Abhishek Satbhai (001587689)

Program Structures & Algorithms

Fall 2021

Assignment No. 5

Task:

1. **Take cutoff in argument for command line and experiment and come up with good value for cutoff**
2. **Show performance of each thread working in parallel, show evidence to support your performance and attach output screenshot**
3. **Conclusion of assignments**

Part 1.

Take cutoff in argument for command line and experiment and come up with good value for cutoff

Implemented Multiple thread for parallel sorting

```
for (int threadCount=1; threadCount< 10; threadCount= threadCount*2){
    ForkJoinPool myPool = new ForkJoinPool(threadCount);
    double avg = 0.0;
    double min = 99999;
    int best_estimated_cutoff = 0;
    for (int j = 50; j < 100; j++) {
        ParSort.cutoff = cut_off * (j + 1);
        // 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, myPool);
        }
        long endTime = System.currentTimeMillis();
        time = (endTime - startTime);
        avg = (avg + time)/ 10;
        if (time/10 < min){
            min = time/10;
            best_estimated_cutoff = cut_off * (j + 1);
        }
        timeList.add(time);

    }
    System.out.println("Thread Number: "+threadCount+ " cutoff: " + best_estimated_cutoff + "\t\t Average time:" + avg/50 + "ms");
}
```

Implemented ForkJoinPool “mypool” for sort function

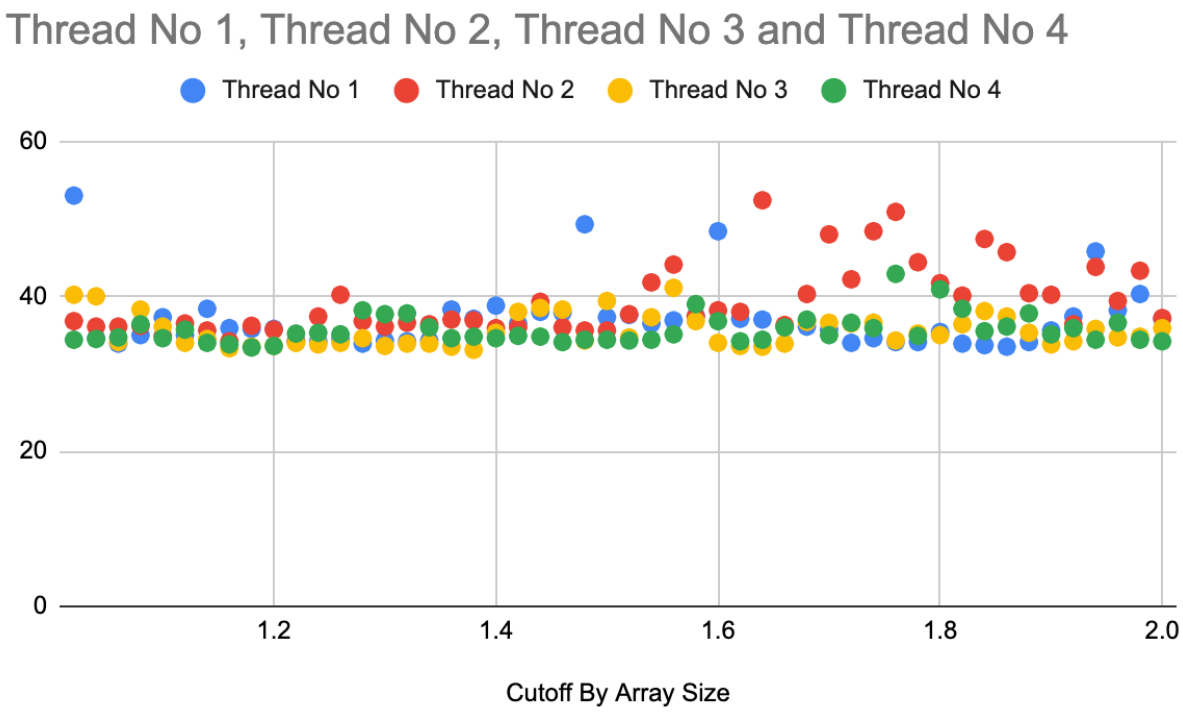
```
public static void sort(int[] array, int from, int to, ForkJoinPool mypool) {
    if (to - from < cutoff) Arrays.sort(array, from, to);
    else {
        // FIXME next few lines should be removed from public repo.
