

Screenshots:

```
phiro@phiro: ~/Documents/CSC410/A3_2
phiro@phiro:~/Documents/CSC410/A3_2$ nano p_integrate.c
phiro@phiro:~/Documents/CSC410/A3_2$ gcc p_integrate.c -o pi
phiro@phiro:~/Documents/CSC410/A3_2$ ./pi
Result of numerical integration: 3.141593
phiro@phiro:~/Documents/CSC410/A3_2$
```

```
phiro@phiro: ~/Documents/CSC410/A3_2
phiro@phiro:~/Documents/CSC410/A3_2$ nano p_merge.c
phiro@phiro:~/Documents/CSC410/A3_2$ gcc p_merge.c p_merge.h main_merge.c -o mpp
phiro@phiro:~/Documents/CSC410/A3_2$ ./mpp
Enter elements: Sequential Merge Sort Time: 33.589 ms
0 1 2 2 3 3 3 3 4 4 8 8 9 10 11 12 13 13 13 14 14 17 17 19 20 20 20 23 24 25 2
5 26 29 29 30 30 31 31 33 33 34 35 36 36 37 39 40 41 42 42 43 44 44 48 48 49 5
0 50 51 52 52 56 56 58 60 62 62 62 63 64 64 66 68 68 69 69 70 71 73 74 75 75 7
6 77 78 79 81 81 81 83 83 84 84 85 86 86 87 88 90 90 91 91 93 94 95 95 96 96 9
6 97 103 104 106 107 107 108 108 113 114 115 115 115 117 118 118 119 120 121 1
23 123 124 124 124 129 131 131 132 132 132 135 136 137 138 138 139 141 143 144
147 148 150 152 152 153 154 155 155 156 158 158 159 160 160 160 161 163 164 1
66 166 168 168 169 169 171 172 172 174 176 176 176 177 177 178 178 178 179 180
181 181 181 181 184 185 185 186 188 189 189 189 189 191 192 193 194 194 194 1
94 195 195 196 196 198 198 200 201 202 202 204 204 205 205 206 206 208 209 210
211 211 212 212 212 213 214 214 215 215 216 216 216 217 219 220 223 224 224 2
24 224 225 225 226 227 228 229 229 234 234 234 237 239 242 242 243 243 244 245
245 246 247 247 248 250 250 251 252 253 253 254 254 255 256 257 257 258 259 2
59 260 262 262 265 266 266 269 269 269 270 271 272 272 275 276 276 277 277 278
278 279 279 281 281 283 285 286 287 287 287 288 288 290 294 295 295 296 2
96 297 298 299 300 301 301 302 302 303 303 303 304 305 306 307 308 308 311
313 314 316 316 317 317 321 321 322 322 323 327 329 331 333 334 335 335 335 336 3
39 340 340 340 341 342 343 343 344 345 346 346 346 348 348 350 352 354 356 359
360 361 361 361 362 363 364 366 371 372 373 373 374 374 375 375 376 378 379 3
80 383 384 385 386 387 387 389 389 390 390 391 391 395 396 397 398 399 400 403
405 406 408 408 409 410 411 411 412 414 415 416 418 419 421 421 421 422 422 4
22 424 425 426 426 426 428 428 429 430 432 434 434 434 435 435 436 438 440 441
441 443 443 444 445 446 447 447 449 449 451 451 453 453 454 454 456 458 458 4
61 461 463 463 464 466 467 469 469 470 470 471 472 472 473 473 474 475 475 478
478 478 478 479 482 482 483 483 484 486 486 487 487 488 488 489 489 489 490 4
91 492 493 494 494 494 497 499 501 501 505 506 507 510 510 511 513 513 514 514
515 515 515 515 517 518 519 519 519 521 523 524 527 527 527 531 532 535 535 5
35 536 536 536 538 539 539 540 541 542 543 543 545 545 546 546 548 549 549 549
550 550 551 551 553 554 557 559 559 559 560 561 562 563 563 564 566 568 569 5
69 573 573 574 576 576 577 579 581 583 584 585 585 586 588 593 594 594 596 597
598 599 600 600 601 601 605 606 607 608 608 611 612 612 613 613 615 615 616 6
17 618 619 622 622 625 625 625 628 629 629 631 631 632 632 636 636 636 637 638 639
639 640 641 641 644 646 648 649 649 650 651 651 652 653 653 655 655 657 658 6
60 664 665 667 668 669 669 669 672 675 675 676 676 676 677 677 677 679 680 681
681 682 682 684 684 686 686 688 689 690 691 692 692 693 694 696 699 701 701 7
02 702 703 704 705 706 706 706 707 707 708 712 712 713 713 714 714 714 715 718
719 721 721 721 721 722 723 723 724 725 725 725 726 727 727 729 731 732 734 7
