

0Design and Analysis of Algorithms Mini-Project Report

Design and Development of a Merge Sort Tool in Python

Name: Parth Ingle (41227)

Mukund Karwa (41234)

Soham Lagad (41241)

Batch: Q2

Title

Design and Development of a Tool for Digital Forensics of Images.

Software and Hardware Requirements

- **Operating System:** Windows/Linux/macOS
- **Programming Language:** Python 3.x
- **Libraries/Modules:** math, multiprocessing, random, sys, time
- **Processor:** Intel i3 or higher
- **RAM:** 4 GB minimum (8 GB recommended)
- **Storage:** 1 GB free space
- **Display:** 1024×768 resolution or higher

Theory

Merge Sort Algorithm

Merge Sort is a divide and conquer sorting algorithm. It recursively splits the list into halves, sorts each half, and then merges the sorted halves to produce the final sorted list.

- Time Complexity: $O(n \log n)$
- Space Complexity: $O(n)$
- Stable Sort: Yes

Steps:

1. Divide: Split the list into two halves.
2. Conquer: Recursively sort each half.
3. Combine: Merge the sorted halves into a single sorted list.

Algorithms Used

1. Standard Merge Sort
 - Recursively splits the list and merges sorted sublists.
 - Suitable for small to medium datasets.

2. Multithreaded Merge Sort

- Utilizes Python's multiprocessing to sort sublists in parallel.
- Improves performance on large datasets by leveraging multiple CPU cores.

Conceptual Explanation of Solution

- Merge Function: Combines two sorted lists into one sorted list.
- Recursive Merge Sort: Splits the list, sorts each half, and merges them.
- Multithreaded Merge Sort: Divides the list into chunks, sorts each chunk in parallel, and merges the results.

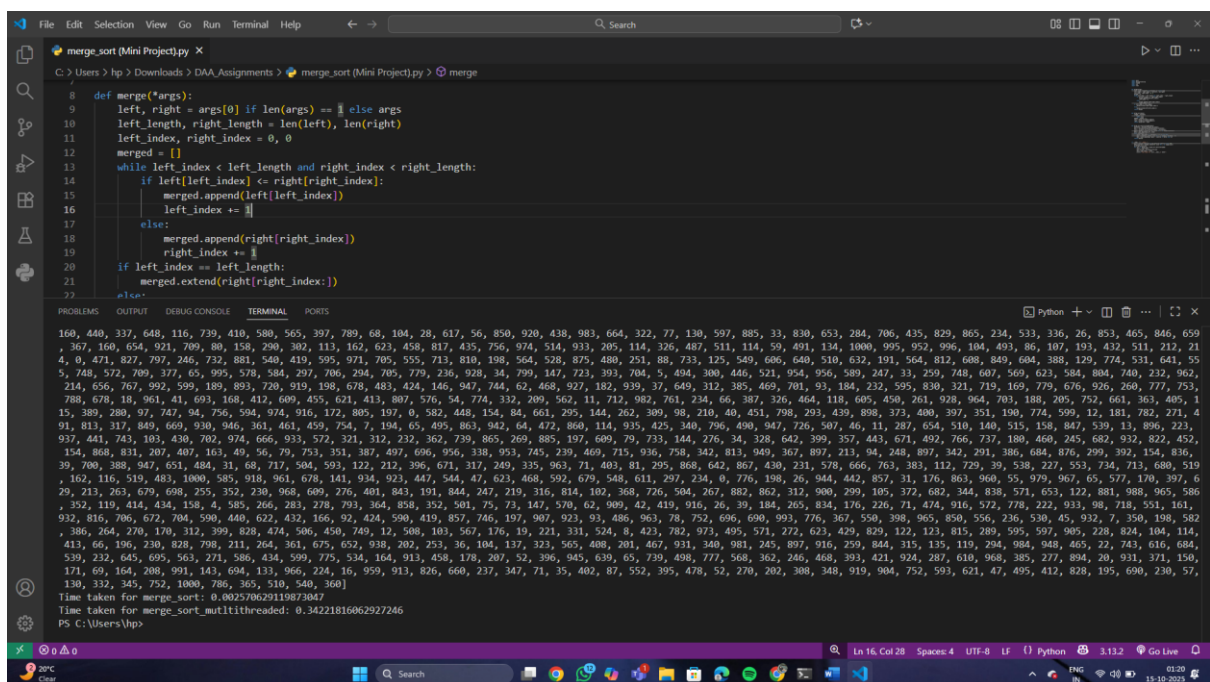
Screenshot of Output (Describe)