        CompletableFuture<int[]> parsort1 = parsort(array, from,  to: from + (to -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

Implemented Completable Future

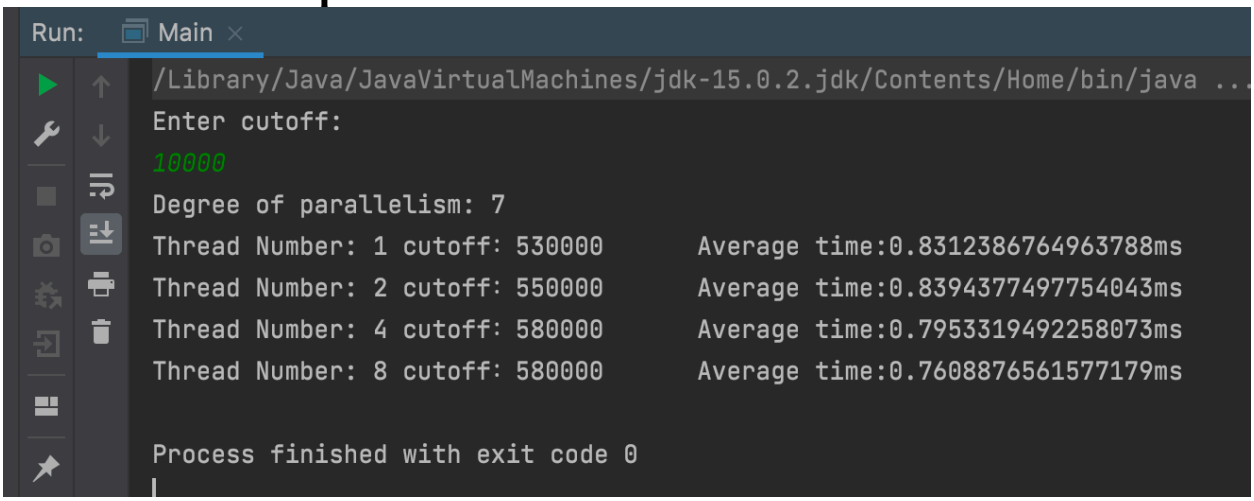
```
    private static CompletableFuture<int[]> parsort(int[] array, int from, int to, ForkJoinPool mypool) {
        return CompletableFuture.supplyAsync(
                () -> {
                    int[] result = new int[to - from];
                    // TO IMPLEMENT
                    System.arraycopy(array, from, result,  destPos: 0, result.length);
                    sort(result,  from: 0,  to: to - from, mypool);
                    return result;
                }, mypool
        );
    }
}
```

Part 2 and 3 :

For Array size : 500000 with 4 Threads Graph of Performance:-



Screenshot of output :-



Results. CSV :-

Cut off by array size	Time (ms)	Threads
1.02	53	0
1.04	34.6	0
1.06	33.9	0
1.08	35	0
1.1	37.3	0
1.12	35	0
1.14	38.4	0
1.16	35.9	0
1.18	35.8	0
1.2	35.8	0
1.22	34.3	0
1.24	34.2	0
1.26	34.5	0
1.28	33.9	0
1.3	34.3	0
1.32	34.2	0
1.34	34.3	0
1.36	38.3	0
1.38	37.1	0
1.4	38.8	0
1.42	36.5	0
1.44	38	0
1.46	37.9	0
1.48	49.3	0
1.5	37.3	0
1.52	37.6	0
1.54	36.5	0
1.56	36.9	0
1.58	37.5	0
1.6	48.4	0
1.62	37.1	0
1.64	37	0
1.66	36.1	0
1.68	36.1	0
1.7	35.7	0
1.72	34	0
1.74	34.6	0
1.76	34.1	0

1.78	34.1	0
1.8	35.4	0
1.82	33.9	0
1.84	33.7	0
1.86	33.5	0
1.88	34.1	0
1.9	35.6	0
1.92	37.4	0
1.94	45.8	0
1.96	38.2	0
1.98	40.3	0
2	37.1	0
1.02	36.8	1
1.04	36.1	1
1.06	36.1	1
1.08	36.2	1
1.1	34.9	1
1.12	36.5	1
1.14	35.6	1
1.16	34.2	1
1.18	36.2	1
1.2	35.7	1
1.22	34.1	1
1.24	37.4	1
1.26	40.2	1
1.28	36.8	1
1.3	36.1	1
1.32	36.6	1
1.34	36.4	1
1.36	37	1
1.38	36.9	1
1.4	35.9	1
1.42	36	1
