

ASSIGNMENT - 12

Thumma Hasini | 2303A52076 | Batch-37

Task 1: Sorting Student Records

Prompt

Generate a Python program that creates a Student class (Name, Roll Number, CGPA), generates at least 10,000 student records, implements recursive Quick Sort and Merge Sort to sort students by CGPA in descending order, measures runtime using the time module, and displays the top 10 students. Include complexity analysis and formatted output.

Output Explanation

The output will contain:

1. **Student class definition**
2. **Quick Sort and Merge Sort functions**
3. Runtime results such as:

Quick Sort Time: 0.021 seconds

Merge Sort Time: 0.028 seconds

4. Top 10 students printed in descending CGPA.

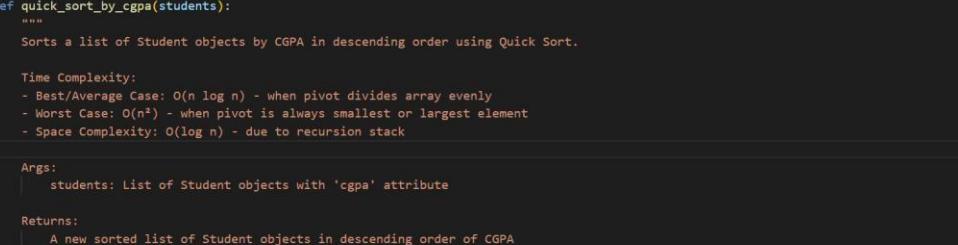
What the Output Shows

- Both algorithms correctly sort in descending order.
- Quick Sort may appear slightly faster in practice.
- Merge Sort provides stable and consistent $O(n \log n)$.
- The top 10 list verifies correct sorting logic.
- Runtime comparison validates algorithm efficiency experimentall

The screenshot shows a code editor window with the following details:

- Title Bar:** Welcome - AIAC Code.py
- File Path:** C:\Users\lenovo\Desktop>3-2>AIAC>AIAC Code.py
- Code Content:**

```
1 # Generate a Python program that defines a Student class with attributes: Name, Roll Number, and CGPA. Store at least 1000 student records
2 import random
3
4 class Student:
5     def __init__(self, name, roll_number, cgpa):
6         self.name = name
7         self.roll_number = roll_number
8         self.cgpa = cgpa
9
10    def __repr__(self):
11        return f"Student(name='{self.name}', roll_number={self.roll_number}, cgpa={self.cgpa})"
12
13
14 def generate_students(count=1000):
15     """Generate student records with random data"""
16     students = []
17     first_names = ["Aarav", "Vivaan", "Aditya", "Arjun", "Rohit", "Priya", "Ananya", "Zara", "Neha", "Pooja"]
18     last_names = ["Sharma", "Patel", "Kumar", "Singh", "Gupta", "Verma", "Nair", "Reddy", "Mishra", "Rao"]
19
20     for i in range(1, count + 1):
21         name = f"{random.choice(first_names)} {random.choice(last_names)}"
22         roll_number = i
23         cgpa = round(random.uniform(5.0, 10.0), 2)
24         students.append(Student(name, roll_number, cgpa))
25
26     return students
27
28
29 def sort_by_cgpa(students, descending=True):
30     """Sort students by CGPA in descending order (default)"""
31     return sorted(students, key=lambda x: x.cgpa, reverse=descending)
32
33
34 # Main execution
```
- Status Bar:** Ln 4, Col 15 | Spaces: 4 | UTF-8 | CRLF | {} Python | 3.14.2 | Python 3.14.2 | Go Live



The screenshot shows a code editor window with the following details:

- Title Bar:** File Edit Selection View Go ⏪ ⏪ ⏪ ⏪ ⏪ ⏪ ⏪ ⏪ ⏪ ⏪ ⏪ ⏪ ⏪ ⏪
- Search Bar:** Q Search
- Toolbar:** Standard file operations (New, Open, Save, Print, etc.)
- Left Sidebar:** Includes icons for File, Edit, Selection, View, Go, and various search and filter options.
- Code Area:** The code is written in Python and defines a function `quick_sort_by_cgpa` that sorts a list of `Student` objects by CGPA in descending order using the Quick Sort algorithm.
- Code Content:**

```
C: > Users > lenovo > Desktop > 3-2 AIAC > AIAC Code.py > quick_sort_by_cgpa
 1 def quick_sort_by_cgpa(students):
 2     """
 3         Sorts a list of Student objects by CGPA in descending order using Quick Sort.
 4
 5         Time Complexity:
 6             - Best/Average Case: O(n log n) - when pivot divides array evenly
 7             - Worst Case: O(n2) - when pivot is always smallest or largest element
 8             - Space Complexity: O(log n) - due to recursion stack
 9
10     Args:
11         students: List of Student objects with 'cgpa' attribute
12
13     Returns:
14         A new sorted list of Student objects in descending order of CGPA
15     """
16
17     # Base case: lists with 0 or 1 element are already sorted
18     if len(students) <= 1:
19         return students
20
21     # Choose pivot as the middle element (better than first/last for partially sorted data)
22     pivot = students[len(students) // 2]
23
24     # Partition into three lists: greater, equal, and less than pivot
25     greater = [] # CGPA > pivot (for descending order)
26     equal = []   # CGPA == pivot
27     less = []    # CGPA < pivot
```



The screenshot shows a code editor window with the following details:

- Title Bar:** File Edit Selection View Go ...
- Search Bar:** Search
- Toolbar:** Includes icons for file operations like Open, Save, Find, and Print.
- Code Area:** Displays Python code for quick sorting students by CGPA. The code includes a Student class and usage examples.
- Status Bar:** Shows the path C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > quick_sort_by_cgpa and the line number 54.

```
File Edit Selection View Go ...
Q Search
Welcome AIAC Code.py X
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > quick_sort_by_cgpa
1 def quick_sort_by_cgpa(students):
28
29     # Partition the list
30     for student in students:
31         if student.cgpa > pivot.cgpa:
32             greater.append(student)
33         elif student.cgpa == pivot.cgpa:
34             equal.append(student)
35         else:
36             less.append(student)
37
38     # Recursively sort and combine: greater + equal + less (descending order)
39     return quick_sort_by_cgpa(greater) + equal + quick_sort_by_cgpa(less)
40
41
42 # Example Student class for testing
43 class Student:
44     def __init__(self, name, cgpa):
45         self.name = name
46         self.cgpa = cgpa
47
48     def __repr__(self):
49         return f"Student({self.name}, {self.cgpa})"
50
51
52 # Example usage
53 if __name__ == "__main__":
54     students = [
```

C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > Student > __init__

```
1 # Implement Merge Sort in Python to sort Student objects by CGPA in descending order. Use recursion and a separate merge function.
2 """
3 Merge Sort Implementation for Student Objects
4
5 Time Complexity: O(n log n) - dividing and merging
6 Space Complexity: O(n) - temporary arrays during merge
7 """
8
9
10 class Student:
11     """Represents a student with name and CGPA."""
12
13     def __init__(self, name, cgpa):
14         """
15             Initialize a Student object.
16
17             Args:
18                 name (str): Student's name
19                 cgpa (float): Student's CGPA
20         """
21         self.name = name
22         self.cgpa = cgpa
23
24     def __repr__(self):
25         return f"Student({self.name}, {self.cgpa})"
26
27
28 def merge(students, left, mid, right):
29     """
30         Merge two sorted subarrays in descending order by CGPA.
31
32         Args:
33             students (list): List of Student objects
34             left (int): Starting index of left subarray
35             mid (int): Ending index of left subarray
36             right (int): Ending index of right subarray
37     """
38     left_part = students[left:mid + 1]
39     right_part = students[mid + 1:right + 1]
40
41     i = j = 0
42     k = left
43
44     # Merge in descending order
45     while i < len(left_part) and j < len(right_part):
46         if left_part[i].cgpa >= right_part[j].cgpa:
47             students[k] = left_part[i]
48             i += 1
49         else:
50             students[k] = right_part[j]
51             j += 1
52         k += 1
53
54     # Copy remaining elements
55     while i < len(left_part):
56         students[k] = left_part[i]
57         i += 1
58         k += 1
59
60     while j < len(right_part):
61         students[k] = right_part[j]
62         j += 1
63         k += 1
```