- The program prints the unsorted and sorted lists.
- It displays the time taken for both standard and multithreaded merge sort.

Result

- Standard Merge Sort: Efficient for moderate data sizes.
- Multithreaded Merge Sort: Faster for large datasets due to parallel processing.
- Output: Sorted list and timing for each method.

Screenshot of Output / Graphs / Result



```
merge_sort (Mini Project).py X
C:\Users\hp> Downloads > DAA_Assignments > merge_sort (Mini Project).py > merge

8 def merge(*args):
9     left, right = args[0] if len(args) == 1 else args
10    left_length, right_length = len(left), len(right)
11    left_index, right_index = 0, 0
12    merged = []
13    while left_index < left_length and right_index < right_length:
14        if left[left_index] <= right[right_index]:
15            merged.append(left[left_index])
16            left_index += 1
17        else:
18            merged.append(right[right_index])
19            right_index += 1
20    if left_index == left_length:
21        merged.extend(right[right_index:])
22    elif right_index == right_length:
23        merged.extend(left[left_index:])
24    return merged

160, 440, 337, 648, 116, 739, 410, 580, 565, 397, 789, 68, 104, 28, 617, 56, 850, 920, 438, 983, 664, 322, 77, 130, 597, 885, 33, 830, 653, 284, 706, 435, 829, 865, 234, 533, 336, 26, 853, 465, 846, 659
, 367, 160, 654, 921, 709, 80, 158, 290, 302, 113, 162, 623, 458, 817, 435, 756, 974, 514, 933, 205, 114, 326, 487, 511, 114, 59, 491, 134, 1000, 995, 952, 996, 104, 493, 86, 107, 193, 432, 511, 212, 21
4, 0, 471, 827, 797, 246, 732, 881, 540, 419, 595, 971, 705, 555, 713, 810, 198, 564, 528, 875, 480, 251, 88, 733, 125, 549, 606, 640, 510, 632, 191, 564, 812, 608, 849, 604, 388, 129, 774, 531, 641, 55
5, 748, 572, 709, 377, 65, 995, 578, 584, 297, 706, 294, 705, 779, 236, 928, 34, 799, 147, 723, 393, 704, 5, 494, 300, 446, 521, 954, 956, 589, 247, 33, 259, 748, 607, 569, 623, 584, 884, 740, 232, 962,
214, 656, 767, 992, 599, 189, 893, 728, 919, 188, 678, 483, 424, 146, 947, 744, 62, 468, 927, 182, 939, 37, 649, 312, 385, 469, 701, 93, 184, 232, 595, 830, 321, 719, 169, 779, 676, 926, 260, 777, 753,
788, 678, 18, 961, 41, 693, 168, 412, 609, 455, 621, 413, 807, 576, 54, 774, 332, 209, 562, 11, 712, 982, 761, 234, 66, 387, 326, 464, 118, 605, 450, 261, 928, 964, 703, 188, 205, 752, 661, 363, 405, 1
15, 389, 280, 97, 747, 94, 756, 594, 974, 916, 172, 805, 197, 0, 582, 448, 154, 84, 661, 295, 144, 262, 309, 98, 210, 40, 451, 798, 293, 439, 898, 373, 480, 397, 351, 190, 774, 599, 12, 181, 782, 271, 4
91, 813, 317, 849, 669, 930, 946, 361, 461, 459, 754, 7, 194, 65, 495, 863, 942, 64, 472, 860, 114, 935, 425, 340, 796, 490, 947, 726, 507, 46, 11, 287, 654, 510, 140, 515, 158, 847, 539, 13, 896, 223,