1.44	39.3	1
1.46	36	1
1.48	35.6	1
1.5	35.6	1
1.52	37.7	1
1.54	41.8	1
1.56	44.1	1

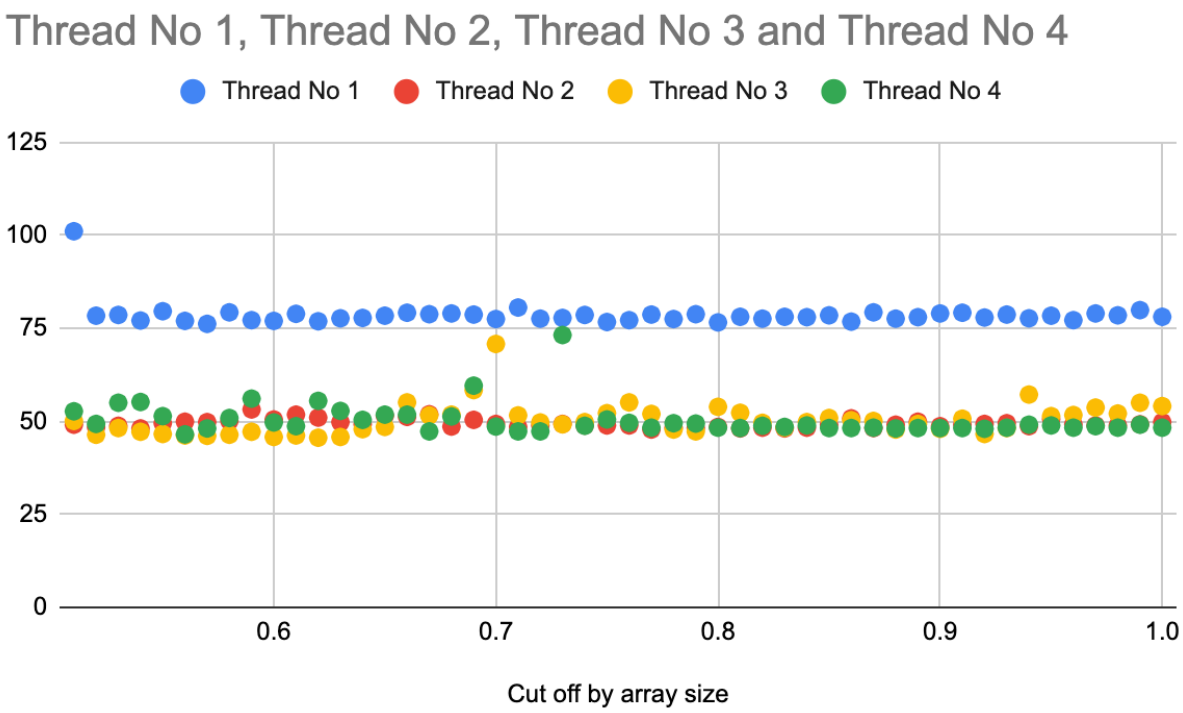
1.58	37.4	1
1.6	38.2	1
1.62	38	1
1.64	52.4	1
1.66	36.3	1
1.68	40.3	1
1.7	48	1
1.72	42.2	1
1.74	48.4	1
1.76	50.9	1
1.78	44.4	1
1.8	41.7	1
1.82	40.1	1
1.84	47.4	1
1.86	45.7	1
1.88	40.4	1
1.9	40.2	1
1.92	36.4	1
1.94	43.8	1
1.96	39.4	1
1.98	43.3	1
2	37.2	1
1.02	40.2	2
1.04	40	2
1.06	34.1	2
1.08	38.3	2
1.1	36.1	2
1.12	34	2
1.14	34.5	2
1.16	33.3	2
1.18	33.5	2
1.2	33.6	2
1.22	34	2
1.24	33.8	2
1.26	34	2
1.28	34.6	2
1.3	33.6	2
1.32	33.9	2
1.34	33.9	2
1.36	33.5	2

1.38	33.1	2
1.4	35.3	2
1.42	38	2
1.44	38.5	2
1.46	38.3	2
1.48	34.3	2
1.5	39.4	2
1.52	34.7	2
1.54	37.3	2
1.56	41.1	2
1.58	36.8	2
1.6	34	2
1.62	33.6	2
1.64	33.5	2
1.66	33.9	2
1.68	36.7	2
1.7	36.6	2
1.72	36.5	2
1.74	36.6	2
1.76	34.3	2
1.78	35.2	2
1.8	35	2
1.82	36.4	2
1.84	38.1	2
1.86	37.4	2
1.88	35.3	2
1.9	33.8	2
1.92	34.2	2
1.94	35.8	2
1.96	34.7	2
1.98	34.8	2
2	35.9	2
1.02	34.4	3
1.04	34.5	3
1.06	34.7	3
1.08	36.4	3
1.1	34.6	3
1.12	35.7	3
1.14	34	3
1.16	33.8	3

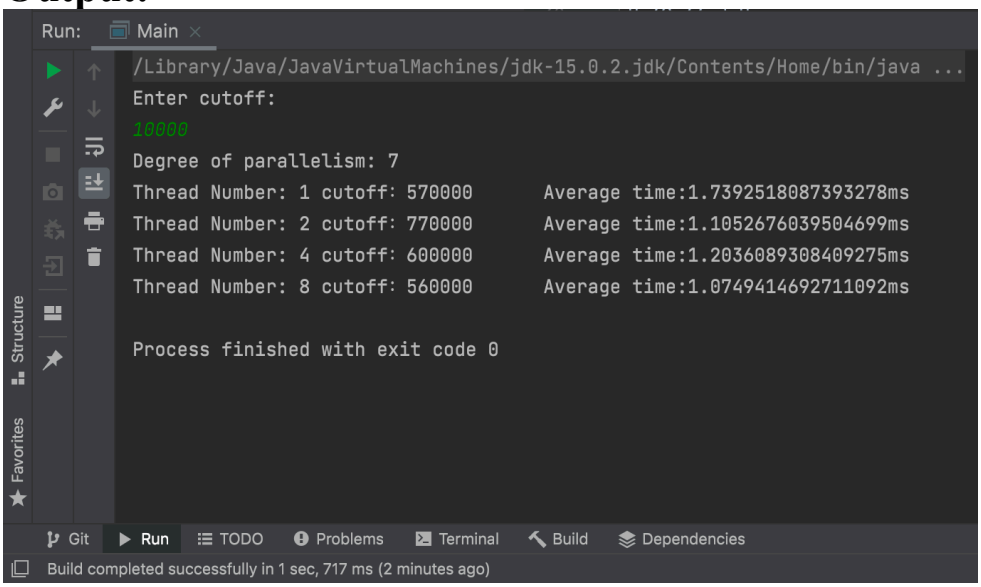
1.18	33.4	3
1.2	33.6	3
1.22	35.2	3
1.24	35.3	3
1.26	35.1	3
1.28	38.2	3
1.3	37.7	3
1.32	37.8	3
1.34	36	3
1.36	34.6	3
1.38	34.8	3
1.4	34.6	3
1.42	34.9	3
1.44	34.8	3
1.46	34.1	3
1.48	34.4	3
1.5	34.4	3
1.52	34.3	3
1.54	34.4	3
1.56	35.1	3
1.58	39	3
1.6	36.8	3
1.62	34.2	3
1.64	34.4	3
1.66	36	3
1.68	37	3
1.7	35	3
1.72	36.6	3
1.74	35.9	3
1.76	42.9	3
1.78	34.9	3
1.8	40.9	3
1.82	38.4	3
1.84	35.5	3
1.86	36.1	3
1.88	37.8	3
1.9	35.1	3
1.92	35.9	3
1.94	34.4	3
1.96	36.6	3

1.98	34.4	3
2	34.2	3

For Array Size 1000000 and 4 threads graph of performance:-



Output:-



Results Csv:

Cut off by array size	Time (ms)	Threads