C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > Student > __init__

```
28 def merge(students, left, mid, right):
29     """
30         Merge two sorted subarrays in descending order by CGPA.
31
32         Args:
33             students (list): List of Student objects
34             left (int): Starting index of left subarray
35             mid (int): Ending index of left subarray
36             right (int): Ending index of right subarray
37     """
38     left_part = students[left:mid + 1]
39     right_part = students[mid + 1:right + 1]
40
41     i = j = 0
42     k = left
43
44     # Merge in descending order
45     while i < len(left_part) and j < len(right_part):
46         if left_part[i].cgpa >= right_part[j].cgpa:
47             students[k] = left_part[i]
48             i += 1
49         else:
50             students[k] = right_part[j]
51             j += 1
52         k += 1
53
54     # Copy remaining elements
55     while i < len(left_part):
56         students[k] = left_part[i]
57         i += 1
58         k += 1
59
60     while j < len(right_part):
61         students[k] = right_part[j]
62         j += 1
63         k += 1
```

The screenshot shows a Python code editor interface with a dark theme. On the left is the code editor pane displaying `AIAC Code.py`. The code implements a merge sort algorithm for a list of `Student` objects based on CGPA. It includes a docstring explaining the recursive divide and sort process. The code then demonstrates sorting a list of four students: Alice (3.8), Bob (3.5), Charlie (3.9), and Diana (3.7). The terminal pane at the bottom shows the execution of the script and its output. The terminal output shows the initial list of students (Before sorting) and the sorted list (After sorting, descending by CGPA).

```
C:\> Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > Student > __init__  
66 def merge_sort(students, left, right):  
67     """  
68         Recursively divide and sort students by CGPA in descending order.  
69     """  
70     Args:  
71         students (list): List of Student objects  
72         left (int): Starting index  
73         right (int): Ending index  
74     """  
75     if left < right:  
76         mid = (left + right) // 2  
77         merge_sort(students, left, mid)  
78         merge_sort(students, mid + 1, right)  
79         merge(students, left, mid, right)  
80       
81     # Example usage  
82 if __name__ == "__main__":  
83     students = [  
84         Student("Alice", 3.8),  
85         Student("Bob", 3.5),  
86         Student("Charlie", 3.9),  
87         Student("Diana", 3.7),  
88     ]  
89       
90     print("Before sorting:")  
91     for s in students:  
92         print(s)  
93       
94     merge_sort(students, 0, len(students) - 1)  
95       
96     print("\nAfter sorting (descending by CGPA):")  
97     for s in students:  
98         print(s)  
99  
Ln 19, Col 41 Spaces: 4 UTF-8 CRLF { } Python 3.14.2 Python 3.14.2  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS  
PS C:\Users\lenovo> & C:/Python314/python.exe "c:/Users/lenovo/Desktop/3-2/AIAC/AIAC Code.py"  
Before sorting:  
Student(Alice, 3.8)  
Student(Bob, 3.5)  
Student(Charlie, 3.9)  
Student(Diana, 3.7)  
  
After sorting (descending by CGPA):  
Student(Charlie, 3.9)  
Student(Alice, 3.8)  
Student(Diana, 3.7)  
Student(Bob, 3.5)  
PS C:\Users\lenovo>
```

```
2 import time
3 from dataclasses import dataclass
4 from typing import List
5
6 @dataclass
7 class Student:
8     name: str
9     student_id: int
10    cgpa: float
11
12    def __repr__(self):
13        return f"Student({self.name}, ID: {self.student_id}, CGPA: {self.cgpa})"
14
15 def generate_random_students(count: int) -> List[Student]:
16     """Generate random student records."""
17     names = ["Alice", "Bob", "Charlie", "Diana", "Eve", "Frank", "Grace", "Henry", "Iris", "Jack"]
18     students = []
19     for i in range(count):
20         students.append(Student(
21             name=random.choice(names) + str(i),
22             student_id=1000 + i,
23             cgpa=round(random.uniform(2.0, 4.0), 2)
24         ))
25     return students
26
27 def quick_sort(arr: List[Student]) -> List[Student]:
28     """Sort students by CGPA using Quick Sort."""
29     if len(arr) <= 1:
30         return arr
31     pivot = arr[len(arr) // 2]
32     left = [x for x in arr if x.cgpa > pivot.cgpa]
33     middle = [x for x in arr if x.cgpa == pivot.cgpa]
34     right = [x for x in arr if x.cgpa < pivot.cgpa]
```

