

Tushar Kurhekar

NUID - 001521707

Program Structures & Algorithms INFO6205

Assignment 4

Task Performed :

Task (List down the tasks performed in the Assignment)

- 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.

Report Outcomes :

- It has been observed for lower cutoff values that System sort is more efficient than the parallel sort implemented which we observed by noting the timestamp.
- The ideal cutoff can be concluded as little more than 10 % of the array element which generates threads as well as takes less time in sorting
- Sorting becomes efficient as we increase the cut-off
- A good cutoff is array size/thread number, when the cutoff is similar to this number, the performance becomes flattened and better.
- The total number of cores of my laptop is 4, an ideal thread number is 4, when the thread number is larger than 4, the performance is similar.

Main function :

```

1  // Sorting
2  // ForkJoinPool
3  // dynamicP
4  // equable
5  // functions
6  // graphs
7  // greedy
8  // hashtable
9  // lab_1
10 // life
11 // pq
12 // randomw
13 // reduction
14 // runLength
15 // sort
16 // classic
17 // counti
18 // elemer
19 // hashC
20 // linearit
21 // par
22 // Main
23 // ParS
24 // BaseHi
25 // Generi
26 // Generi
27 // Generi
28 // Helper
29 // Helper
30 // Instru
31 // Sort
32 // SortEx
33 // SortWi
34 // symbolTa
35 // threesum
36 // Array
37 // Build
38 // Dependencies
39 // Event Log
40 //
41 //
42 //
43 //
44 //
45 //
46 //
47 //
48 //
49 //
50 //
51 //
52 //
53 //
54 //
55 //
56 //
57 //
58 //
59 //
60 //
61 //
62 //
63 //
64 //
65 //
66 //
67 //
68 //
69 //
70 //
71 //
72 //
73 //
74 //
75 //
76 //
77 //
78 //
79 //
80 //
81 //
82 //
83 //
84 //
85 //
86 //
87 //
88 //
89 //
90 //
91 //
92 //
93 //
94 //
95 //
96 //
97 //
98 //
99 //
100 //
101 //
102 //
103 //
104 //
105 //
106 //
107 //
108 //
109 //
110 //
111 //
112 //
113 //
114 //
115 //
116 //
117 //
118 //
119 //
120 //
121 //
122 //
123 //
124 //
125 //
126 //
127 //
128 //
129 //
130 //
131 //
132 //
133 //
134 //
135 //
136 //
137 //
138 //
139 //
140 //
141 //
142 //
143 //
144 //
145 //
146 //
147 //
148 //
149 //
150 //
151 //
152 //
153 //
154 //
155 //
156 //
157 //
158 //
159 //
160 //
161 //
162 //
163 //
164 //
165 //
166 //
167 //
168 //
169 //
170 //
171 //
172 //
173 //
174 //
175 //
176 //
177 //
178 //
179 //
180 //
181 //
182 //
183 //
184 //
185 //
186 //
187 //
188 //
189 //
190 //
191 //
192 //
193 //
194 //
195 //
196 //
197 //
198 //
199 //
200 //
201 //
202 //
203 //
204 //
205 //
206 //
207 //
208 //
209 //
210 //
211 //
212 //
213 //
214 //
215 //
216 //
217 //
218 //
219 //
220 //
221 //
222 //
223 //
224 //
225 //
226 //
227 //
228 //
229 //
230 //
231 //
232 //
233 //
234 //
235 //
236 //
237 //
238 //
239 //
240 //
241 //
242 //
243 //
244 //
245 //
246 //
247 //
248 //
249 //
250 //
251 //
252 //
253 //
254 //
255 //
256 //
257 //
258 //
259 //
260 //
261 //
262 //
263 //
264 //
265 //
266 //
267 //
268 //
269 //
270 //
271 //
272 //
273 //
274 //
275 //
276 //
277 //
278 //
279 //
280 //
281 //
282 //
283 //
284 //
285 //
286 //
287 //
288 //
289 //
290 //
291 //
292 //
293 //
294 //
295 //
296 //
297 //
298 //
299 //
300 //
301 //
302 //
303 //
304 //
305 //
306 //
307 //
308 //
309 //
310 //
311 //
312 //
313 //
314 //
315 //
316 //
317 //
318 //
319 //
320 //
321 //
322 //
323 //
324 //
325 //
326 //