0.51	101.1	0
0.52	78.4	0
0.53	78.6	0
0.54	77.1	0
0.55	79.6	0
0.56	77	0
0.57	76.2	0
0.58	79.3	0
0.59	77.2	0
0.6	77	0
0.61	78.9	0
0.62	76.9	0
0.63	77.7	0
0.64	77.8	0
0.65	78.4	0
0.66	79.2	0
0.67	78.8	0
0.68	79	0
0.69	78.7	0
0.7	77.5	0
0.71	80.6	0
0.72	77.6	0
0.73	77.8	0
0.74	78.6	0
0.75	76.7	0
0.76	77.2	0
0.77	78.7	0
0.78	77.5	0
0.79	78.8	0
0.8	76.6	0
0.81	78.1	0
0.82	77.6	0
0.83	78.1	0
0.84	78	0
0.85	78.5	0
0.86	76.8	0
0.87	79.3	0

0.88	77.6	0
0.89	78	0
0.9	79	0
0.91	79.2	0
0.92	77.9	0
0.93	78.7	0
0.94	77.7	0
0.95	78.4	0
0.96	77.2	0
0.97	79	0
0.98	78.5	0
0.99	79.9	0
1	78.1	0
0.51	49.1	1
0.52	48.4	1
0.53	48.8	1
0.54	48.1	1
0.55	49.5	1
0.56	49.9	1
0.57	49.8	1
0.58	50.1	1
0.59	53.2	1
0.6	50.5	1
0.61	51.8	1
0.62	51	1
0.63	49.8	1
0.64	49.2	1
0.65	51.5	1
0.66	51.2	1
0.67	51.9	1
0.68	48.6	1
0.69	50.4	1
0.7	49.3	1
0.71	48.5	1
0.72	48.9	1
0.73	49.3	1
0.74	48.9	1
0.75	48.9	1
0.76	48.9	1
0.77	47.8	1

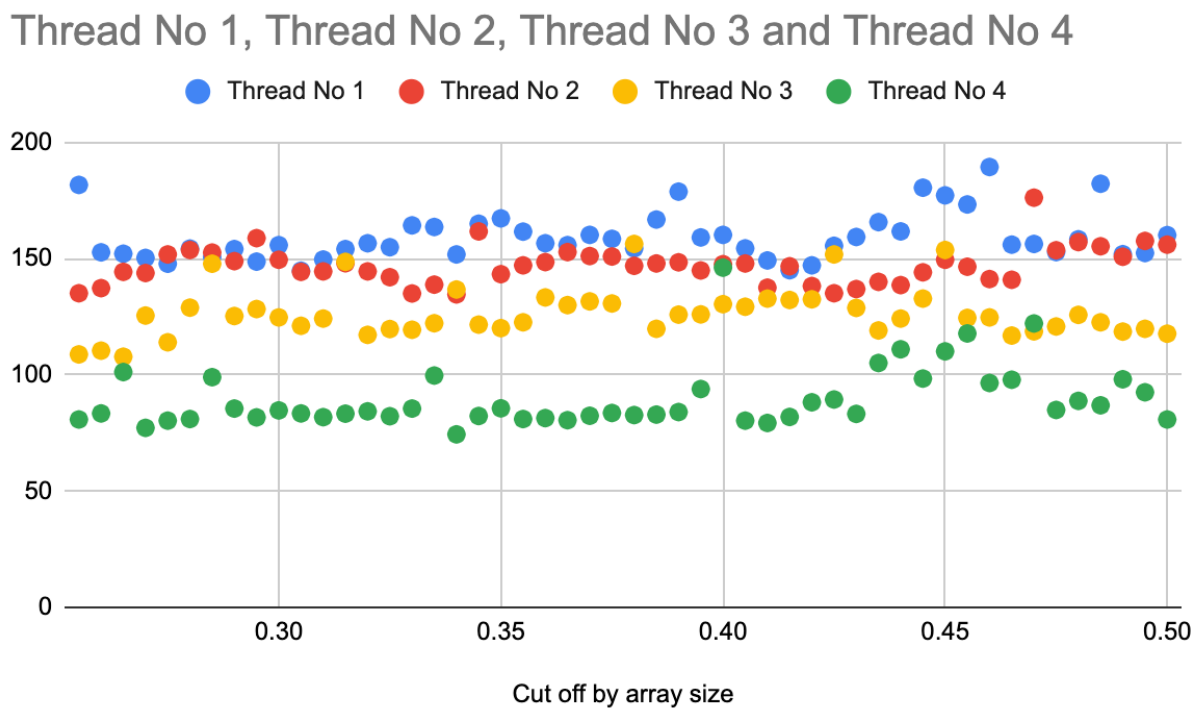
0.78	48.3	1
0.79	48.2	1
0.8	48.5	1
0.81	48.1	1
0.82	48.3	1
0.83	48.1	1
0.84	48.3	1
0.85	49.1	1
0.86	50.8	1
0.87	48.2	1
0.88	49.1	1
0.89	49.9	1
0.9	48.7	1
0.91	49.1	1
0.92	49.3	1
0.93	49.5	1
0.94	48.7	1
0.95	50.6	1
0.96	49.2	1
0.97	48.9	1
0.98	48.9	1
0.99	49.2	1
1	49.8	1
0.51	50.1	2
0.52	46.4	2
0.53	48.2	2
0.54	47.2	2
0.55	46.6	2
0.56	46.2	2
0.57	46.1	2
0.58	46.4	2
0.59	47.2	2
0.6	45.8	2
0.61	46.2	2
0.62	45.6	2
0.63	45.8	2
0.64	47.9	2
0.65	48.5	2
0.66	55.1	2
0.67	51.6	2

0.68	51.8	2
0.69	58.4	2
0.7	70.8	2
0.71	51.6	2
0.72	49.7	2
0.73	49.2	2
0.74	49.8	2
0.75	52.2	2
0.76	55.1	2
0.77	52	2
0.78	47.8	2
0.79	47.3	2
0.8	53.9	2
0.81	52.3	2
0.82	49.6	2
0.83	48.3	2
0.84	49.8	2
0.85	50.9	2
0.86	50.3	2
0.87	50.1	2
0.88	47.8	2
0.89	49.3	2
0.9	48	2
0.91	50.7	2
0.92	46.6	2
