Program Structures and Algorithms
Spring 2022

Name: Sagar Jalindar Khandagale
NUID: 002197761
Assignment No: 4 (Parallel Sort)

Task

Task is to implement a parallel sorting algorithm such that each partition of the array is sorted in parallel. You will consider two different schemes for deciding whether to sort in parallel.

1. A cutoff (defaults to, say, 1000) which you will update according to the first argument in the command line when running. It's your job to experiment and come up with a good value for this cutoff. If there are fewer elements to sort than the cutoff, then you should use the system sort instead.
2. Recursion depth or the number of available threads. Using this determination, you might decide on an ideal number (*t*) of separate threads (stick to powers of 2) and arrange for that number of partitions to be parallelized (by preventing recursion after the depth of *lg t* is reached).
3. An appropriate combination of these.

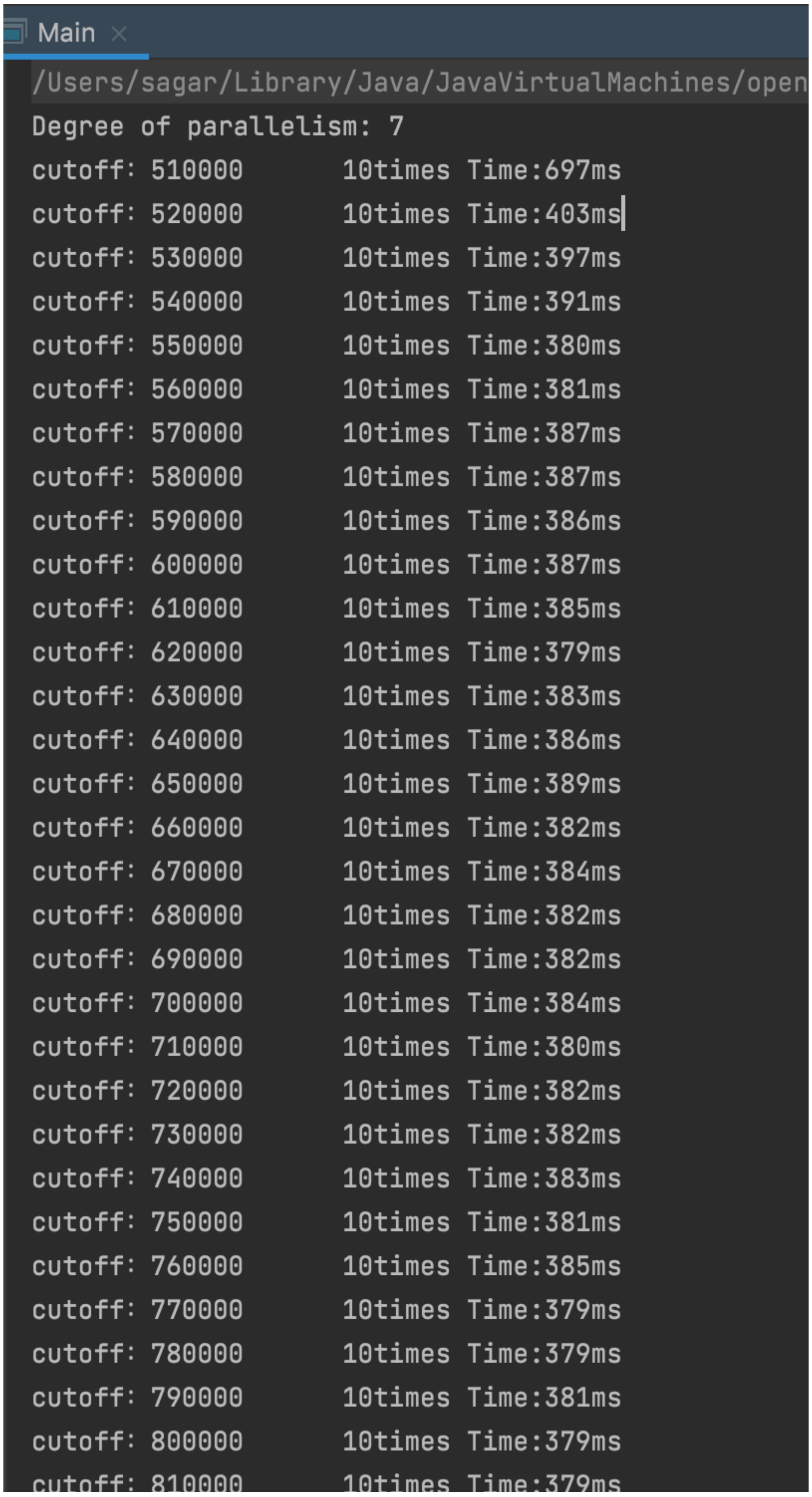
You must prepare a report that shows the results of your experiments and draws a conclusion (or more) about the efficacy of this method of parallelizing sort. Your experiments should involve sorting arrays of sufficient size for the parallel sort to make a difference. You should run with many different array sizes (they must be sufficiently large to make parallel sorting worthwhile, obviously) and different cutoff schemes.

Output:

1. Cutoff Method:

In this method we are providing different cutoff values for parallel sort. We are checking each cutoff for different size of array. We got the following output for the algorithm. In the screenshots below we can see that for cutoff ranging from 510000 to 1000000. And time is milliseconds is calculated against that for the parallel sort. Every time we are passing the different size of array from 100000-500000 and getting the following output.