327 //
328 //
329 //
330 //
331 //
332 //
333 //
334 //
335 //
336 //
337 //
338 //
339 //
340 //
341 //
342 //
343 //
344 //
345 //
346 //
347 //
348 //
349 //
350 //
351 //
352 //
353 //
354 //
355 //
356 //
357 //
358 //
359 //
360 //
361 //
362 //
363 //
364 //
365 //
366 //
367 //
368 //
369 //
370 //
371 //
372 //
373 //
374 //
375 //
376 //
377 //
378 //
379 //
380 //
381 //
382 //
383 //
384 //
385 //
386 //
387 //
388 //
389 //
390 //
391 //
392 //
393 //
394 //
395 //
396 //
397 //
398 //
399 //
400 //
401 //
402 //
403 //
404 //
405 //
406 //
407 //
408 //
409 //
410 //
411 //
412 //
413 //
414 //
415 //
416 //
417 //
418 //
419 //
420 //
421 //
422 //
423 //
424 //
425 //
426 //
427 //
428 //
429 //
430 //
431 //
432 //
433 //
434 //
435 //
436 //
437 //
438 //
439 //
440 //
441 //
442 //
443 //
444 //
445 //
446 //
447 //
448 //
449 //
450 //
451 //
452 //
453 //
454 //
455 //
456 //
457 //
458 //
459 //
460 //
461 //
462 //
463 //
464 //
465 //
466 //
467 //
468 //
469 //
470 //
471 //
472 //
473 //
474 //
475 //
476 //
477 //
478 //
479 //
480 //
481 //
482 //
483 //
484 //
485 //
486 //
487 //
488 //
489 //
490 //
491 //
492 //
493 //
494 //
495 //
496 //
497 //
498 //
499 //
500 //
501 //
502 //
503 //
504 //
505 //
506 //
507 //
508 //
509 //
510 //
511 //
512 //
513 //
514 //
515 //
516 //
517 //
518 //
519 //
520 //
521 //
522 //
523 //
524 //
525 //
526 //
527 //
528 //
529 //
530 //
531 //
532 //
533 //
534 //
535 //
536 //
537 //
538 //
539 //
540 //
541 //
542 //
543 //
544 //
545 //
546 //
547 //
548 //
549 //
550 //
551 //
552 //
553 //
554 //
555 //
556 //
557 //
558 //
559 //
560 //
561 //
562 //
563 //
564 //
565 //
566 //
567 //
568 //
569 //
570 //
571 //
572 //
573 //
574 //
575 //
576 //
577 //
578 //
579 //
580 //
581 //
582 //
583 //
584 //
585 //
586 //
587 //
588 //
589 //
590 //
591 //
592 //
593 //
594 //
595 //
596 //
597 //
598 //
599 //
600 //
601 //
602 //
603 //
604 //
605 //
606 //
607 //
608 //
609 //
610 //
611 //
612 //
613 //
614 //
615 //
616 //
617 //
618 //
619 //
620 //
621 //
622 //
623 //
624 //
625 //
626 //
627 //
628 //
629 //
630 //
631 //
632 //
633 //
634 //
635 //
636 //
637 //
638 //
639 //
640 //
641 //
642 //
643 //
644 //
645 //
646 //
647 //
648 //
649 //
650 //
651 //
652 //
653 //
654 //
655 //
656 //
657 //
658 //
659 //
660 //
661 //
662 //
663 //
664 //
665 //
666 //
667 //
668 //
669 //
670 //
671 //
672 //
673 //
674 //
675 //
676 //
677 //
678 //
679 //
680 //
681 //
682 //
683 //
684 //
685 //
686 //
687 //
688 //
689 //
690 //
691 //
692 //
693 //
694 //
695 //
696 //
697 //
698 //
699 //
700 //
701 //
702 //
703 //
704 //
705 //
706 //
707 //
708 //
709 //
710 //
711 //
712 //
713 //
714 //
715 //
716 //
717 //
718 //
719 //
720 //
721 //
722 //
723 //
724 //
725 //
726 //
727 //
728 //
729 //
730 //
731 //
732 //
733 //
734 //
735 //
736 //
737 //
738 //
739 //
740 //
741 //
742 //
743 //
744 //
745 //
746 //
747 //
748 //
749 //
750 //
751 //
752 //
753 //
754 //
755 //
756 //
757 //
758 //
759 //
760 //
761 //
762 //
763 //
764 //
765 //
766 //
767 //
768 //
769 //
770 //
771 //
772 //
773 //
774 //
775 //
776 //
777 //
778 //
779 //
780 //
781 //
782 //
783 //
784 //
785 //
786 //
787 //
788 //
789 //
790 //
791 //
792 //
793 //
794 //
795 //
796 //
797 //
798 //
799 //
800 //
801 //
802 //
803 //
804 //
805 //
80
```

partSort()