937, 441, 743, 103, 430, 702, 974, 666, 933, 572, 321, 312, 232, 362, 739, 865, 269, 885, 197, 609, 79, 733, 144, 276, 34, 328, 642, 399, 357, 443, 671, 492, 766, 737, 180, 460, 245, 682, 932, 822, 452,
154, 808, 831, 207, 407, 163, 49, 56, 79, 753, 351, 387, 497, 696, 956, 338, 953, 745, 239, 469, 715, 936, 738, 342, 813, 949, 107, 897, 213, 94, 240, 897, 342, 291, 380, 604, 876, 299, 392, 154, 836,
39, 780, 388, 947, 651, 484, 31, 68, 717, 504, 592, 122, 212, 396, 671, 217, 249, 335, 963, 71, 403, 81, 295, 888, 642, 867, 430, 221, 570, 666, 763, 383, 112, 729, 29, 538, 227, 553, 724, 713, 680, 519
, 162, 116, 519, 483, 1000, 585, 918, 961, 678, 141, 934, 923, 447, 544, 47, 623, 468, 592, 679, 548, 611, 297, 234, 0, 776, 198, 26, 944, 442, 857, 31, 176, 863, 960, 55, 979, 967, 65, 577, 170, 397, 6
29, 213, 263, 679, 698, 255, 352, 230, 968, 609, 276, 401, 843, 191, 844, 247, 219, 316, 814, 102, 368, 726, 504, 267, 882, 862, 312, 900, 299, 105, 372, 682, 344, 838, 571, 653, 122, 881, 988, 965, 586
, 352, 119, 414, 434, 158, 4, 585, 266, 283, 278, 793, 364, 858, 352, 501, 75, 73, 147, 570, 62, 909, 42, 419, 916, 26, 39, 184, 265, 834, 176, 226, 71, 474, 916, 572, 778, 222, 933, 98, 718, 551, 161,
932, 816, 706, 672, 704, 590, 440, 622, 432, 166, 92, 424, 590, 419, 857, 746, 197, 907, 923, 93, 486, 963, 78, 752, 696, 690, 993, 776, 367, 550, 398, 965, 856, 556, 236, 530, 45, 932, 7, 350, 198, 582
, 386, 264, 270, 170, 312, 399, 828, 474, 506, 450, 749, 12, 508, 103, 567, 176, 19, 221, 331, 524, 8, 423, 782, 973, 495, 571, 272, 623, 429, 829, 122, 123, 815, 289, 595, 597, 985, 228, 824, 104, 114,
413, 66, 196, 230, 828, 798, 211, 264, 361, 675, 652, 938, 202, 253, 36, 184, 137, 323, 565, 488, 281, 467, 931, 340, 981, 245, 897, 916, 259, 844, 315, 135, 119, 294, 984, 945, 465, 22, 743, 616, 684,
539, 232, 645, 695, 563, 271, 586, 434, 599, 775, 534, 164, 913, 458, 178, 297, 52, 396, 945, 639, 65, 730, 498, 777, 568, 362, 246, 468, 393, 421, 924, 287, 610, 968, 385, 277, 894, 20, 931, 371, 150,
171, 69, 164, 208, 991, 143, 694, 133, 966, 224, 16, 959, 913, 826, 660, 237, 347, 71, 35, 402, 87, 552, 395, 478, 52, 270, 282, 388, 348, 919, 904, 752, 593, 621, 47, 495, 412, 828, 195, 690, 230, 57,
130, 332, 345, 752, 1000, 786, 365, 510, 540, 360]
Time taken for merge_sort: 0.002570629119873047
Time taken for merge_sort_multithreaded: 0.34221816862927246
PS C:\Users\hp>
```

Conclusion

The Merge Sort tool in Python successfully demonstrates both standard and multithreaded sorting. It highlights the efficiency of divide and conquer algorithms and the benefits of parallel processing for large datasets. Further enhancements could include a graphical user interface and support for additional sorting algorithms.