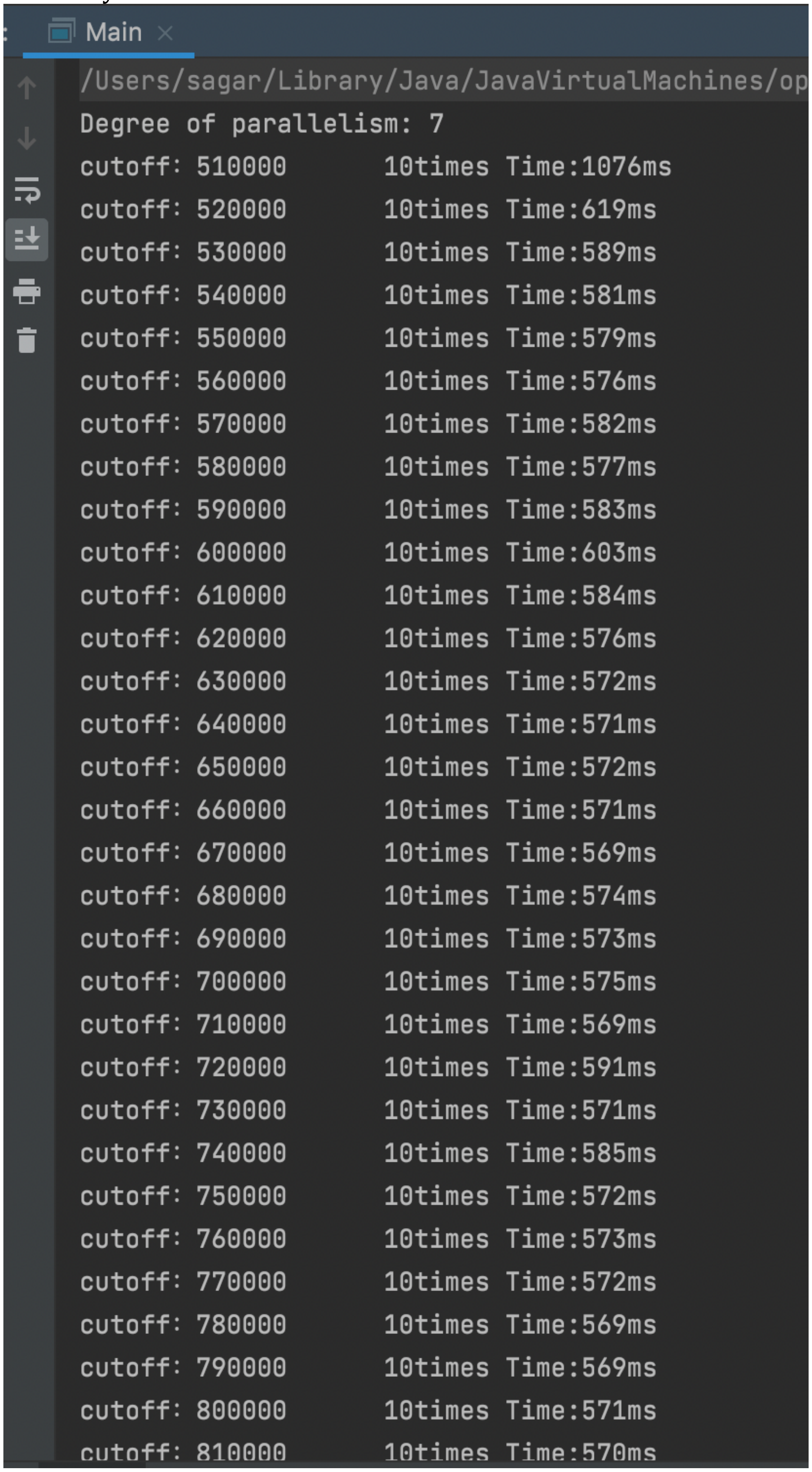
For array size: 1000000



Main ×

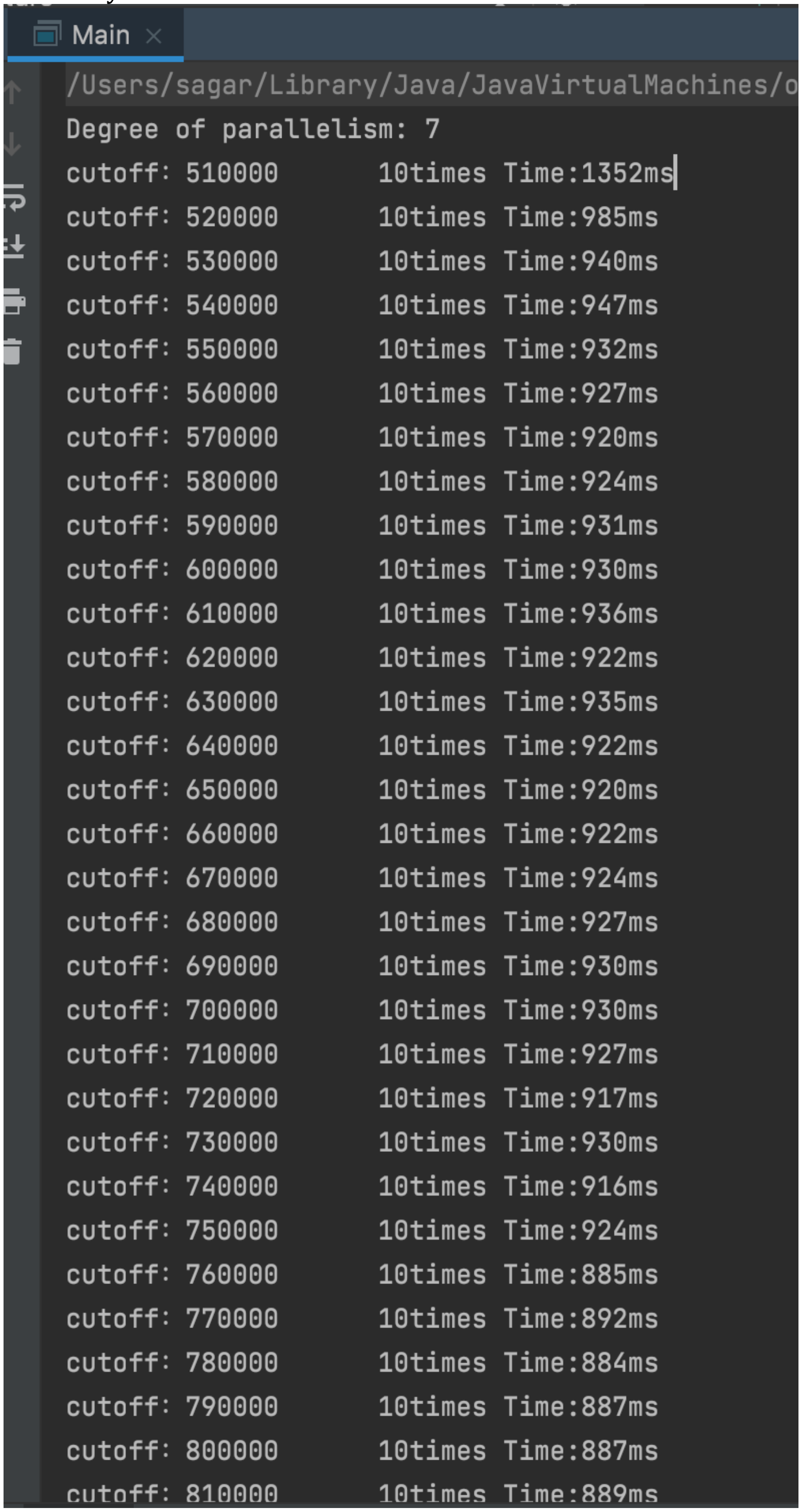
```
/Users/sagar/Library/Java/JavaVirtualMachines/open
Degree of parallelism: 7
cutoff: 510000      10times Time:697ms
cutoff: 520000      10times Time:403ms
cutoff: 530000      10times Time:397ms
cutoff: 540000      10times Time:391ms
cutoff: 550000      10times Time:380ms
cutoff: 560000      10times Time:381ms
cutoff: 570000      10times Time:387ms
cutoff: 580000      10times Time:387ms
cutoff: 590000      10times Time:386ms
cutoff: 600000      10times Time:387ms
cutoff: 610000      10times Time:385ms
cutoff: 620000      10times Time:379ms
cutoff: 630000      10times Time:383ms
cutoff: 640000      10times Time:386ms
cutoff: 650000      10times Time:389ms
cutoff: 660000      10times Time:382ms
cutoff: 670000      10times Time:384ms
cutoff: 680000      10times Time:382ms
cutoff: 690000      10times Time:382ms
cutoff: 700000      10times Time:384ms
cutoff: 710000      10times Time:380ms
cutoff: 720000      10times Time:382ms
cutoff: 730000      10times Time:382ms
cutoff: 740000      10times Time:383ms
cutoff: 750000      10times Time:381ms
cutoff: 760000      10times Time:385ms
cutoff: 770000      10times Time:379ms
cutoff: 780000      10times Time:379ms
cutoff: 790000      10times Time:381ms
cutoff: 800000      10times Time:379ms
cutoff: 810000      10times Time:379ms
```