The screenshot shows an IDE window titled "INFO6205 - ParSort.java". The file explorer on the left displays a project structure with a "par" directory selected. The main editor shows the "ParSort.java" file with the following code:

```

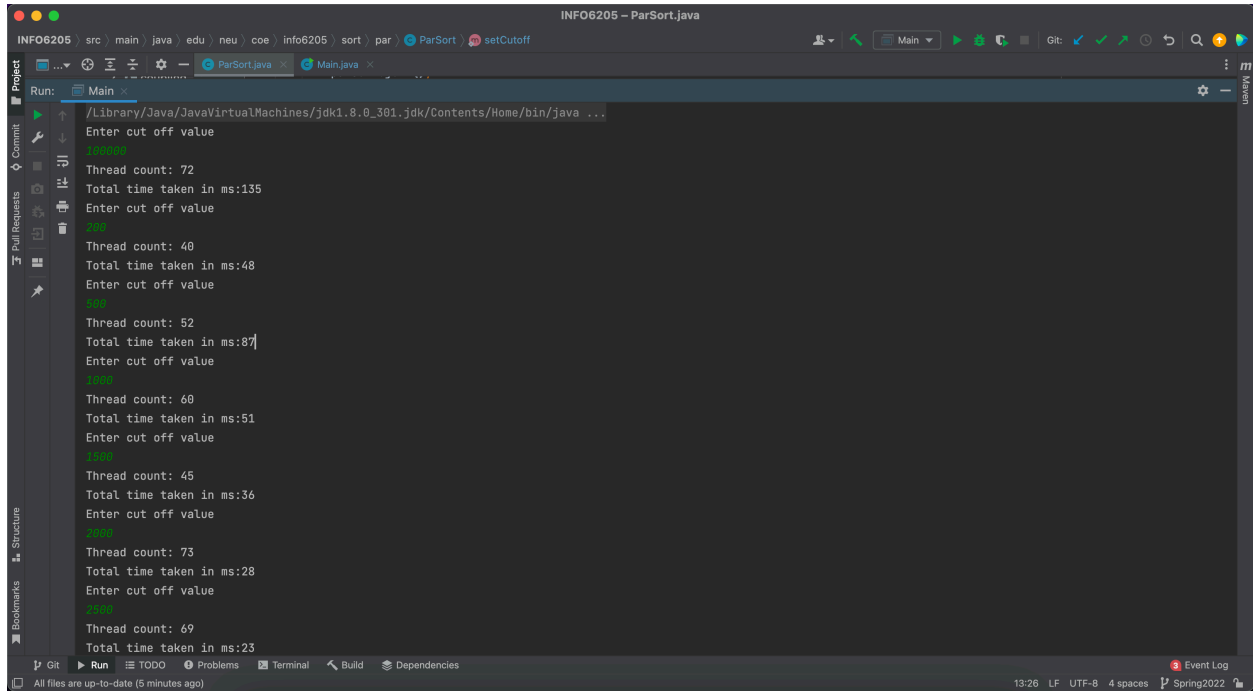
public static void sort(int[] array, int from, int to, int cutoff) {
    if (to - from < ParSort.cutoff) Arrays.sort(array, from, to);
    else {
        // FIXME next few lines should be removed from public repo.
        CompletableFuture<int[]> p1 = parsort(array, from, to - (to - from) / 2); // TO IMPLEMENT
        CompletableFuture<int[]> p2 = parsort(array, from + (to - from) / 2, to); // TO IMPLEMENT
        CompletableFuture<int[]> p3 = p1.thenCombine(p2, (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;
        });
        p3.whenComplete((result, throwable) -> {
            //System.arraycopy(result, 0, array, from, result.length);
        });
    }
}

```

The status bar at the bottom shows "Run" and "Terminal" tabs. The bottom right corner displays "20:47 1F UTF-8 4 spaces" and "Spring 2022".

