Program Structures and Algorithms Spring 2024

Name : Shaofan Wei

NUID: 002815198

GITHUB LINK:https://github.com/Arthurccone123/INFO6205

Task:

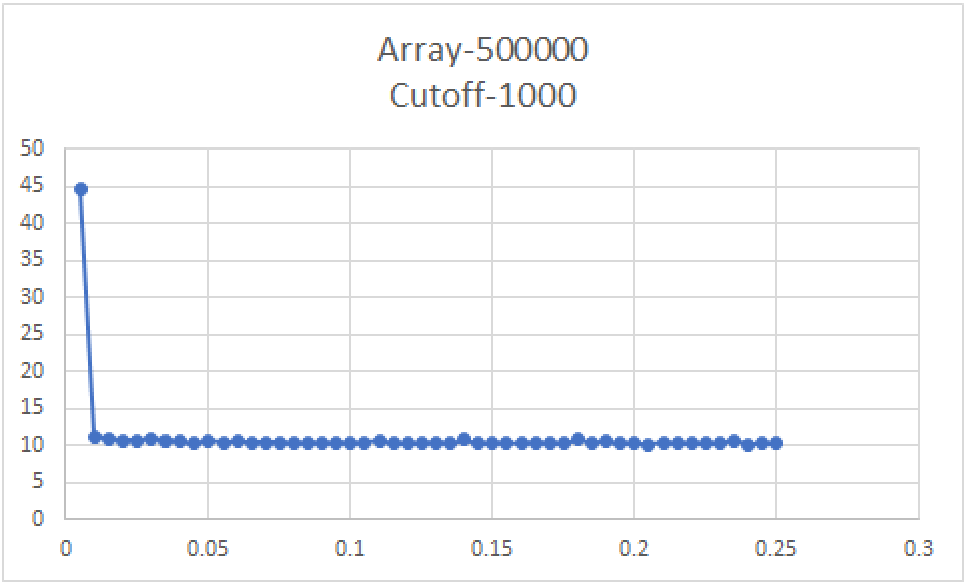
This task requires us to implement a parallel sorting algorithm and explore the effect of different cutoff values on the performance of the sorting algorithm under different arrays.

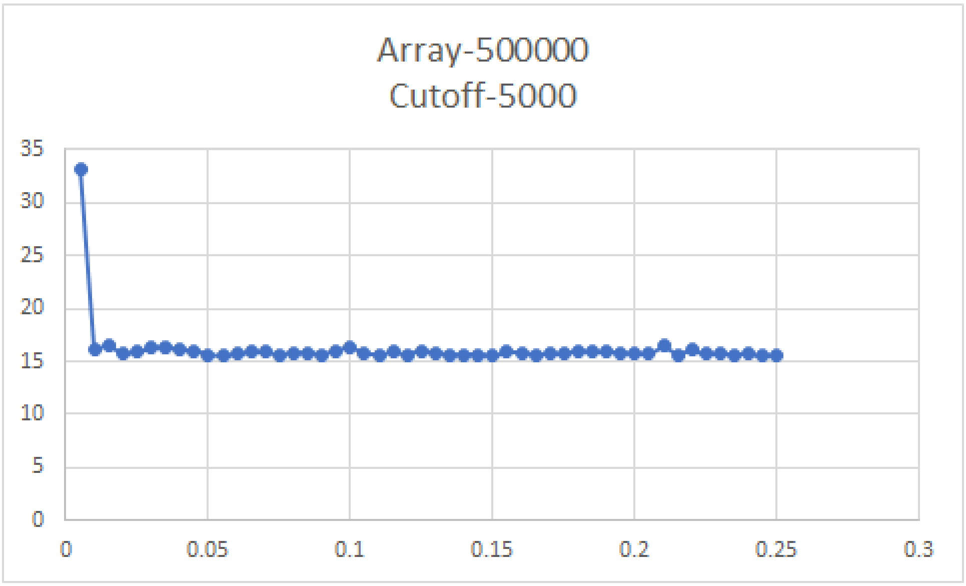
Relationship Conclusion:

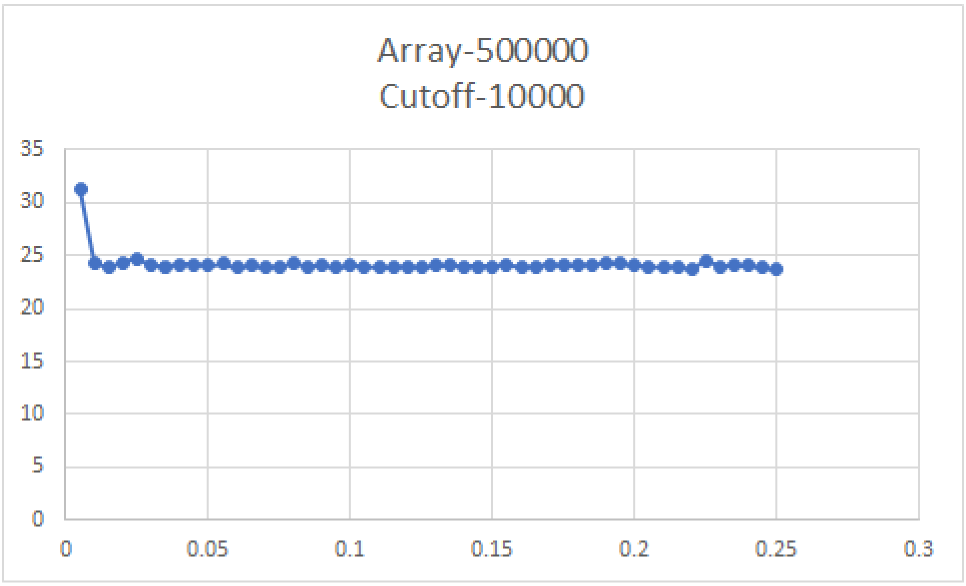
As the array size increases, the cutoff value may need to be increased accordingly. For smaller arrays, lower cutoff values are more appropriate. For larger arrays, higher cutoff values are more appropriate.

Evidence to support that conclusion:

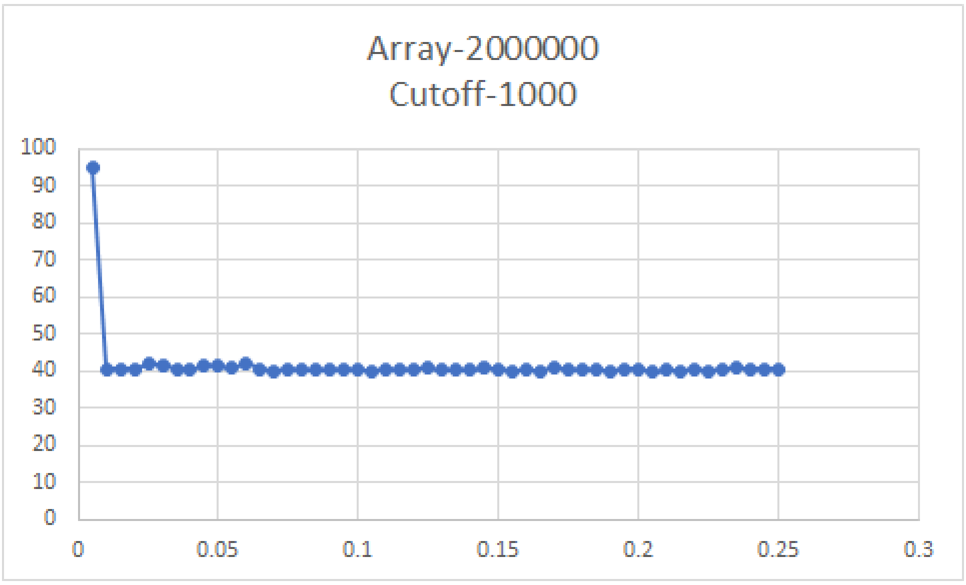
When the array is only 500000, we can find that when cutoff is 1000, the algorithm performs best, and the running time is about 10ms; when cutoff is 5000, the running time rises to 15ms; when cutoff is 10000, the running time comes to about 24ms.