For Array Size: 2000000



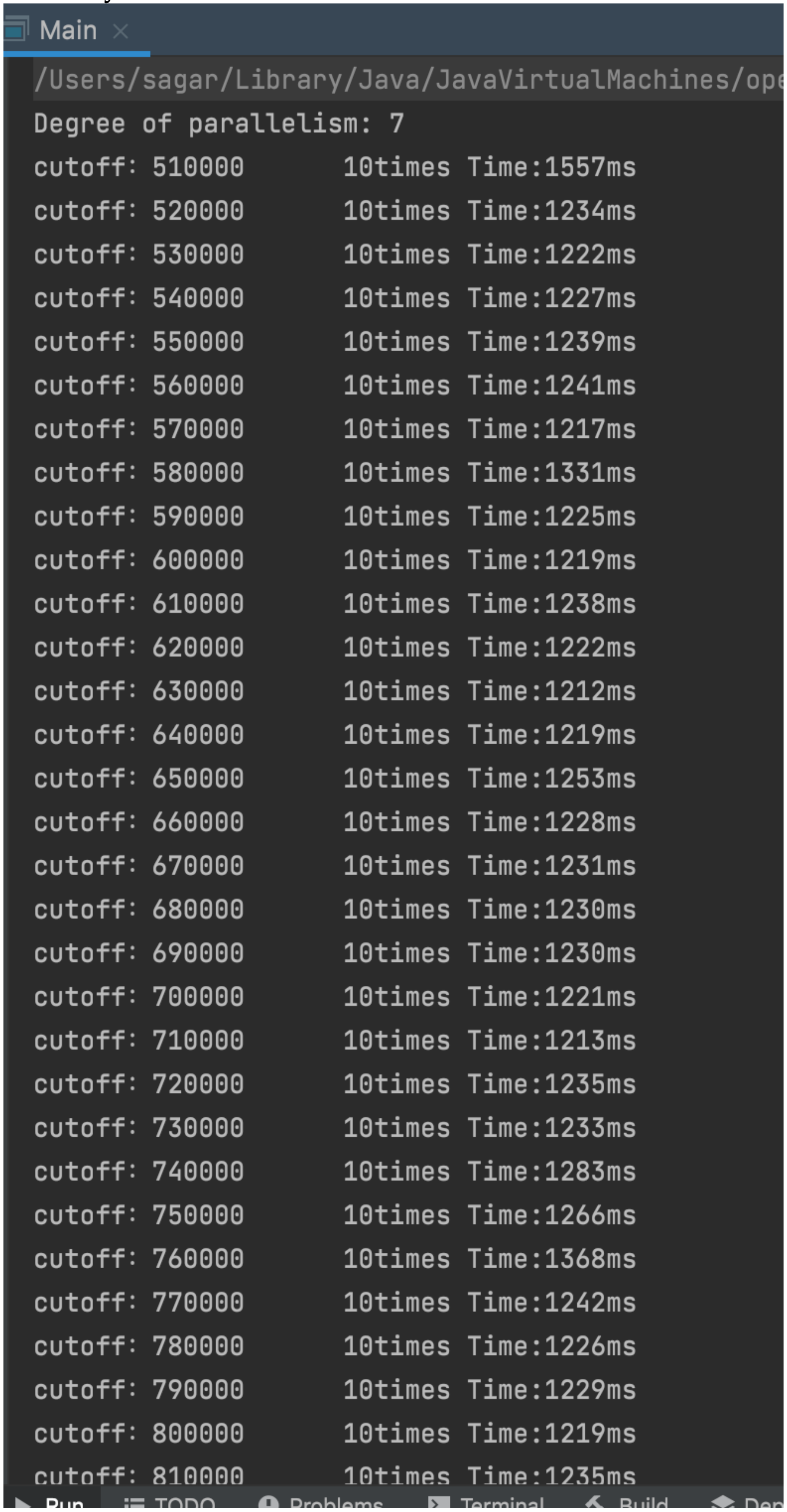
```
/Users/sagar/Library/Java/JavaVirtualMachines/op
Degree of parallelism: 7
cutoff: 510000      10times Time:1076ms
cutoff: 520000      10times Time:619ms
cutoff: 530000      10times Time:589ms
cutoff: 540000      10times Time:581ms
cutoff: 550000      10times Time:579ms
cutoff: 560000      10times Time:576ms
cutoff: 570000      10times Time:582ms
cutoff: 580000      10times Time:577ms
cutoff: 590000      10times Time:583ms
cutoff: 600000      10times Time:603ms
cutoff: 610000      10times Time:584ms
cutoff: 620000      10times Time:576ms
cutoff: 630000      10times Time:572ms
cutoff: 640000      10times Time:571ms
cutoff: 650000      10times Time:572ms
cutoff: 660000      10times Time:571ms
cutoff: 670000      10times Time:569ms
cutoff: 680000      10times Time:574ms
cutoff: 690000      10times Time:573ms
cutoff: 700000      10times Time:575ms
cutoff: 710000      10times Time:569ms
cutoff: 720000      10times Time:591ms
cutoff: 730000      10times Time:571ms
cutoff: 740000      10times Time:585ms
cutoff: 750000      10times Time:572ms
cutoff: 760000      10times Time:573ms
cutoff: 770000      10times Time:572ms
cutoff: 780000      10times Time:569ms
cutoff: 790000      10times Time:569ms
cutoff: 800000      10times Time:571ms
cutoff: 810000      10times Time:570ms
```

