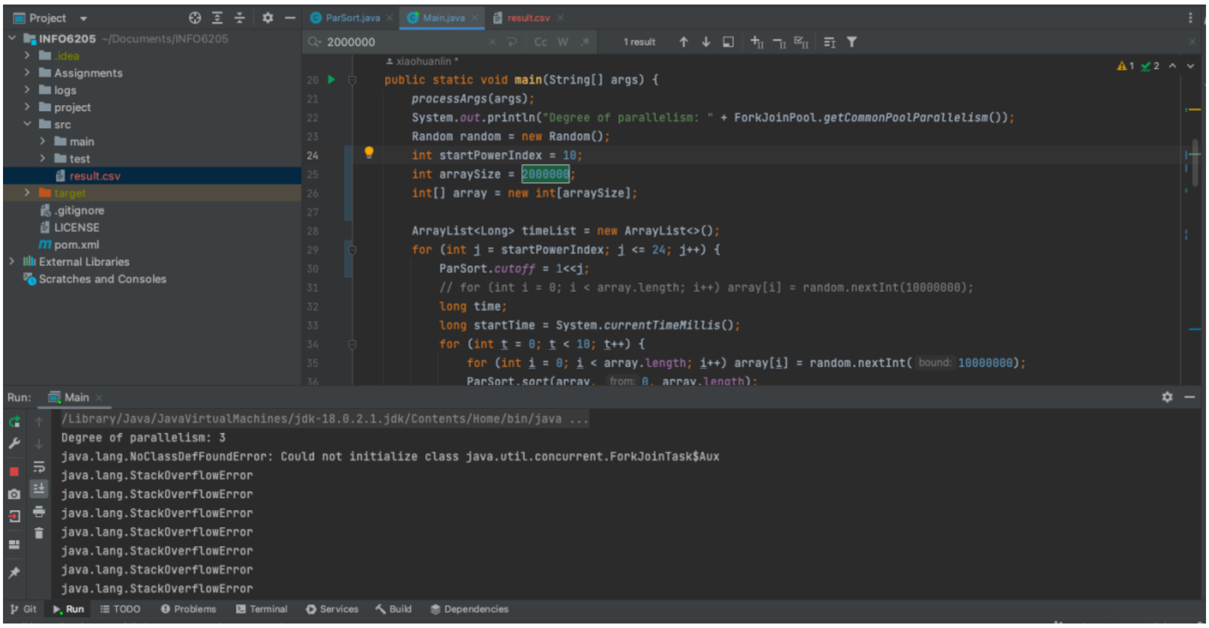
Assignment – 5

Parallel sort was executed for different array sizes. For array size of **2,000,000 elements**, different cut-off values were tried in powers of 2. For values less than 2048, stack overflow exception is thrown since the number of recursions increases exponentially as shown below.

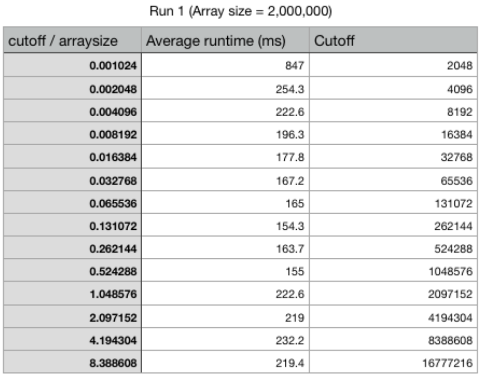


```java
public static void main(String[] args) {
    processArgs(args);
    System.out.println("Degree of parallelism: " + ForkJoinPool.getCommonPoolParallelism());
    Random random = new Random();
    int startPowerIndex = 10;
    int arraySize = 2000000;
    int[] array = new int[arraySize];

    ArrayList<Long> timeList = new ArrayList<>();
    for (int j = startPowerIndex; j <= 24; j++) {
        ParSort.cutoff = 1<<j;
        // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
        long time;
        long startTime = System.currentTimeMillis();
        for (int t = 0; t < 10; t++) {
            for (int i = 0; i < array.length; i++) array[i] = random.nextInt( bound: 10000000);
            ParSort.sort(array,  from: 0, array.length);
```