35 735 736 736 736 736 739 739 741 742 745 745 746 746 746 748 749 750 752 753
754 755 757 758 759 760 760 760 760 762 763 763 766 767 767 768 769 769 770 7
```

```

phiro@phiro: ~/Documents/CSC410/A3_2
phiro@phiro:~/Documents/CSC410/A3_2$ ls
main_merge.c  p_bubble_2.c  p_integrateOutput.png  p_merge.h.gch
mpp           pi            p_merge.c
p_bubble_1.c  p_integrate.c  p_merge.h
phiro@phiro:~/Documents/CSC410/A3_2$ nano p_bubble_1.c
phiro@phiro:~/Documents/CSC410/A3_2$ gcc p_bubble_1.c -o pb1
phiro@phiro:~/Documents/CSC410/A3_2$ ./pb1
Unsorted: 89383 30886 92777 36915 47793 38335 85386 60492 16649 41421 2362 900
27 68690 20059 97763 13926 80540 83426 89172 55736 5211 95368 2567 56429 65782
21530 22862 65123 74067 3135 13929 79802 34022 23058 33069 98167 61393 18456
75011 78042 76229 77373 84421 44919 13784 98537 75198 94324 98315 64370 66413
3526 76091 68980 59956 41873 6862 99170 6996 97281 2305 20925 77084 36327 6033
6 26505 50846 21729 61313 25857 16124 53895 19582 545 98814 33367 15434 90364
44043 13750 71087 26808 17276 47178 95788 93584 5403 2651 92754 12399 99932 95
060 49676 93368 47739 10012 36226 98586 48094 97539 40795 80570 51434 60378 97
467 66601 10097 12902 73317 70492 26652 60756 97301 60280 24286 9441 53865 296
89 28444 46619 58440 44729 58031 8117 38097 5771 34481 90675 20709 98927 4567
77856 79497 72353 54586 76965 55306 64683 6219 28624 51528 32871 5732 48829 95
03 30019 58270 63368 59708 86715 26340 18149 47796 723 42618 2245 22846 93451
92921 43555 92379 97488 37764 88228 69841 92350 65193 41500 57034 87764 70124
24914 36987 75856 73743 46491 22227 48365 9859 81936 51432 52551 16437 99228 5
3275 75407 1474 76121 68858 94395 36029 61237 8235 73793 65818 94428 66143 310
11 35928 39529 18776 22404 64443 55763 14613 54538 18606 36840 2904 44818 3512
8 70688 97369 67917 69917 66996 43324 87743 59470 12183 98490 95499 89772 6725
85644 55590 17505 68139 2954 69786 7669 38082 8542 88464 10197 39507 59355 28
804 76348 78611 73622 27828 49299 87343 95746 35568 54340 55422 23311 13810 67
605 21801 25661 73730 44878 11305 29320 78736 79444 48626 48522 3465 86708 734
16 8282 13258 12924 67637 42062 5624 62600 32036 33452 11899 19379 45550 47468
90071 973 87131 3881 84930 8933 45894 58660 70163 57199 87981 48899 52996 529
59 13773 72813 39668 87190 81095 52926 16466 65084 11340 22090 27684 43376 555
42 55936 79107 17445 19756 69179 18418 6887 89412 3348 32172 51659 62009 2336
25210 66342 67587 78206 19301 97713 67372 75321 1255 64819 44599 17721 29904 5
5939 39811 73940 15667 11705 46228 11127 29150 65984 96658 63920 89224 2422 67
269 21396 54081 45630 40084 79292 11972 7672 73850 47625 5385 41222 39299 6640
6042 83898 40713 52298 56190 80524 42590 88209 8581 88819 99336 37732 71155 9
5994 18004 60379 14769 85273 81776 68850 47255 21860 48142 75579 45884 21993 2
3205 67621 79567 62504 90613 1961 62754 31326 54259 18944 28202 13202 23506 36
784 2021 22842 90868 89528 35189 8872 49908 49958 10498 48036 18808 57753 8624
8 83303 33333 32133 21648 72890 99754 17567 51746 90368 19529 14500 38046 7378
8 49797 66249 86990 73303 3033 5363 12497 10253 94892 47686 19125 61152 13996
45975 9188 49157 3729 95436 32460 53414 43921 70460 26304 60028 88027 78050 66
748 7556 8902 4794 97697 58699 71043 1039 32002 90428 6403 44500 681 17647 853
8 36159 95151 22535 82134 4339 71692 2215 16127 20504 55629 60049 90964 98285
36429 95343 76335 3177 2900 85238 7971 16949 60289 95367 17988 92292 85795 407