For Array Size: 3000000



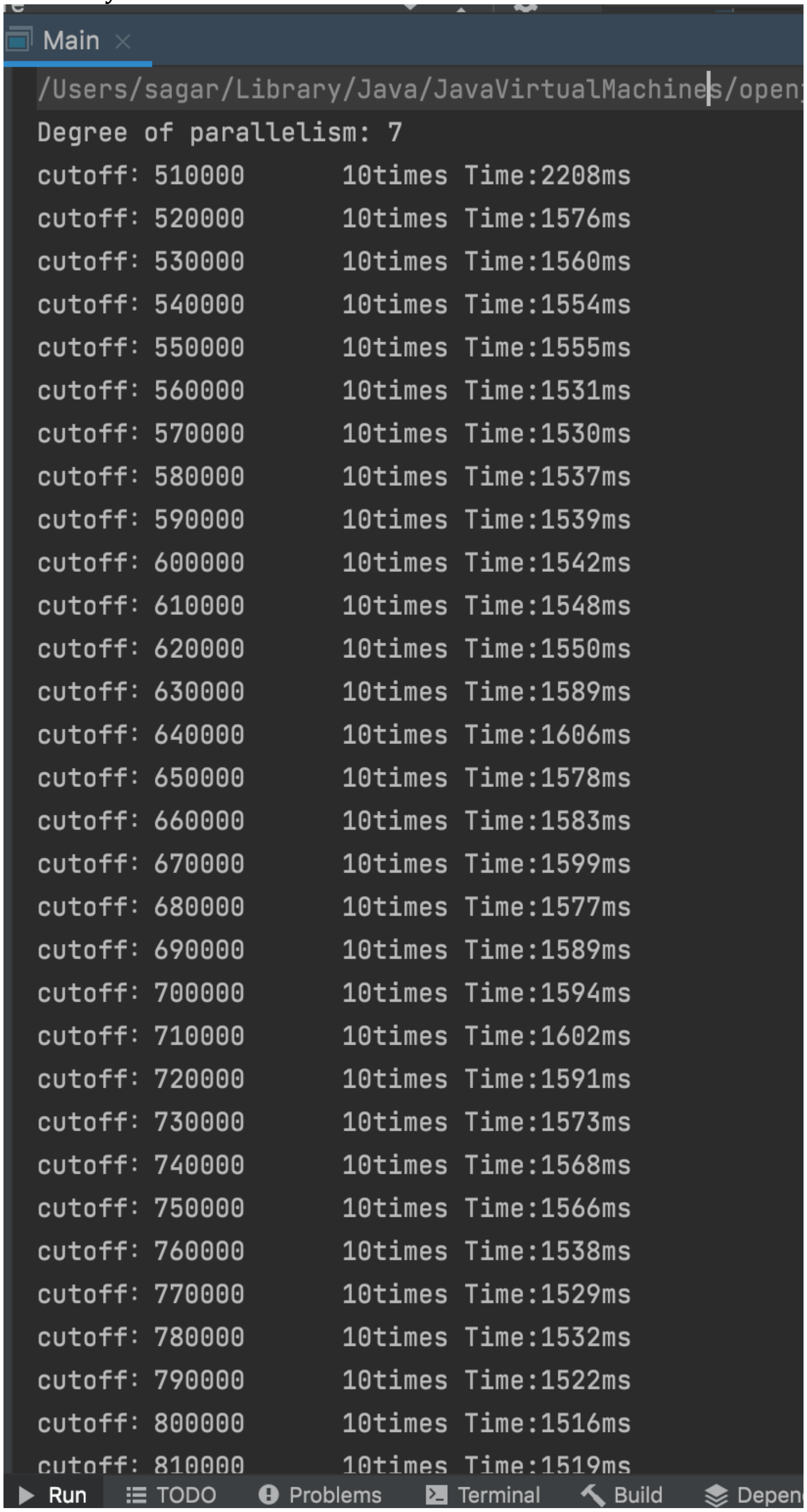
```
/Users/sagar/Library/Java/JavaVirtualMachines/o
Degree of parallelism: 7
cutoff: 510000       10times Time:1352ms
cutoff: 520000       10times Time:985ms
cutoff: 530000       10times Time:940ms
cutoff: 540000       10times Time:947ms
cutoff: 550000       10times Time:932ms
cutoff: 560000       10times Time:927ms
cutoff: 570000       10times Time:920ms
cutoff: 580000       10times Time:924ms
cutoff: 590000       10times Time:931ms
cutoff: 600000       10times Time:930ms
cutoff: 610000       10times Time:936ms
cutoff: 620000       10times Time:922ms
cutoff: 630000       10times Time:935ms
cutoff: 640000       10times Time:922ms
cutoff: 650000       10times Time:920ms
cutoff: 660000       10times Time:922ms
cutoff: 670000       10times Time:924ms
cutoff: 680000       10times Time:927ms
cutoff: 690000       10times Time:930ms
cutoff: 700000       10times Time:930ms
cutoff: 710000       10times Time:927ms
cutoff: 720000       10times Time:917ms
cutoff: 730000       10times Time:930ms
cutoff: 740000       10times Time:916ms
cutoff: 750000       10times Time:924ms
cutoff: 760000       10times Time:885ms
cutoff: 770000       10times Time:892ms
cutoff: 780000       10times Time:884ms
cutoff: 790000       10times Time:887ms
cutoff: 800000       10times Time:887ms
cutoff: 810000       10times Time:889ms
```