0.93	48.2	2
0.94	57.2	2
0.95	51.4	2
0.96	51.7	2
0.97	53.7	2
0.98	52.1	2
0.99	55	2
1	54.1	2
0.51	52.7	3
0.52	49.3	3
0.53	55	3
0.54	55.2	3
0.55	51.4	3
0.56	46.5	3
0.57	48.1	3

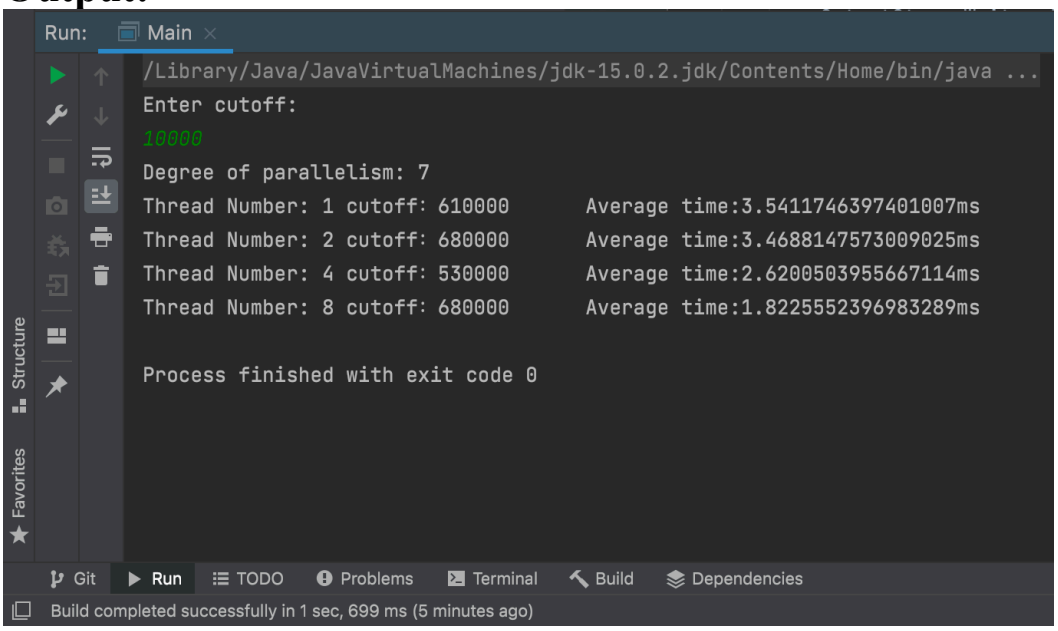
0.58	50.9	3
0.59	56.1	3
0.6	49.7	3
0.61	48.7	3
0.62	55.5	3
0.63	52.8	3
0.64	50.4	3
0.65	51.8	3
0.66	51.8	3
0.67	47.3	3
0.68	51.3	3
0.69	59.6	3
0.7	48.6	3
0.71	47.3	3
0.72	47.3	3
0.73	73.2	3
0.74	48.8	3
0.75	50.5	3
0.76	49.6	3
0.77	48.3	3
0.78	49.5	3
0.79	49.4	3
0.8	48.3	3
0.81	48.2	3
0.82	48.8	3
0.83	48.5	3
0.84	48.9	3
0.85	48.2	3
0.86	48.2	3
0.87	48.3	3
0.88	48.1	3
0.89	48.2	3
0.9	48.2	3
0.91	48.2	3
0.92	48	3
0.93	48.3	3
0.94	49.1	3
0.95	48.9	3
0.96	48.3	3
0.97	48.7	3

0.98	48.3	3
0.99	49.1	3
1	48.3	3

For array size 2000000 with 4 threads graph performance:-



Output:-



Results csv:-

Cut off by array size	Time (ms)	Threads
0.255	181.7	0
0.26	152.7	0
0.265	152.1	0
0.27	150.3	0
0.275	147.8	0
0.28	154.3	0
0.285	150.6	0
0.29	154.1	0
0.295	148.7	0
0.3	155.8	0
0.305	144.8	0
0.31	149.6	0
0.315	154.1	0
0.32	156.6	0
0.325	154.9	0
0.33	164.3	0
0.335	163.6	0
0.34	151.8	0
0.345	165	0
0.35	167.3	0
0.355	161.6	0
0.36	156.6	0
0.365	155.8	0
0.37	160.2	0
0.375	158.5	0
0.38	154.4	0
0.385	166.8	0
0.39	178.8	0
0.395	159.1	0
0.4	160.2	0
0.405	154.4	0
0.41	149.2	0
0.415	145	0
0.42	147.1	0
0.425	155.5	0
0.43	159.3	0
0.435	165.7	0

0.44	161.7	0
0.445	180.5	0
0.45	177.2	0
0.455	173.3	0
0.46	189.4	0
0.465	156	0
0.47	156.3	0
0.475	152.8	0
0.48	158.3	0
0.485	182.2	0
0.49	151.9	0
0.495	152.4	0
0.5	160.1	0
0.255	135.1	1
0.26	137.3	1
0.265	144.3	1
0.27	143.8	1
0.275	151.8	1
0.28	153.7	1
0.285	152.6	1
0.29	148.9	1
0.295	158.8	1
0.3	149.5	1
0.305	144.3	1
0.31	144.5	1
0.315	148	1
0.32	144.5	1