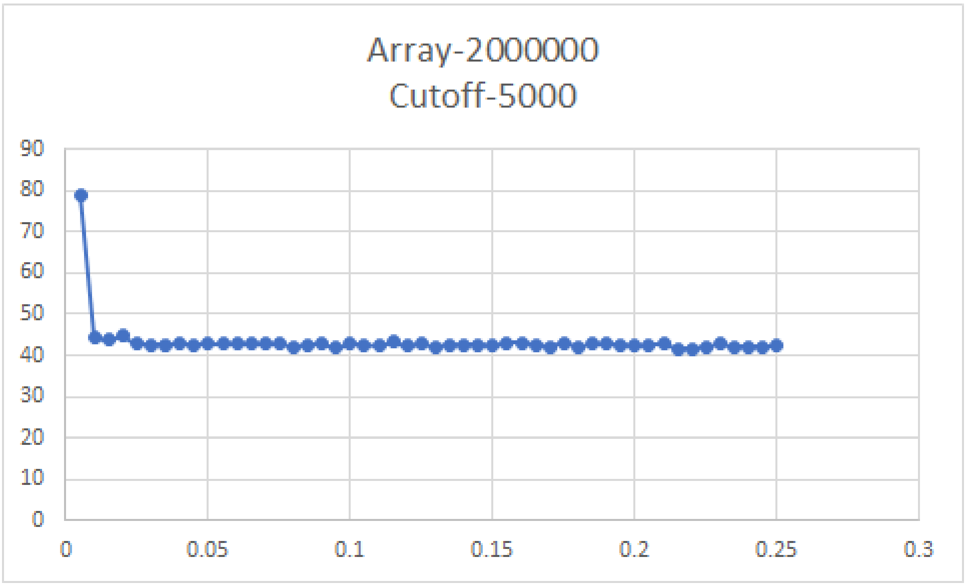


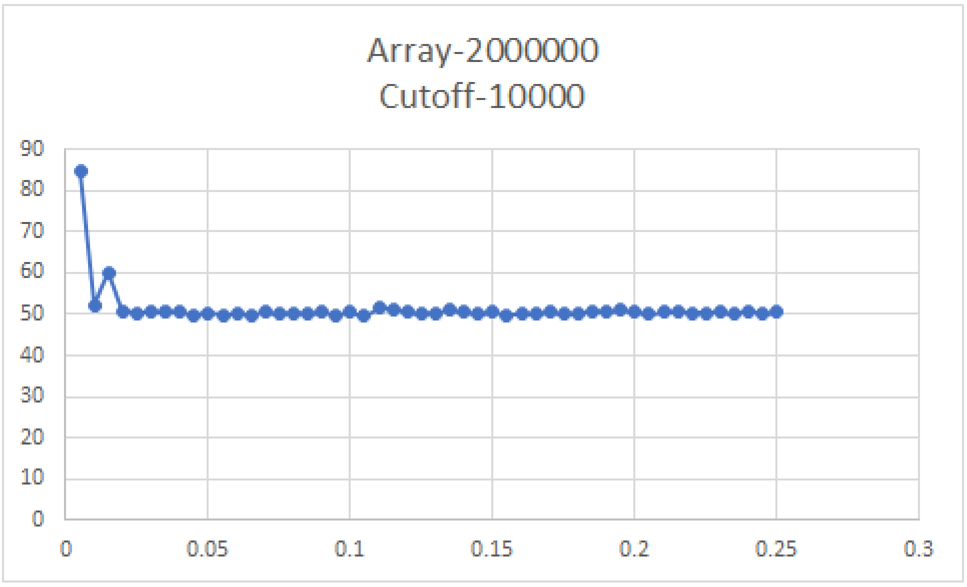




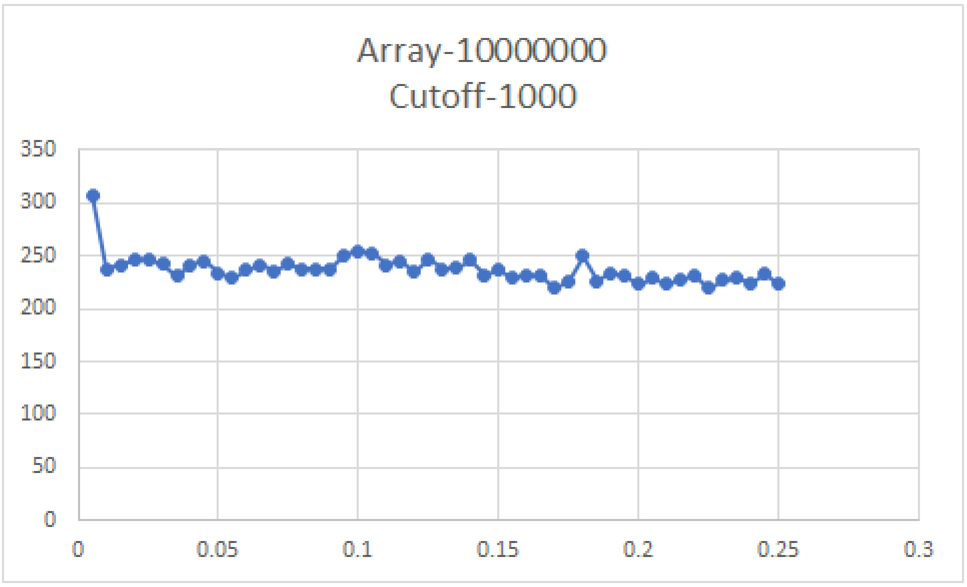
When the array comes to 2000000, we can find that when cutoff is 1000 and cutoff is 5000, the running time is almost the same; when cutoff is 10000, the running time comes to about 50ms.