For Array Size: 4000000



```
/Users/sagar/Library/Java/JavaVirtualMachines/ope
Degree of parallelism: 7
cutoff: 510000      10times Time:1557ms
cutoff: 520000      10times Time:1234ms
cutoff: 530000      10times Time:1222ms
cutoff: 540000      10times Time:1227ms
cutoff: 550000      10times Time:1239ms
cutoff: 560000      10times Time:1241ms
cutoff: 570000      10times Time:1217ms
cutoff: 580000      10times Time:1331ms
cutoff: 590000      10times Time:1225ms
cutoff: 600000      10times Time:1219ms
cutoff: 610000      10times Time:1238ms
cutoff: 620000      10times Time:1222ms
cutoff: 630000      10times Time:1212ms
cutoff: 640000      10times Time:1219ms
cutoff: 650000      10times Time:1253ms
cutoff: 660000      10times Time:1228ms
cutoff: 670000      10times Time:1231ms
cutoff: 680000      10times Time:1230ms
cutoff: 690000      10times Time:1230ms
cutoff: 700000      10times Time:1221ms
cutoff: 710000      10times Time:1213ms
cutoff: 720000      10times Time:1235ms
cutoff: 730000      10times Time:1233ms
cutoff: 740000      10times Time:1283ms
cutoff: 750000      10times Time:1266ms
cutoff: 760000      10times Time:1368ms
cutoff: 770000      10times Time:1242ms
cutoff: 780000      10times Time:1226ms
cutoff: 790000      10times Time:1229ms
cutoff: 800000      10times Time:1219ms
cutoff: 810000      10times Time:1235ms
```

For Array Size: 5000000



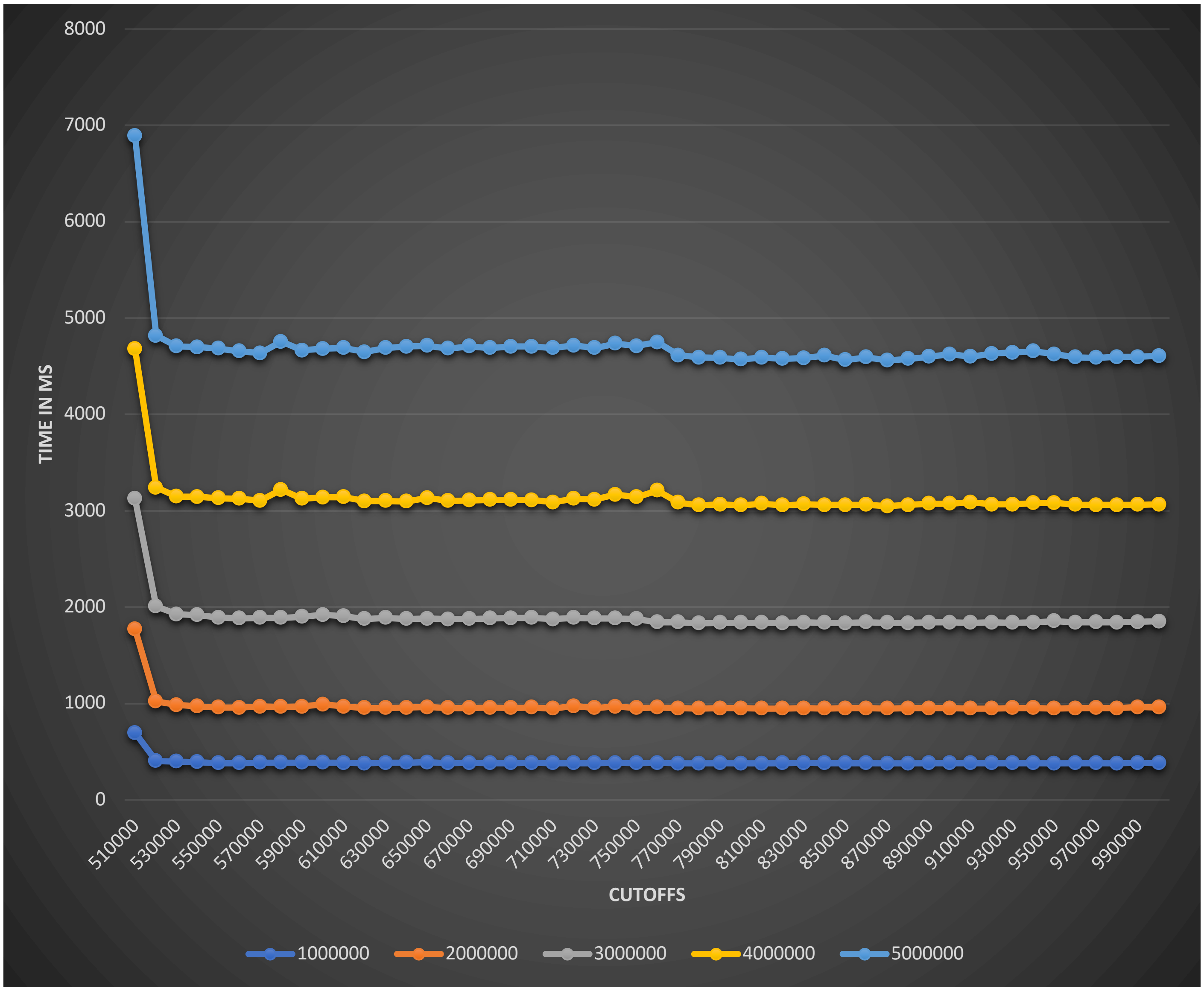
Main

```
/Users/sagar/Library/Java/JavaVirtualMachines/open
Degree of parallelism: 7
cutoff: 510000      10times Time:2208ms
cutoff: 520000      10times Time:1576ms
cutoff: 530000      10times Time:1560ms
cutoff: 540000      10times Time:1554ms
cutoff: 550000      10times Time:1555ms
cutoff: 560000      10times Time:1531ms
cutoff: 570000      10times Time:1530ms
cutoff: 580000      10times Time:1537ms
cutoff: 590000      10times Time:1539ms
cutoff: 600000      10times Time:1542ms
cutoff: 610000      10times Time:1548ms
cutoff: 620000      10times Time:1550ms
cutoff: 630000      10times Time:1589ms
cutoff: 640000      10times Time:1606ms
cutoff: 650000      10times Time:1578ms
cutoff: 660000      10times Time:1583ms
cutoff: 670000      10times Time:1599ms
cutoff: 680000      10times Time:1577ms
cutoff: 690000      10times Time:1589ms
cutoff: 700000      10times Time:1594ms
cutoff: 710000      10times Time:1602ms
cutoff: 720000      10times Time:1591ms
cutoff: 730000      10times Time:1573ms
cutoff: 740000      10times Time:1568ms
cutoff: 750000      10times Time:1566ms
cutoff: 760000      10times Time:1538ms
cutoff: 770000      10times Time:1529ms
cutoff: 780000      10times Time:1532ms
cutoff: 790000      10times Time:1522ms
cutoff: 800000      10times Time:1516ms
cutoff: 810000      10times Time:1519ms
```

Cutoffs result against each array size:

Cutoffs	**Array Size**				
	1000000	**2000000**	**3000000**	**4000000**	**5000000**
510000	697	1076	1352	1557	2208
520000	403	619	985	1234	1576
530000	397	589	940	1222	1560
540000	391	581	947	1227	1554
550000	380	579	932	1239	1555
560000	381	576	927	1241	1531
570000	387	582	920	1217	1530
580000	387	577	924	1331	1537
590000	386	583	931	1225	1539
600000	387	603	930	1219	1542
610000	385	584	936	1238	1548
620000	379	576	922	1222	1550
630000	383	572	935	1212	1589
640000	386	571	922	1219	1606
650000	389	572	920	1253	1578
660000	382	571	922	1228	1583
670000	384	569	924	1231	1599
680000	382	574	927	1230	1577
690000	382	573	930	1230	1589
700000	384	575	930	1221	1594
710000	380	569	927	1213	1602
720000	382	591	917	1235	1591
730000	382	571	930	1233	1573
740000	383	585	916	1283	1568
750000	381	572	924	1266	1566
760000	385	573	885	1368	1538
770000	379	572	892	1242	1529
780000	379	569	884	1226	1532
790000	381	569	887	1229	1522
800000	379	571	887	1219	1516
810000	379	570	889	1235	1519
820000	380	568	888	1221	1519
830000	383	568	889	1229	1516
840000	380	569	888	1223	1550
850000	381	569	886	1223	1508
860000	381	571	893	1217	1532
870000	379	570	890	1210	1512
880000	379	571	884	1226	1517
890000	381	570	891	1232	1526
900000	381	570	889	1233	1551

910000	381	568	890	1250	1512
920000	380	569	891	1226	1564
930000	381	572	885	1228	1575
940000	382	571	887	1241	1575
950000	379	568	907	1230	1542
960000	381	570	889	1224	1531
970000	382	572	890	1216	1529
980000	379	572	889	1221	1537
990000	383	578	886	1216	1535
1000000	380	579	892	1215	1542



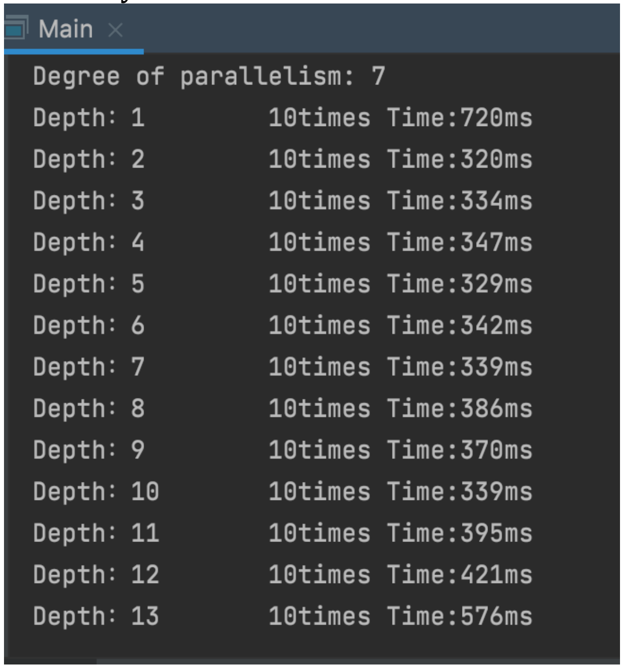
Conclusion:

From the above results and graph, we can see that for different array size cutoff value range 77000-87000 provides good results. Lowest time comes in the above range.

2. Recursion Depth:

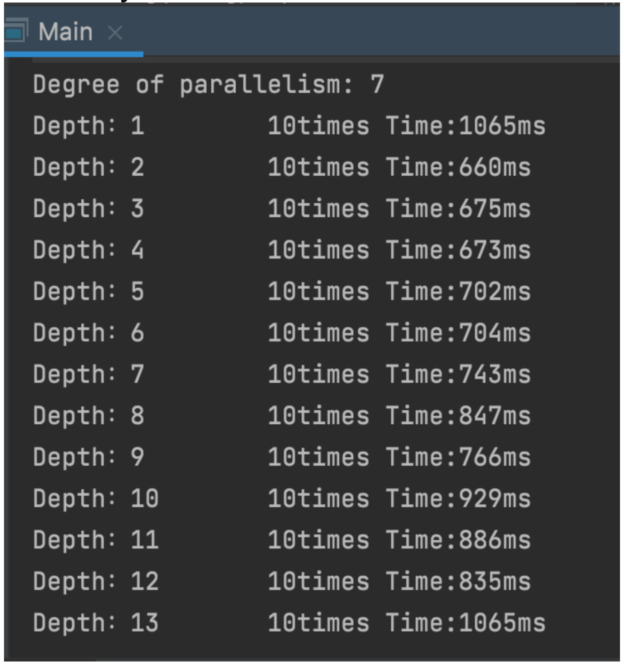
In this method we are providing different recursion depth for parallel sort. We are checking each depth for different size of array. We got the following output for the algorithm. In the screenshots below we can see that depth is ranging from 1 to 13. And time is milliseconds is calculated against that for the parallel sort. Every time we are passing the different size of array from 100000-500000 and getting the following output.

For Array Size: 1000000



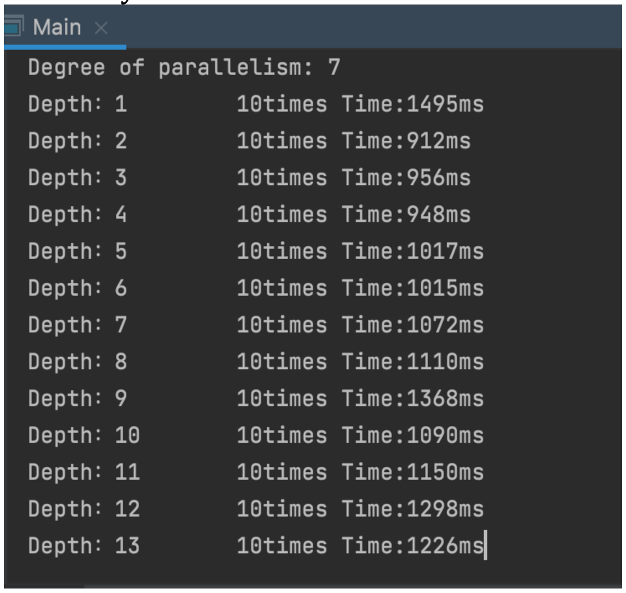
```
Main
Degree of parallelism: 7
Depth: 1        10times Time:720ms
Depth: 2        10times Time:320ms
Depth: 3        10times Time:334ms
Depth: 4        10times Time:347ms
Depth: 5        10times Time:329ms
Depth: 6        10times Time:342ms
Depth: 7        10times Time:339ms
Depth: 8        10times Time:386ms
Depth: 9        10times Time:370ms
Depth: 10       10times Time:339ms
Depth: 11       10times Time:395ms
Depth: 12       10times Time:421ms
Depth: 13       10times Time:576ms
```