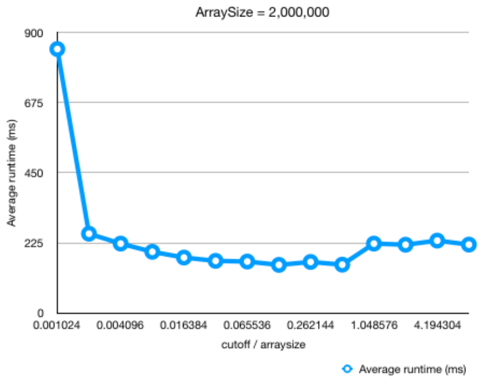
```
/Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bin/java ...
Degree of parallelism: 3
java.lang.NoClassDefFoundError: Could not initialize class java.util.concurrent.ForkJoinTask$Aux
java.lang.StackOverflowError
java.lang.StackOverflowError
java.lang.StackOverflowError
java.lang.StackOverflowError
java.lang.StackOverflowError
java.lang.StackOverflowError
java.lang.StackOverflowError
```

As we increase the value of the cutoff, the number of recursions is bound (like branch and bound recursion) and the performance also improves as they are sorted parallelly. A graphs is plotted between (cutoff/arraySize) and average run time(ms). The run time starts dropping and stabilizes after 8192 cut off as we can observe from the graph below.

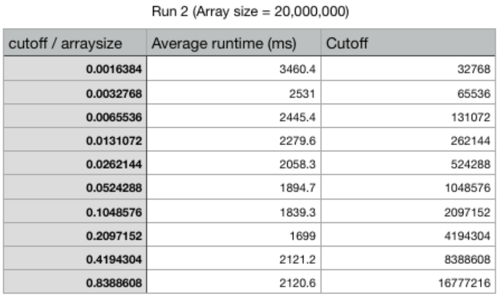


Run 1 (Array size = 2,000,000)

cutoff / arraysize	Average runtime (ms)	Cutoff
0.001024	847	2048
0.002048	254.3	4096
0.004096	222.6	8192
0.008192	196.3	16384
0.016384	177.8	32768
0.032768	167.2	65536
0.065536	165	131072
0.131072	154.3	262144
0.262144	163.7	524288
0.524288	155	1048576
1.048576	222.6	2097152
2.097152	219	4194304
4.194304	232.2	8388608
8.388608	219.4	16777216

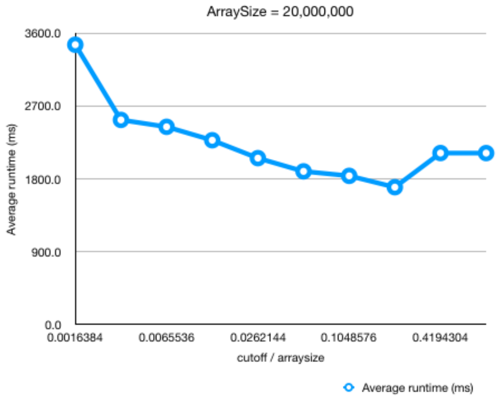


Another trial was done using an array size of **20,000,000 elements.** With this array, stack overflow exception was encountered with cut off lower than 32768. From the graph we can observe that the runtime starts to drop after 65536 cut off as shown below.



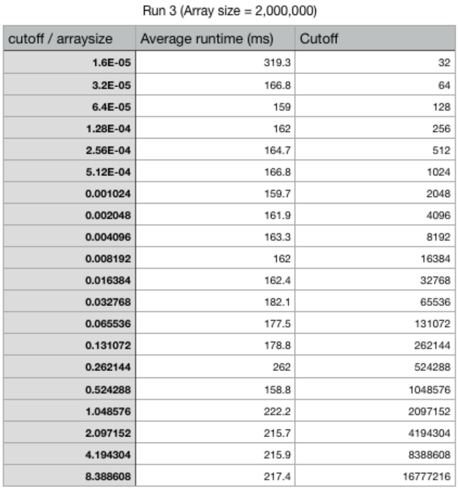
Run 2 (Array size = 20,000,000)

cutoff / arraysize	Average runtime (ms)	Cutoff
0.0016384	3460.4	32768
0.0032768	2531	65536
0.0065536	2445.4	131072
0.0131072	2279.6	262144
0.0262144	2058.3	524288
0.0524288	1894.7	1048576
0.1048576	1839.3	2097152
0.2097152	1699	4194304
0.4194304	2121.2	8388608
0.8388608	2120.6	16777216



