1) What is your definition of an object?

An object is an instantiated entity containing data.

2) What strategies should be deployed in terms of accepting input (i.e. the large number of objects.)?

Ruby only has two main container classes, Arrays and Hashes. Because hashes use any object as its index, it would not make sense to sort a hash. Thus, our contract enforces that the input is an Array of any non-zero size of objects, and that the objects implement the Comparable mixin.

3) Is your sort generic? What sorting criterion does your system use? Is this criterion flexible; i.e. changeable by the user at the point of execution? What limitations have you placed upon legal sorting criteria? To make it reusable to other people, how should we include it into the Ruby Class hierarchy?

The sort is generic. The system expects that objects passed in implement the Comparable mixin to establish relations between objects, and uses a passed in block to determine how to use those operations to sort the array. This criterion is semi-flexible (they can change the implementation of Comparable mixin), but they can't change the sorting at the execution point. The limitations we have placed on legal sorting criteria all follow the limitations provided by the Comparison mixin. To make it reusable to other people, the module should be included under the Array class as a mixin.

4) In reality we have a uniprocessor system, describe "equationally" what is happening to your solution as you increase the concurrency (e.g. produce a regression model (remember regression from ECE 231) of processing time against the number of threads. Your solution can be modeled as a program which: (1) has a component which produces threads; and (2) a set of threads which undertake the same (small) task.
This is in essence the basis of stress testing (discussed in CMPE 320?)

As our solution increases in concurrency, the processing time decreases logarithmically. This is because our solution spawns threads approximately 2 to the power of the size of the input, followed by at most one thread per ¾ of an input during merging.
That is to say:

$$Processing\ Time\ (numThreads) = \frac{1}{2^n} + \frac{4}{3n}$$

Thus:

$$T(n) = \Theta(n/lg^2 n)$$

5) Concurrent systems tend to crash frequently – what approach to exception- handling have you devised? Consider the content of the library at: http://c2.com/cgi/wiki?ExceptionPatterns; which are applicable to this problem? Is Module Errno useful in this problem? What components of the Ruby exception hierarchy are applicable to this problem? Discuss in detail your strategy for exception-handling.

Our strategy for exception handling is to use the AbortRetryIgnore pattern, aborting computation once an exception occurs, as we expect all exceptions to be abnormal cases. Applicable exception handling patterns that apply are the AbortRetryIgnore and the CatchWhatYouCanHandle patterns.

Module Errno is useful for this problem, particularly:

ENOMEM: Out of memory, no room to spawn a new thread.
EAGAIN: Try again, unable to create a new thread at that time so try again

SystemError, IndexError, ThreadError, TypeError, StandardError, NoMethodError and ArgumentError are the components of the Ruby Exception hierarchy that are applicable to this problem.

6) What differences exist between thread-based and process-based solutions? How has this impacted the design of your solution?

Differences:

1. A process can have multiple threads, a thread can have only one process.
2. Each process requires its own address space, threads can share an address space.
3. Difficult for one process to be corrupted by another process as compared to threads.
4. Easier for one thread to talk to another thread as compared to processes.

Due to the above differences, a thread-based solution is more appropriate for our system. Our design is concerned with sorting data. For this purposes speed and communication are major priorities. Speed for its obvious impact on total execution time and communication in order to allow the threads to coordinate their sorted data together. Multiple threads allow our design to easily sort the same set of data in the same memory space.

7) Do you have any race-condition or task synchronization concerns about your solution? How do we tidy-up a multi-threaded program, if stopped mid- execution?

There are task synchronization concerns, in that parent threads must wait for each child thread to sort their data and handle the result. A multi-threaded program needs to kill all of it child threads if it is stopped mid-execution.

8) As discussed in CMPE 300: What is configuration management? What is version

control? Are you using either concept? If “yes”, describe your process and any tool support what you utilize – illustrate your process with regard to Assignments 1 and 2; if “no”, justify your decision.

Configuration management is tracking and controlling changes to a system. Version control (a feature of configuration management) is the managing of these changes and collecting them into organized sets of revisions. We are using both concepts. For all assignments done, code and supporting documentation are being managed through git with hosting being done on github.

9) Briefly Explain:
a. What is refactoring (as discussed in CMPE 300)?

Refactoring is the process of altering code to improve its readability and clarity while maintaining its intended functionality.

b. Are you using refactoring in your development process? Justify your answer?

Yes. We have used, and will continue to use refactoring in our development to improve the quality of our code.

c. If “yes”, give examples, minimum of 2, of the refactoring “patterns” that you used in Assignment 1

Encapsulate Field - our AddNewValue method in sparse_matrix.rb was made to encapsulate setting the value of a element in the sparse matrix.

Extract Class - our matrix_element.rb class was extracted from the sparse_matrix.rb file

Rename Method - used at several times during development, including in the sparse_matrix.rb file, where the method previously called ‘transform’ was changed to ‘to_matrix’

d. If “no”, give examples of where your solution to Assignment 1 would be improved by applying refactoring patterns. Supply a minimum of two different (i.e. different refactoring patterns) as examples.

N/A