For Array Size: 2000000



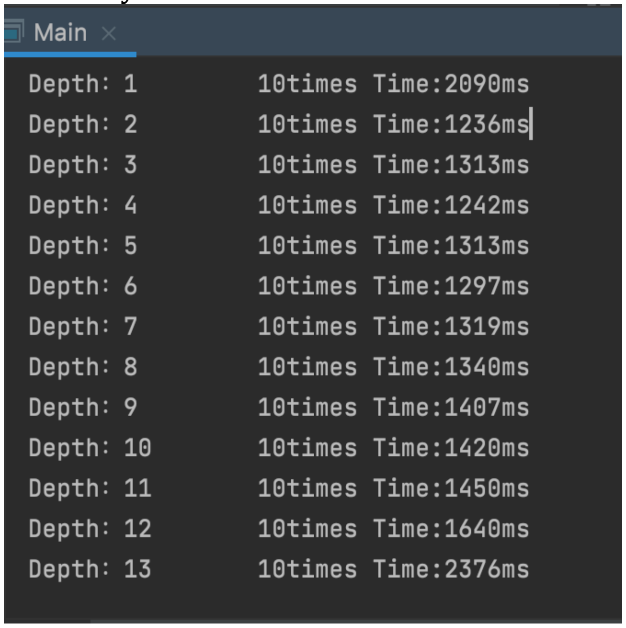
```
Main
Degree of parallelism: 7
Depth: 1        10times Time:1065ms
Depth: 2        10times Time:660ms
Depth: 3        10times Time:675ms
Depth: 4        10times Time:673ms
Depth: 5        10times Time:702ms
Depth: 6        10times Time:704ms
Depth: 7        10times Time:743ms
Depth: 8        10times Time:847ms
Depth: 9        10times Time:766ms
Depth: 10       10times Time:929ms
Depth: 11       10times Time:886ms
Depth: 12       10times Time:835ms
Depth: 13       10times Time:1065ms
```

For Array Size: 3000000



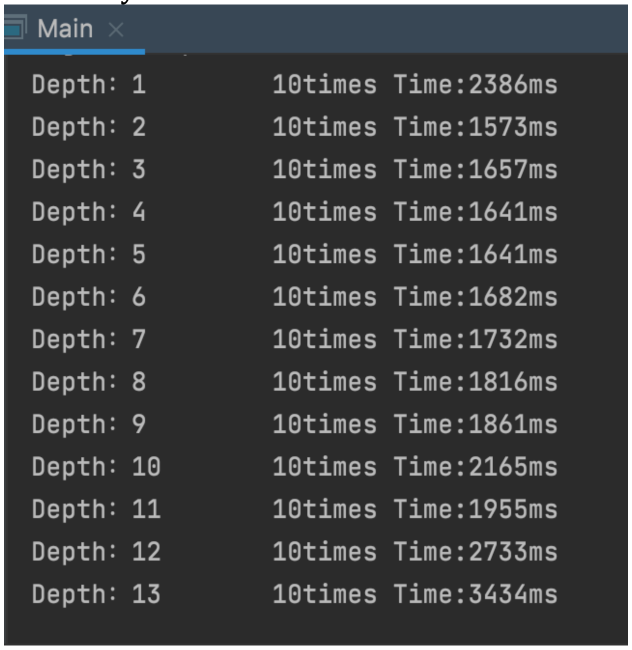
```
Degree of parallelism: 7
Depth: 1        10times Time:1495ms
Depth: 2        10times Time:912ms
Depth: 3        10times Time:956ms
Depth: 4        10times Time:948ms
Depth: 5        10times Time:1017ms
Depth: 6        10times Time:1015ms
Depth: 7        10times Time:1072ms
Depth: 8        10times Time:1110ms
Depth: 9        10times Time:1368ms
Depth: 10       10times Time:1090ms
Depth: 11       10times Time:1150ms
Depth: 12       10times Time:1298ms
Depth: 13       10times Time:1226ms
```

For Array Size: 4000000



```
Depth: 1         10times Time:2090ms
Depth: 2         10times Time:1236ms
Depth: 3         10times Time:1313ms
Depth: 4         10times Time:1242ms
Depth: 5         10times Time:1313ms
Depth: 6         10times Time:1297ms
Depth: 7         10times Time:1319ms
Depth: 8         10times Time:1340ms
Depth: 9         10times Time:1407ms
Depth: 10        10times Time:1420ms
Depth: 11        10times Time:1450ms
Depth: 12        10times Time:1640ms
Depth: 13        10times Time:2376ms
```

