Advanced Enterprise Java
Project-3

Submission by Swapnil Patel (1966690)
https://github.com/swap1210/ Adv-java-assignment3/

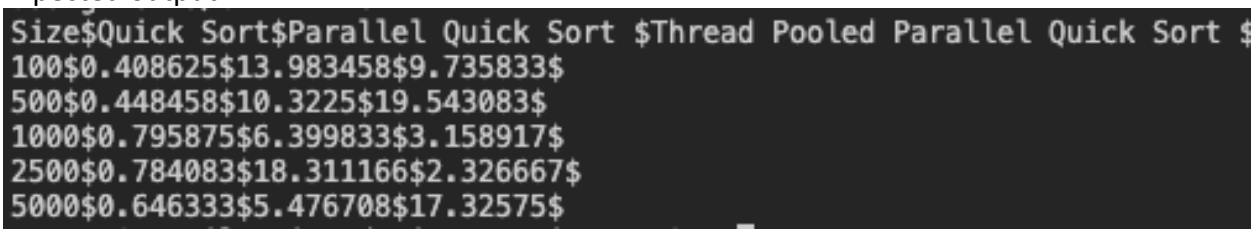
Table of Contents

How to Execute:

To execute just run the Driver.java file inside src folder.

> javac Driver.java && java Driver

Expected output:



The outcome can be pasted in excel column 1 and the provided excel will automatically separate $ delimited values and plot a chart against it.

It shows for various array sizes how long each sort algorithm ran.

Driver Class:

It performs benchmark tests on three different implementations of the Quick Sort algorithm. Here's an explanation of what the code does:

- The program defines a class called "Driver" with a static variable "testResults", which is a list of lists of strings. This variable will store the results of the benchmark tests.
- The main method of the program initializes the "testResults" list, creates a header row for the results, adds the header row to the list, and then runs three test iterations for different array sizes.
- The "fillUniqueArray" method is called by the main method to generate an array of unique integers with a specified size. This method uses a HashSet to ensure that each number in the array is unique.
- The "performThreeTest" method is called by the main method to run three tests for a given array of data. This method creates three copies of the input array, and then applies three different implementations of the Quick Sort algorithm to each of the copies. The method then measures the execution time of each of the three sorts and stores the results in a new list of strings. This list is then added to the "testResults" list.
- Finally, the main method iterates through the "testResults" list and prints out the contents of each row, separated by "$" characters.

QuickSort class

This class has three methods:

swap - a static utility method that swaps two elements in an integer array. It takes three arguments: the array and the indices of the two elements to be swapped.

partition - a static method that takes an integer array, a low index, and a high index. It selects the last element in the array as the pivot and rearranges the elements in the array such that all elements smaller than the pivot is on its left, and all elements greater than the pivot is on its right. It then returns the index of the pivot.

perform - a static method that takes an integer array and sorts it using the QuickSort algorithm. It is a wrapper method that calls a recursive helper method perform(arr, low, high) with low=0 and high=arr.length-1. The helper method recursively partitions the array and sorts the two resulting sub-arrays.

Overall, the class implements the QuickSort algorithm to sort an array of integers in ascending order.

ParallelQuickSort class

The class ParallelQuickSort extends the RecursiveTask class, which is a class in the Fork/Join framework that represents a task that can be split into smaller sub-tasks and executed concurrently. The perform method takes an array arr as input and starts the first thread in the Fork/Join pool to sort the array.

The partition method finds a random pivot and partitions the array around the pivot. It returns the index of the pivot element.

The constructor of the ParallelQuickSort class takes the start and end indices of the sub-array to be sorted, and the array itself. The compute method is the main method that performs the sorting. It first checks if the start index is less than the end index, which is the base case for the recursion. It then calls the partition method to find the pivot index, and creates two new ParallelQuickSort objects for the left and right sub-arrays. The left sub-problem is forked and the right sub-problem is computed sequentially. Then, the left sub-problem is joined to wait until it is complete. Finally, null is returned.

The perform method creates a ForkJoinPool and starts the first thread in the pool to sort the entire array. The ParallelQuickSort object for the entire array is created and passed to the invoke method of the ForkJoinPool.

This implementation of QuickSort is parallelized using the Fork/Join framework, which allows the sorting to be performed concurrently on multiple cores. The algorithm is divided into smaller sub-tasks, which are executed concurrently using the fork and join methods.

ParallelQuickSortThreadPool class

This class uses an Executor, specifically a ThreadPoolExecutor, to manage the threads. The Executor provides a pool of threads that can be used to execute Runnables. The implementation creates a fixed thread pool with a size equal to the number of available processors.

The main method, perform, takes an input array and sorts it in place using multiple threads. It creates a QuicksortRunnable with the input array, starting index, ending index, and an AtomicInteger count initialized to 1. The count is used to keep track of the number of threads currently executing. The runnable is added to the thread pool using the Executor.execute() method.

The QuicksortRunnable class is a nested class that implements the Runnable interface. It contains the logic for sorting a subarray of the input array. The run() method is the entry point for the thread. It calls the quicksort() method to sort the subarray and then checks if all threads have finished executing. If all threads have finished executing, it notifies the count object to signal that sorting is complete.

The quicksort() method contains the actual sorting logic. It first partitions the subarray using the partition() method and then sorts the left and right subarrays recursively if the number of threads currently executing is less than a certain threshold. If the number of threads currently executing is greater than or equal to the threshold, it creates new QuicksortRunnable objects and submits them to the thread pool.

The partition() method is a simple implementation of the partitioning step in the quicksort algorithm. It selects the pivot value as the rightmost element in the subarray and partitions the subarray into two parts: elements less than the pivot value and elements greater than or equal to the pivot value.

The swap() method is a helper method for swapping two elements in the input array.

Overall, the implementation provides a parallel quicksort algorithm that can take advantage of multiple processors to sort an input array faster than a single-threaded quicksort algorithm.

Analysis

How to interpret the output:

```
swapn@Swapnils-Air Adv-java-assignmen
gnment3 ; /usr/bin/env /Library/Java/
/java -XX:+ShowCodeDetailsInExceptionM
assignment3/bin Driver
Size$Quick Sort$Parallel Quick Sort $
100$0.78025$15.369$11.0775$
500$0.244333$3.106333$3.07025$
1000$3.10425$9.599625$13.839792$
2500$0.287834$15.432625$8.491167$
5000$1.359709$4.585416$15.183292$
```

Size	Quick Sort	Parallel Quick Sort	Thread Pooled Parallel Quick Sort
100	0.78025	15.369	11.0775
500	0.244333	3.106333	3.07025
1000	3.10425	9.599625	13.839792
2500	0.287834	15.432625	8.491167
5000	1.359709	4.585416	15.183292