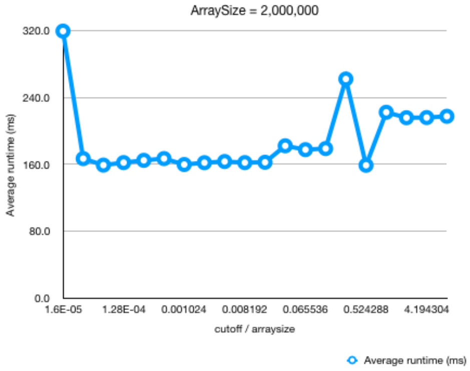
Adding Recursion Depth Optimization

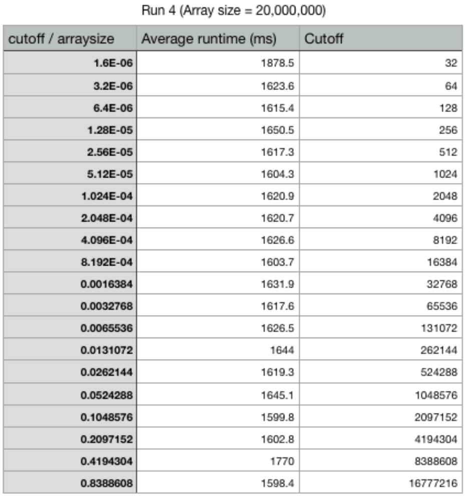
In order to make the algorithm more efficient and avoid stack overflows, another level of optimization was added to limit the depth of recursion based on the threads available on the current system. A log of number of threads is taken as max allowed depth. A condition is added to limit recursive depth based on either log of number of threads or max cutoff array size, whichever occurs earlier. With this optimization, we observe that the performance for different cut off sizes has widely stabilized without any overflow exceptions.



Run 3 (Array size = 2,000,000)

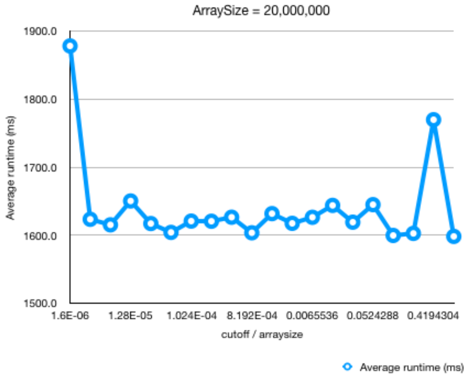
cutoff / arraysize	Average runtime (ms)	Cutoff
1.6E-05	319.3	32
3.2E-05	166.8	64
6.4E-05	159	128
1.28E-04	162	256
2.56E-04	164.7	512
5.12E-04	166.8	1024
0.001024	159.7	2048
0.002048	161.9	4096
0.004096	163.3	8192
0.008192	162	16384
0.016384	162.4	32768
0.032768	182.1	65536
0.065536	177.5	131072
0.131072	178.8	262144
0.262144	262	524288
0.524288	158.8	1048576
1.048576	222.2	2097152
2.097152	215.7	4194304
4.194304	215.9	8388608
8.388608	217.4	16777216





Run 4 (Array size = 20,000,000)

cutoff / arraysize	Average runtime (ms)	Cutoff
1.6E-06	1878.5	32
3.2E-06	1623.6	64
6.4E-06	1615.4	128
1.28E-05	1650.5	256
2.56E-05	1617.3	512
5.12E-05	1604.3	1024
1.024E-04	1620.9	2048
2.048E-04	1620.7	4096
4.096E-04	1626.6	8192
8.192E-04	1603.7	16384
0.0016384	1631.9	32768
0.0032768	1617.6	65536
0.0065536	1626.5	131072
0.0131072	1644	262144
0.0262144	1619.3	524288
0.0524288	1645.1	1048576
0.1048576	1599.8	2097152
0.2097152	1602.8	4194304
0.4194304	1770	8388608
0.8388608	1598.4	16777216



Thus, we can conclude that with a combined optimization using cutoff value and max available threads, we can make merge sort faster by processing the divided partitions parallelly.