Results for the conclusion :

Output Terminal :

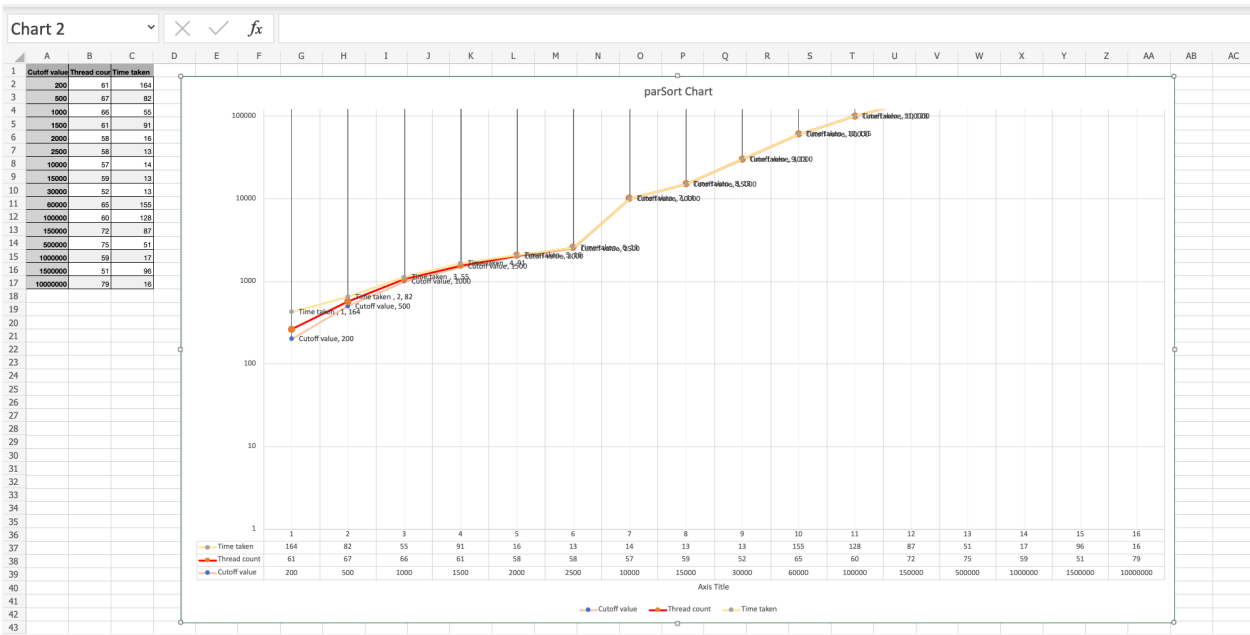


```
INFO6205 - ParSort.java
INFO6205 \src \main \java \edu \neu \coe \info6205 \sort \par \ParSort \setCutoff
Run: Main
/Library/Java/JavaVirtualMachines/jdk1.8.0_301.jdk/Contents/Home/bin/java ...
Enter cut off value
1000000
Thread count: 72
Total time taken in ms:135
Enter cut off value
1000000
Thread count: 40
Total time taken in ms:48
Enter cut off value
1000000
Thread count: 52
Total time taken in ms:87
Enter cut off value
1000000
Thread count: 60
Total time taken in ms:51
Enter cut off value
1000000
Thread count: 45
Total time taken in ms:36
Enter cut off value
1000000
Thread count: 73
Total time taken in ms:28
Enter cut off value
1000000
Thread count: 69
Total time taken in ms:23
```