0.325	142	1
0.33	135	1
0.335	138.8	1
0.34	134.6	1
0.345	161.7	1
0.35	143.3	1
0.355	147.1	1
0.36	148.5	1
0.365	152.8	1
0.37	151.1	1
0.375	150.9	1
0.38	146.9	1
0.385	147.9	1

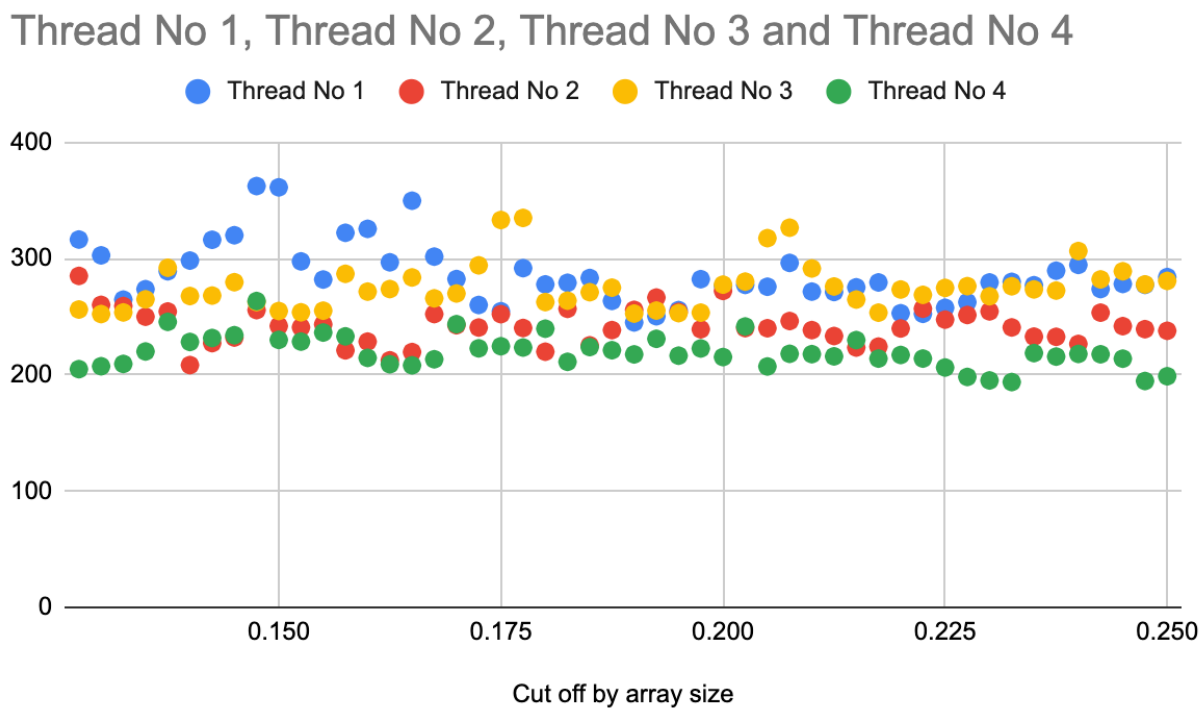
0.39	148.4	1
0.395	144.9	1
0.4	147.6	1
0.405	147.9	1
0.41	137.5	1
0.415	146.7	1
0.42	138.2	1
0.425	135.1	1
0.43	136.9	1
0.435	140	1
0.44	138.6	1
0.445	144.1	1
0.45	149.5	1
0.455	146.5	1
0.46	141.2	1
0.465	140.9	1
0.47	176.2	1
0.475	153.6	1
0.48	157.1	1
0.485	155.3	1
0.49	150.8	1
0.495	157.6	1
0.5	156	1
0.255	108.8	2
0.26	110.4	2
0.265	107.8	2
0.27	125.5	2
0.275	114	2
0.28	128.9	2
0.285	147.8	2
0.29	125.3	2
0.295	128.3	2
0.3	124.7	2
0.305	121.1	2
0.31	124.2	2
0.315	148.6	2
0.32	117.2	2
0.325	119.7	2
0.33	119.4	2
0.335	122.2	2

0.34	136.7	2
0.345	121.6	2
0.35	120.1	2
0.355	122.6	2
0.36	133.3	2
0.365	130	2
0.37	131.6	2
0.375	130.7	2
0.38	156.3	2
0.385	119.8	2
0.39	125.9	2
0.395	126	2
0.4	130.4	2
0.405	129.3	2
0.41	132.8	2
0.415	132.2	2
0.42	132.5	2
0.425	151.8	2
0.43	128.8	2
0.435	119.1	2
0.44	124.2	2
0.445	132.8	2
0.45	153.7	2
0.455	124.5	2
0.46	124.7	2
0.465	116.9	2
0.47	118.7	2
0.475	120.8	2
0.48	125.8	2
0.485	122.6	2
0.49	118.6	2
0.495	119.8	2
0.5	117.7	2
0.255	80.8	3
0.26	83.4	3
0.265	101.2	3
0.27	77.2	3
0.275	80.3	3
0.28	81	3
0.285	99	3

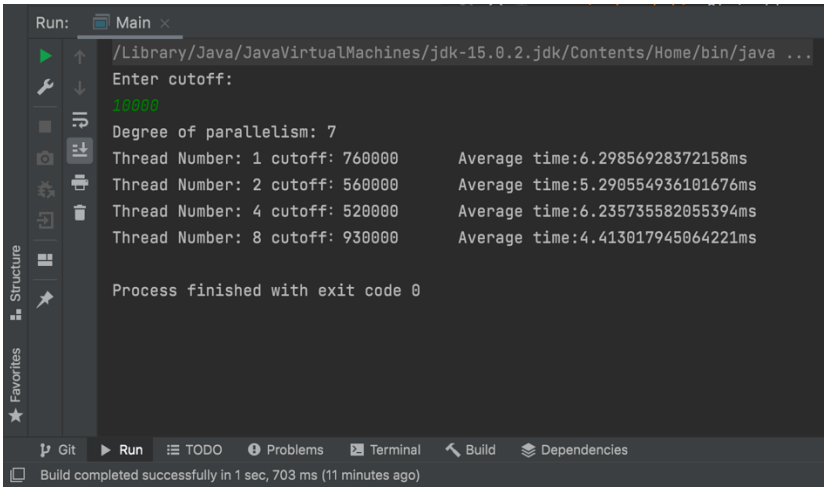
0.29	85.5	3
0.295	81.7	3
0.3	84.7	3
0.305	83.4	3
0.31	81.8	3
0.315	83.3	3
0.32	84.3	3
0.325	82.2	3
0.33	85.5	3
0.335	99.7	3
0.34	74.4	3
0.345	82.3	3
0.35	85.6	3
0.355	81	3
0.36	81.4	3
0.365	80.5	3
0.37	82.4	3
0.375	83.6	3
0.38	82.7	3
0.385	82.9	3
0.39	84	3
0.395	93.9	3
0.4	146.1	3
0.405	80.3	3
0.41	79.3	3
0.415	81.9	3
0.42	88.2	3
0.425	89.4	3
0.43	83.2	3
0.435	105.1	3
0.44	111	3