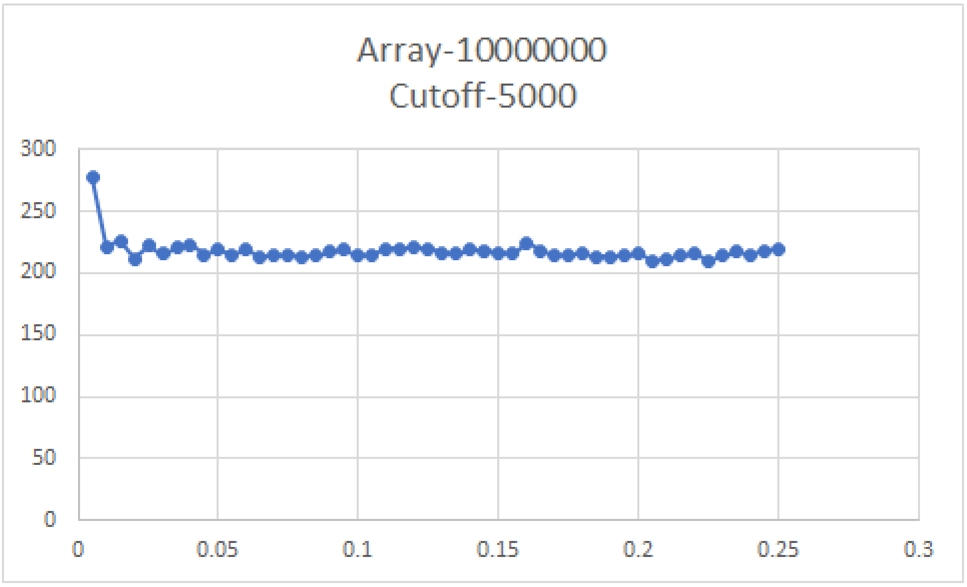


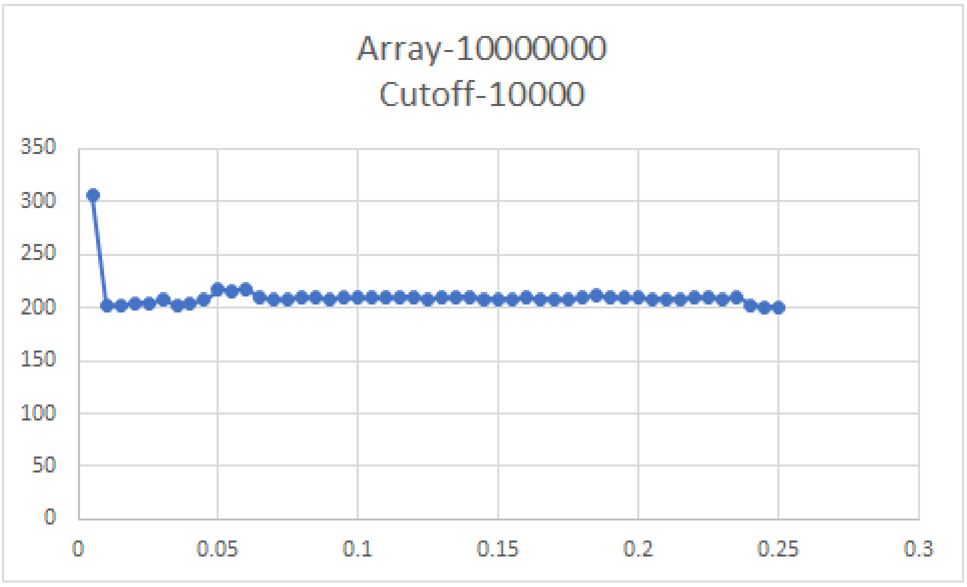




When the array is about 10000000, we can see that the run time with cutoff of 1000 and cutoff of 5000 is much higher than the run time with cutoff of 10000.







So, we can conclude that as the array size increases, the cutoff value may need to be increased accordingly.

Here is the code:

```
public static void sort(int[] array, int from, int to) {
        if (to - from < cutoff) Arrays.sort(array, from, to);
        else {
            // FIXME next few lines should be removed from public repo.
            int mid = from + (to - from) / 2;
            CompletableFuture<int[]> parsort1 = parsort(array, from, mid);
            CompletableFuture<int[]> parsort2 = parsort(array, mid, to);
            CompletableFuture<int[]> parsort = parsort1.thenCombine(parsort2,
(xs1, xs2) -> {
                int[] result = new int[xs1.length + xs2.length];
                int i = 0;
                int j = 0;
                int k = 0;
                while (i < xs1.length && j < xs2.length) {
                    result[k++] = xs1[i] <= xs2[j] ? xs1[i++] : xs2[j++];
                }
                while (i < xs1.length) {
                    result[k++] = xs1[i++];
                }
                while (j < xs2.length) {
                    result[k++] = xs2[j++];
                }
                return result;
            });

            parsort.whenComplete((result, throwable) -> System.arraycopy(result,
0, array, from, result.length));
//            System.out.println("# threads: "+
ForkJoinPool.commonPool().getRunningThreadCount());
            parsort.join();
        }
    }

    private static CompletableFuture<int[]> parsort(int[] array, int from, int
to) {
        return CompletableFuture.supplyAsync(
                () -> {
                    int[] partialArray = Arrays.copyOfRange(array, from, to);
                    sort(partialArray, 0, partialArray.length);
                    return partialArray;
                }
        );
    }
}
```