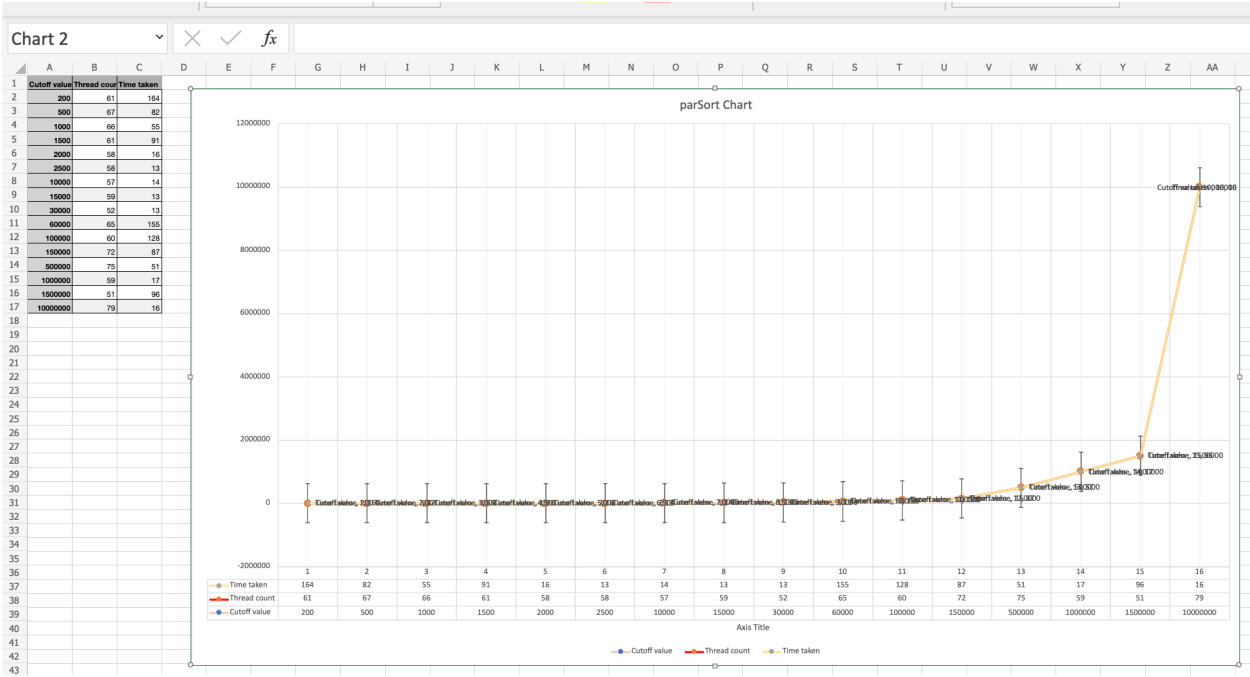
Table is to thread count is to time taken index

Cutoff value	Thread count (parallelism)	Time taken
200	61	164
500	67	82
1000	66	55
1500	61	91
2000	58	16
2500	58	13
10000	57	14
15000	59	13
30000	52	13
60000	65	155
100000	60	128
150000	72	87
500000	75	51
1000000	59	17
1500000	51	96
10000000	79	16

Logarithmic Scale :



Actual Scale :

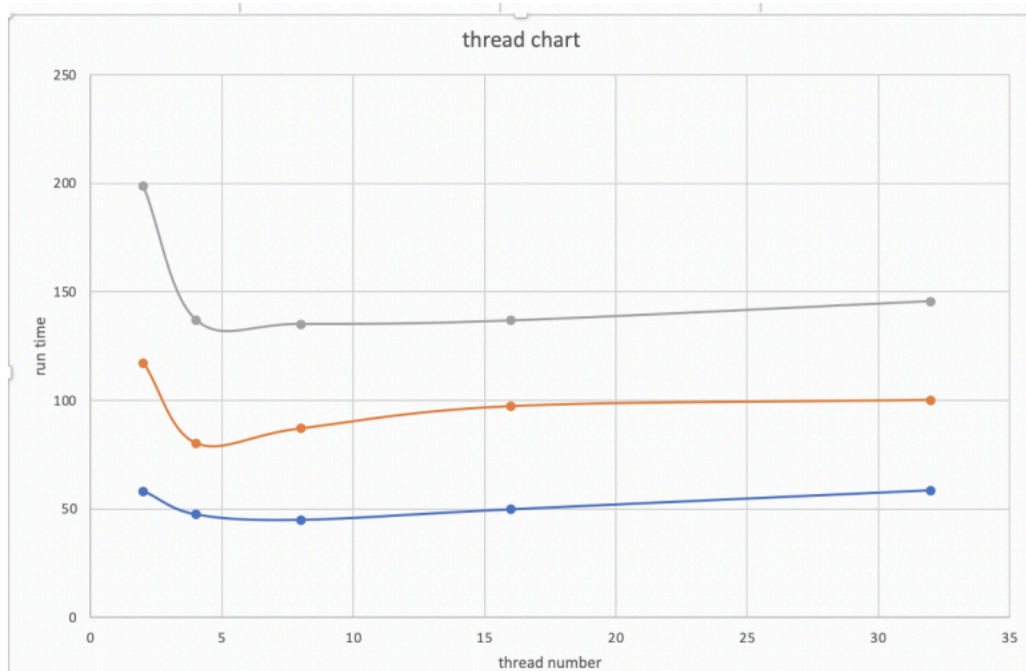


Extended Conclusion :

From the above charts, we can conclude that a relatively better

CUTOFFF = array size / thread number. (middle value)

I have seen the time taken flatten by the cutoff values



Generalized observation of runtime and number of threads