Mutlithreaded Sorting Algorithms

Mary Markart  
San Jose State University  
CS 149San Jose, CA

*Abstract*—This report provides the necessary information to understand and develop an algorithm that implements MergeSort and QuickSort that utilizes multithreaded using Java’s ForkJoinPool and RecursiveAction.

# Introduction

The use of sorting algorithms proves beneficial when sorting large amounts of data. Two of the most efficient sorting algorithms are MergeSort and QuickSort due to their time complexity of . Although, QuickSort’s worst case time complexity can increase exponentially to . Both QuickSort and MergeSort addressing sorting with the divide and conquer approach, meaning they divide up the array to be sorted and perform their sorting algorithm on the smaller subsets of the array. Due to this approach, it makes them ideal for multithreading.

Multithreading is when a process is divided into smaller tasks that can be executed in concurrently or in parallel. By implementing multithreading with these sorting algorithms, the work is broken up and executed using threads in Java. Java.util.concurrent.\* will be utilized with ForkJoinTask and RecursiveAction, which will be further explained in the Software Architecture section.

In this report, an approach to using multithreading to implement QuickSort and MergeSort will be explained. The report contains a Problem Statement which is the specifications for the multithreaded programs, the “software architecture” which described the data structures and framework implemented in the program, tests and results, and a conclusion.

This files submitted for this project are

1. QuicksortTask.java: The class QuicksortTask uses the quicksorting algorithm to sort the values of a given ArrayList or List with Type <E>
2. MergesortTask.java: The class MergesortTask extends RecursiveTask and uses the mergesort algorithm to sort the values of a given ArrayList or List with Type <E>
3. MultithreadTask.java: The class MultithreadTask contains the program to run the mergesort and quicksort tasks

# Problem Statement

## The program must take a list of integers and split it into two equal sublists. The sublists are then sorted by a given sorting algorithm using a separate thread for each sublist. These will be called the sorting threads. The sorting threads will be merged together by a merge thread.

In this project, ArrayLists and Lists will be the data structure utilized. The program is implemented using generics so that any datatype could be input into the ArrayLists or Lists. The threads will be constructed using ForkJoinPool, fork(), join() and invoke().

## The above specifications must be implemented in two different programs. MergeSort and QuickSort.

For this project, two separate classes have been created. QuicksortTask and MergesortTask. Each program contains the code for the specified sorting algorithm to be executed using multithreading. These classes extend RecursiveTask and RecursiveAction.

## Use a separate sorting algorithm if the given list is under a specified threshold

The threshold value in this project is 100. Any list shorted than 100 value will be sorted using InsertionSort because of it’s best case time complexity of .

## The project is required to use Java’s comparable interface

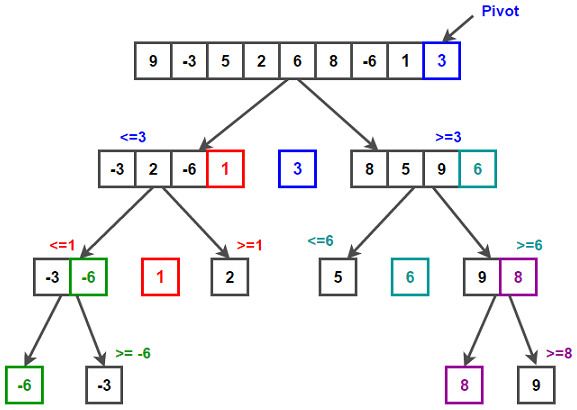
The QuicksortTask and MergesortTask classes have been written so that the merge() and partition() methods use compareTo() to sort the sublists.

# Software Architecture

The following section will outline QuickSort, MergeSort, ForkJoinPool, RecursiveTask, and RecursiveAction.

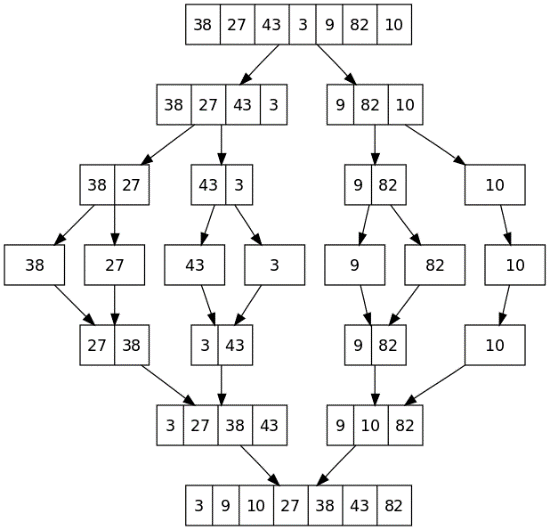
## QuickSort

QuickSort is an algorithm that breaks up a List based on a pivot value. The algorithm first calls on a method called partion() that determines a pivot value, and divides the list into two parts, those values that are more than or equal to the pivot, and the values that are less than the pivot. Quicksort is recursively called



## MergeSort

MergeSort is an algorithm that divides a list in half and recursively sorts the list before merging the two halves back together sorted.



