
    @Ward_Type = h
    @Ward_Facility_Code = i
    @Bed_Grade = j
    @patient_Id = k
    @City_Code_Patient = l
    @Type_of_Admission = m
    @Severity_of_Illness = n
    @Visitors_with_Patient = o
    @Age = p
    @Admission_Deposit = q
    @Stay = r
  end
end
```

Creating the Sort Function:

```
require 'csv'
require_relative 'Record'
@done = false
@rows = 318438
@list = Array.new
columns = 18
table = CSV.parse(File.read("DataSet.csv", headers: true))
puts table[0][0]
puts table[0][15]
puts table[1][0]
puts table[1][15]
for i in 1..(318438 - 1) do
  @list.push(Record.new(table[i][0], table[i][1], table[i][2], table[i][3],
table[i][4], table[i][5], table[i][6], table[i][7], table[i][8], table[i][9],
table[i][10], table[i][11], table[i][12], table[i][13], table[i][14],
table[i][15], table[i][16], table[i][17]))
end
  array_length = @list.size
  return @list if array_length <= 1

loop do
  # we need to create a variable that will be checked so that we don't run
  into an infinite loop scenario.
  swapped = false

  # subtract one because Ruby arrays are zero-index based
  (array_length-1).times do |i|
    if @list[i].age > @list[i+1].age
      @list[i], @list[i+1] = @list[i+1], @list[i]
      swapped = true
    end
  end
  break if not swapped
end
for i in 0..(@rows - 1) do
  puts @list[i].age
end
```