0.445	98.4	3
0.45	110.1	3
0.455	117.8	3
0.46	96.5	3
0.465	97.9	3
0.47	122.1	3
0.475	84.9	3
0.48	88.8	3
0.485	86.9	3

0.49	98.1	3
0.495	92.5	3
0.5	80.8	3

For array size 4000000, with 4 threads graphs performance :-



Output result:-



Result csv:-

Cut off by array size	Time (ms)	Threads
0.1275	316.4	0
0.13	302.8	0
0.1325	264.7	0
0.135	273.7	0
0.1375	289.1	0
0.14	298.3	0
0.1425	316.2	0
0.145	320.1	0
0.1475	362.4	0
0.15	361.3	0
0.1525	297.6	0
0.155	282	0
0.1575	322.2	0
0.16	325.6	0
0.1625	296.8	0
0.165	349.8	0
0.1675	301.7	0
0.17	282.4	0
0.1725	260.1	0
0.175	254.7	0
0.1775	291.8	0
0.18	277.8	0
0.1825	279.2	0
0.185	283.3	0
0.1875	263.6	0
0.19	245.1	0
0.1925	250.4	0
0.195	255.7	0
0.1975	282.3	0
0.2	276.5	0
0.2025	277.4	0
0.205	275.8	0
0.2075	296.3	0
0.21	271.6	0
0.2125	271.3	0
0.215	275.3	0
0.2175	279.5	0

0.22	253	0
0.2225	252.5	0
0.225	257.7	0
0.2275	262.5	0
0.23	279.5	0
0.2325	280	0
0.235	277.1	0
0.2375	289.6	0
0.24	294.6	0
0.2425	273.8	0
0.245	278.2	0
0.2475	277.4	0
0.25	284.1	0
0.1275	285.1	1
0.13	260.2	1
0.1325	259.2	1
0.135	250	1
0.1375	254.5	1
0.14	208.4	1
0.1425	227.3	1
0.145	232.1	1
0.1475	255.7	1
0.15	242	1
0.1525	240.9	1
0.155	243.4	1
0.1575	221.1	1
0.16	228.6	1
0.1625	212.5	1
0.165	219.6	1
0.1675	252.3	1
0.17	242.7	1
0.1725	240.7	1
0.175	252.2	1
0.1775	240.3	1
0.18	219.9	1
0.1825	256.8	1
0.185	225.4	1
0.1875	238.5	1
0.19	255.9	1
0.1925	266.5	1

0.195	254.1	1
0.1975	239.2	1
0.2	272.2	1
0.2025	240.2	1
0.205	240.1	1
0.2075	246.3	1
0.21	238.5	1
0.2125	233.4	1
0.215	223.5	1
0.2175	224.3	1
0.22	239.9	1
0.2225	256.9	1
0.225	247.5	1
0.2275	251.5	1
0.23	254.6	1
0.2325	240.8	1
0.235	232.9	1
0.2375	232.7	1
0.24	226.6	1
0.2425	253.5	1
0.245	241.9	1
0.2475	239.3	1
0.25	237.9	1