For Array Size: 5000000

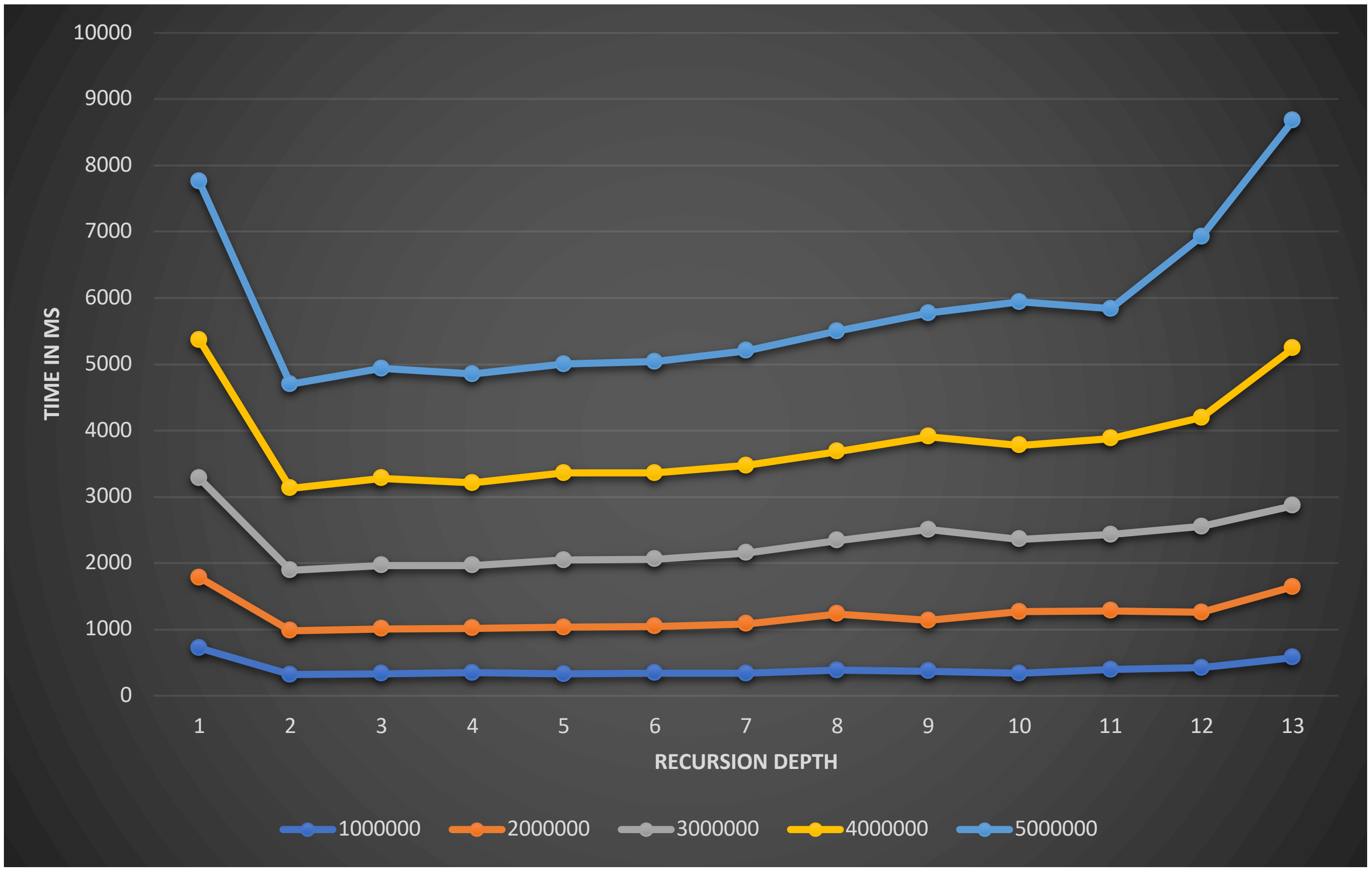


Main

```
Depth: 1          10times Time:2386ms
Depth: 2          10times Time:1573ms
Depth: 3          10times Time:1657ms
Depth: 4          10times Time:1641ms
Depth: 5          10times Time:1641ms
Depth: 6          10times Time:1682ms
Depth: 7          10times Time:1732ms
Depth: 8          10times Time:1816ms
Depth: 9          10times Time:1861ms
Depth: 10         10times Time:2165ms
Depth: 11         10times Time:1955ms
Depth: 12         10times Time:2733ms
Depth: 13         10times Time:3434ms
```

Recursion Depth result against each array size:

Depth	**Array Size**				
	1000000	**2000000**	**3000000**	**4000000**	**5000000**
1	720	1065	1495	2090	2386
2	320	660	912	1236	1573
3	334	675	956	1313	1657
4	347	673	948	1242	1641
5	329	702	1017	1313	1641
6	342	704	1015	1297	1682
7	339	743	1072	1319	1732
8	386	847	1110	1340	1816
9	370	766	1368	1407	1861
10	339	929	1090	1420	2165
11	395	886	1150	1450	1955
12	421	835	1298	1640	2733
13	576	1065	1226	2376	3434



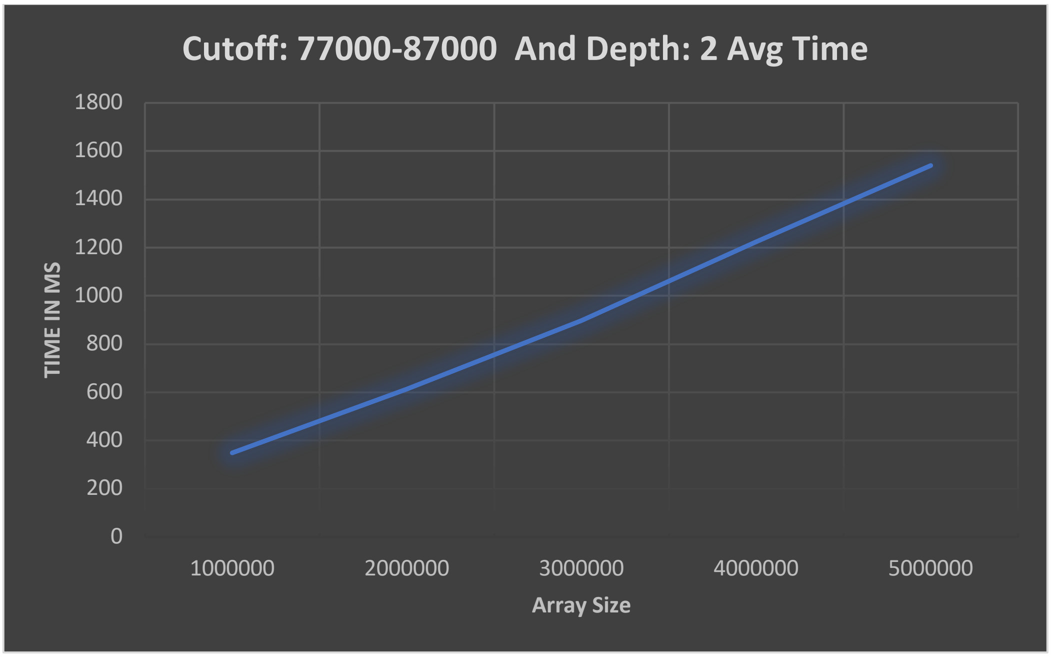
Conclusion:

From the above result and graph we can observe that for different array size at recursion depth 2 parallel sort performs better. Lowest time occurs at Depth 2.

3. Combination of above 2

We have mapped best cutoff range with best recursion depth and calculated average value of time in ms against each array size as below.

Cutoff: 77000-87000	Depth: 2		Cutoff: 77000-87000 And Depth: 2 Avg Time
379	320	**1000000**	349.5
568	660	**2000000**	614
884	912	**3000000**	898
1210	1236	**4000000**	1223
1508	1573	**5000000**	1540.5


Cutoff: 77000-87000 And Depth: 2 Avg Time
TIME IN MS
Array Size