```

Task 3:

Parallel Bubble Sort 1 (Segment-Based):

This program divides the array into chunks, and each thread performs bubble passes locally within its segment, synchronizing with barriers after each pass. It is simple to implement and modifies the classic bubble sort with minimal changes. However, heavy synchronization overhead limits speedups, especially due to dependencies at chunk boundaries. This approach

suffers from limited parallelism because the outer loop remains sequential in effect. Overall, its scalability is poor, and performance gains over sequential are minimal for large arrays.

Parallel Bubble Sort 2 (Odd-Even / Brick Sort):

This another version of bubble sort program uses the odd-even transposition method, alternating between even-indexed and odd-indexed adjacent swaps in phases that can safely run in parallel. It achieves better concurrency by avoiding overlapping data access within each phase and synchronizing only at phase boundaries. The algorithm is naturally suited to parallelization with predictable synchronization points and fewer dependencies. Despite this, the total number of phases remains $O(n)O(n)$, so the quadratic time complexity persists. It outperforms the first approach but still far lags behind more advanced parallel sorts for large datasets.

Both approaches parallelize bubble sort but remain limited by bubble sort's inherent quadratic time and synchronization overhead, making them mainly educational tools rather than practical solutions for large or performance-critical datasets.

Task 4:

Program	Sequential	1 Thread	2 Threads	3 Threads	4 Threads
Num_Integrate	14.027	14.027	7.374	7.420	7.876
Merge_approach_1	0.074	0.074	0.063	0.056	0.062
Merge_approach_2	X	0.071	0.061	0.061	0.064
Merge_approach_3	X	0.362	0.342	0.377	0.390
Bubble_1	48.605	48.415	37.445	32.574	31.933
Bubble_2	X	17.831	12.240	16.870	14.036

Task 5:

I noticed that more threads increase the programs' performance but there are not a huge difference in the performance between the programs.

When parallelizing programs using pthreads, it is important to carefully divide the work so that each thread has its own portion of data to process, which reduces contention and race conditions. Synchronization mechanisms like mutexes and barriers must be used to ensure that shared data is not accessed simultaneously, preventing errors and guaranteeing correctness. Balancing the number of threads with the hardware capabilities is key, as too many threads can

cause overhead that outweighs the benefits of parallelism. Another consideration is minimizing the overhead of thread creation and joining, so tasks should be large enough to justify the parallelization cost. Overall, pthreads provide great control and flexibility but require careful programming to avoid common multithreading pitfalls like deadlocks and excessive synchronization.

Pthreads offer direct access to low-level thread management, allowing efficient use of system resources and potentially significant speedups on multicore processors. They are portable across POSIX systems and integrate well with C and C++ programs. The main challenges include complexity in writing and debugging concurrent code, as timing-dependent bugs can be very subtle. Additionally, programmers must explicitly manage all synchronization and data sharing, which increases the chance of errors. Despite this, mastering pthreads is valuable for understanding the foundations of parallel programming and producing high-performance programs when done carefully.