The screenshot shows the PyCharm IDE interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, ...
- Search Bar:** Search
- Toolbars:** Welcome, AIAC Code.py X
- Code Area:** The main area displays the Python code for sorting students by CGPA using Merge Sort. The code includes functions for merge sort, merging two sorted lists, and printing top N students.
- Right Panel:** Shows the file tree and navigation pane.
- Status Bar:** Line 26, Col 1, Spaces: 4, UTF-8, {}, Python 3.14.2, Python 3.14.2

```
37 def merge_sort(arr: List[Student]) -> List[Student]:  
38     """Sort students by CGPA using Merge Sort."""  
39     if len(arr) <= 1:  
40         return arr  
41     mid = len(arr) // 2  
42     left = merge_sort(arr[:mid])  
43     right = merge_sort(arr[mid:])  
44     return merge(left, right)  
45  
46 def merge(left: List[Student], right: List[Student]) -> List[Student]:  
47     """Merge two sorted lists."""  
48     result = []  
49     i = j = 0  
50     while i < len(left) and j < len(right):  
51         if left[i].cgpa >= right[j].cgpa:  
52             result.append(left[i])  
53             i += 1  
54         else:  
55             result.append(right[j])  
56             j += 1  
57     result.extend(left[i:])  
58     result.extend(right[j:])  
59     return result  
60  
61 def print_top_students(students: List[Student], top_n: int = 10):  
62     """Print top N students by CGPA in formatted output."""  
63     print(f"\n{'='*60}")  
64     print(f"TOP {top_n} STUDENTS BY CGPA")  
65     print(f"{'='*60}")  
66     print(f'{Rank:<6} {'Name':<20} {'ID':<10} {'CGPA':<10}')  
67     print(f"{'='*60}")  
68     for i, student in enumerate(students[:top_n], 1):  
69         print(f'{i:<6} {student.name:<20} {student.student_id:<10} {student.cgpa:<10.2f}')
```

The image shows a dark-themed code editor interface with two tabs open, both titled "AIAC Code.py".

Top Tab Content:

```

71 def compare_sorting_algorithms():
72     """Compare Quick Sort and Merge Sort runtime."""
73     print("Generating 10,000 random student records...")
74     students = generate_random_students(10000)
75
76     print(f"\n{'='*60}")
77     print("SORTING ALGORITHM COMPARISON")
78     print(f"{'='*60}")
79     print(f"{'Algorithm':<20} {'Execution Time (seconds)':<25}")
80     print(f"{'-'*60}")
81
82     # Quick Sort
83     students_copy = students.copy()
84     start_time = time.time()
85     sorted_qs = quick_sort(students_copy)
86     qs_time = time.time() - start_time
87     print(f"{'Quick Sort':<20} {qs_time:<25.6f}")
88
89     # Merge Sort
90     students_copy = students.copy()
91     start_time = time.time()
92     sorted_ms = merge_sort(students_copy)
93     ms_time = time.time() - start_time
94     print(f"{'Merge Sort':<20} {ms_time:<25.6f}")
95
96     print(f"{'='*60}\n")
97
98     # Print top 10 students from both sorts
99     print_top_students(sorted_qs, 10)
100    print_top_students(sorted_ms, 10)
101
102 if __name__ == "__main__":
103     compare_sorting_algorithms()

```

Bottom Tab Content:

```

72 def compare_sorting_algorithms():
73     """Compare Quick Sort and Merge Sort runtime."""
74     students_copy = students.copy()
75     start_time = time.time()
76     sorted_ms = merge_sort(students_copy)
77     ms_time = time.time() - start_time
78     print(f"{'Merge Sort':<20} {ms_time:<25.6f}")
79
80     print(f"{'='*60}\n")
81
82     # Print top 10 students from both sorts
83     print_top_students(sorted_qs, 10)
84     print_top_students(sorted_ms, 10)
85
86 if __name__ == "__main__":
87     compare_sorting_algorithms()

```

Terminal View:

Rank	Name	ID	CGPA
1	Alice761	1761	4.00
2	Alice1126	2126	4.00
3	Bob1277	2277	4.00
4	Grace1845	2845	4.00
5	Frank2557	3557	4.00
6	Charlie3745	4745	4.00
7	Diana3993	4993	4.00
8	Bob4887	5887	4.00
9	Diana6307	7307	4.00
10	Henry7013	8013	4.00

Task 2: Bubble Sort

Prompt

Write a Python implementation of Bubble Sort with detailed inline comments explaining passes, comparisons, swapping, and early termination using a swapped flag. Include time and space complexity analysis.