## ForkJoinPool

In this project, multithreading will be implemented using ForkJoinPool framework. This framework allows for effective and efficient parallel execution using pools of threads. It works by utilizing a divide and conquer approach, which is why it works well with the divide and conquer sorting algorithms. To “divide” the method fork() is called. This will recursively divide the task into smaller subtasks. The “conquer” aspect is accomplished by the join() method, which is where all the work done in the subtasks are joined or the program waits until every subtask has executed.

ForkJoinPool is an ExecutorService for running ForkJoinTasks and serves as an entry point for non-ForkJoinTask clients. ForkJoinPool implements work-stealing, i.e. all the threads in the pool attempt to find and execute tasks submitted to the pool or created by other active tasks.

The three main task execution methods for ForkJoinPool. The ones that can be called from non-fork/join clients are execute(ForkJoinTask) which arranges the asynchronous execution of a task, invoke(ForkJoinTask) which performs the given task and returns the results when it completes, and submit(ForkJoinTask) which arranges the execution and obtains Future. Alternatively, when called within fork/join clients these methods are ForkJoinTask.fork(), ForkJoinTask.invoke(), and ForkJoinTask.fork() respectively.

The call construct ForkJoinPool() creates a ForkJoinPool with parallelism equal to Runtime.availableProcessors() using the default thread library.

## RecursiveTask and RecursiveAction

ForkJoinPool will be utilized by using RecursiveTask and RecursiveAction.

Diagram

Description automatically generated

RecursiveTask is a recursive result bearing task.

RecursiveAction is a resultless recursive task.

Both RecursiveTask and RecursiveAction must implement the method compute(). This is where the work will be done recursively when the pool is invoked, i.e. when the ForkJoinPool object’s initial task is initialized by ForkJoinPool.invoke(). The difference between RecursiveTask and RecursiveAction is that a result is being returned with RecursiveTask. RecursiveTask is used in the MergesortTask program because compute() returns the sorted array after merge() is called. RecursiveAction is used in QuicksortTask because the array is sorted in place, and therefore did not need to be returned.

## Project Structure

1. QuicksortTask UML Diagram

Diagram

Description automatically generated

1. MergesortTask UML Diagram

Diagram

Description automatically generated

# Implementation

## QuicksortTask

1. Class Definition:

public class QuicksortTask<E extends Comparable<E>> extends RecursiveAction

This defines the Java class QuicksortTask that is able to compare values of a generic type and implements the ForkJoinPool framework by extending RecursiveAction.

1. Initial Constructor

public QuicksortTask(ArrayList<E> a, int low, int high){  
 this.a = a;  
 this.low = low;  
 this.high = high;  
}

## MergesortTask

THis

## MultithreadTask

This

# Test & Results

## Identify the Headings

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

Component heads identify the different components of your paper and are not topically subordinate to each other. Examples include Acknowledgments and References and, for these, the correct style to use is “Heading 5”. Use “figure caption” for your Figure captions, and “table head” for your table title. Run-in heads, such as “Abstract”, will require you to apply a style (in this case, italic) in addition to the style provided by the drop down menu to differentiate the head from the text.

Text heads organize the topics on a relational, hierarchical basis. For example, the paper title is the primary text head because all subsequent material relates and elaborates on this one topic. If there are two or more sub-topics, the next level head (uppercase Roman numerals) should be used and, conversely, if there are not at least two sub-topics, then no subheads should be introduced. Styles named “Heading 1”, “Heading 2”, “Heading 3”, and “Heading 4” are prescribed.

## Figures and Tables

#### Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation “Fig. 1”, even at the beginning of a sentence.

1. Table Type Styles

| Table Head | Table Column Head | | |
| --- | --- | --- | --- |
| Table column subhead | Subhead | Subhead |
| copy | More table copya |  |  |

1. Sample of a Table footnote. (*Table footnote*)
2. Example of a figure caption. (*figure caption*)

# Conclusion

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity “Magnetization”, or “Magnetization, M”, not just “M”. If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write

# Appendix

“Magnetization (A/m)” or “Magnetization {A[m(1)]}”, not just “A/m”. Do not label axes with a ratio of quantities and units. For example, write “Temperature (K)”, not “Temperature/K”.

##### Acknowledgment *(Heading 5)*

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression “one of us (R. B. G.) thanks ...”. Instead, try “R. B. G. thanks...”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

##### References

The template will number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use “Ref. [3]” or “reference [3]” except at the beginning of a sentence: “Reference [3] was the first ...”

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the abstract or reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors’ names; do not use “et al.”. Papers that have not been published, even if they have been submitted for publication, should be cited as “unpublished” [4]. Papers that have been accepted for publication should be cited as “in press” [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

1. **The text book**. *(references)*
2. **THE slides**.
3. **146 Slides**.
4. K. Elissa, “Title of paper if known,” unpublished.
5. Java Oracle 8 for ArrayList. List, ForkJoinPool, Recursive task.
6. Quicksort Image https://www.techiedelight.com/quicksort/
7. M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

**IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove template text from your paper may result in your paper not being published.**