0.1275	256.2	2
0.13	252.3	2
0.1325	253.7	2
0.135	265	2
0.1375	292.1	2
0.14	267.8	2
0.1425	268.2	2
0.145	279.7	2
0.1475	262.6	2
0.15	254.9	2
0.1525	253.8	2
0.155	255.2	2
0.1575	286.9	2
0.16	271.6	2
0.1625	273.8	2
0.165	283.8	2
0.1675	265.8	2

0.17	270	2
0.1725	294.2	2
0.175	333.2	2
0.1775	335	2
0.18	262.6	2
0.1825	263.8	2
0.185	271.1	2
0.1875	275	2
0.19	252.7	2
0.1925	255.3	2
0.195	253.1	2
0.1975	253.5	2
0.2	277.6	2
0.2025	280.2	2
0.205	317.6	2
0.2075	326.6	2
0.21	291.5	2
0.2125	276	2
0.215	264.8	2
0.2175	253.5	2
0.22	273.4	2
0.2225	268.8	2
0.225	274.8	2
0.2275	276.3	2
0.23	267.6	2
0.2325	276.2	2
0.235	273.5	2
0.2375	272.5	2
0.24	306.5	2
0.2425	282.1	2
0.245	289.1	2
0.2475	277.8	2
0.25	280.8	2
0.1275	204.9	3
0.13	207.4	3
0.1325	209.4	3
0.135	220.1	3
0.1375	245.6	3
0.14	228.3	3
0.1425	231.7	3

0.145	234.1	3
0.1475	263.5	3
0.15	230.1	3
0.1525	228.6	3
0.155	236.4	3
0.1575	233.1	3
0.16	214.6	3
0.1625	209.1	3
0.165	208.2	3
0.1675	213.4	3
0.17	243.6	3
0.1725	222.8	3
0.175	224.6	3
0.1775	223.5	3
0.18	239.7	3
0.1825	211.2	3
0.185	223.9	3
0.1875	221.2	3
0.19	217.6	3
0.1925	231	3
0.195	216.5	3
0.1975	222.7	3
0.2	215.3	3
0.2025	241.6	3
0.205	207.2	3
0.2075	218.1	3
0.21	217.8	3
0.2125	215.9	3
0.215	229.9	3
0.2175	214	3
0.22	217.1	3
0.2225	214	3
0.225	206.3	3
0.2275	198.2	3
0.23	195.2	3
0.2325	193.8	3
0.235	218.7	3
0.2375	215.7	3
0.24	218	3
0.2425	217.7	3

0.245	213.9	3
0.2475	194.7	3
0.25	198.8	3

Conclusion:

From the graphs above we can say that, sorting with 4 threads is the fastest one and sorting with one thread is slowest.

This means that if we implement parallel sorting then performance of sorting program is improved as the number of threads increases and work is performed parallelly.