Output Explanation

The output will include:

- **Bubble Sort implementation**
- **Comments explaining:**
 - Outer loop = number of passes
 - Inner loop = comparisons
 - Swapping adjacent elements
 - Early break if no swaps occur
- **Complexity section:**

Best Case: $O(n)$

Average Case: $O(n^2)$

Worst Case: $O(n^2)$

Space Complexity: $O(1)$

What the Output Shows

- **If input is already sorted, algorithm stops early ($O(n)$).**
- **For large datasets, Bubble Sort is inefficient.**
- **Space complexity remains constant.**
- **Code clarity improves understanding of algorithm flow.**

The image shows two side-by-side screenshots of a code editor interface, likely Visual Studio Code, displaying Python code for bubble sort. Both screenshots have a dark theme.

Top Screenshot:

```
1 # Write a Python implementation of Bubble Sort that:Sorts a list of integers in ascending order.Includes detailed inline comments.
2 """
3     Bubble Sort Algorithm Implementation
4
5     Time Complexity:
6         - Best Case: O(n) - when list is already sorted (with optimization)
7         - Average Case: O(n2) - random order
8         - Worst Case: O(n2) - when list is reverse sorted
9
10    Space Complexity: O(1) - sorts in-place, no extra space needed
11
12    Practical Use Cases:
13        - Small datasets (< 50 elements)
14        - Nearly sorted data
15        - Educational purposes
16        - When simplicity is prioritized over efficiency
17
18    NOT recommended for large datasets; use quicksort, mergesort, or timsort instead.
19 """
20
21 n = len(arr)
22
23 # Outer loop: make multiple passes through the list
24 for i in range(n):
25     # Optimization flag: if no swaps occur, list is sorted
26     swapped = False
27
28     # Inner loop: compare adjacent elements and swap if needed
29     # Each pass moves the largest unsorted element to its final position
30     for j in range(0, n - i - 1):
31         # Comparison: check if current element is greater than next
32         if arr[j] > arr[j + 1]:
33             arr[j], arr[j + 1] = arr[j + 1], arr[j]
34             # Mark that a swap occurred
35             swapped = True
36
37     # Early termination optimization:
38     # If no swaps happened in this pass, the list is already sorted
39     if not swapped:
40         break
41
42 return arr
43
44
45 # Test cases
46 if __name__ == "__main__":
47     # Test 1: Unsorted list
48     test1 = [64, 34, 25, 12, 22, 11, 90]
49     print("Unsorted:", test1)
50     print("Sorted:", bubble_sort(test1.copy()))
51
52     # Test 2: Already sorted (best case - O(n))
53     test2 = [1, 2, 3, 4, 5]
54     print("\nAlready sorted:", test2)
55     print("Result:", bubble_sort(test2.copy()))
56
57     # Test 3: Reverse sorted (worst case - O(n2))
58     test3 = [5, 4, 3, 2, 1]
59     print("\nReverse sorted:", test3)
60     print("Result:", bubble_sort(test3.copy()))
61
62     # Test 4: Single element
63     test4 = [42]
64     print("\nSingle element:", test4)
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
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
716
717
718
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
767
768
769
769
770
771
772
773
774
775
776
777
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
838
839
839
840
841
842
843
844
845
846
847
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
867
868
869
869
870
871
872
873
874
875
876
877
877
878
879
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
938
939
939
940
941
942
943
944
945
946
947
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
967
968
969
969
970
971
972
973
974
975
976
977
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1015
1016
1017
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1058
1059
1060
1061
1062
1063
1064
1065
1066
1066
1067
1068
1068
1069
1070
1071
1072
1073
1074
1075
1076
1076
1077
1078
1078
1079
1080
1081
1082
1083
1084
1085
1086
1086
1087
1088
1088
1089
1090
1091
1092
1093
1094
1095
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1105
1106
1107
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1115
1116
1117
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1126
1127
1128
1128
1129
1130
1131
1132
1133
1134
1135
1136
1136
1137
1138
1138
1139
1140
1141
1142
1143
1144
1145
1145
1146
1147
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1156
1157
1158
1158
1159
1160
1161
1162
1163
1164
1164
1165
1166
1166
1167
1168
1168
1169
1170
1171
1172
1173
1174
1175
1175
1176
1177
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1185
1186
1187
1187
1188
1189
1189
1190
1191
1192
1193
1194
1194
1195
1196
1196
1197
1198
1198
1199
1200
1201
1202
1203
1204
1204
1205
1206
1206
1207
1208
1208
1209
1210
1211
1212
1213
1214
1214
1215
1216
1216
1217
1218
1218
1219
1220
1221
1222
1223
1224
1225
1225
1226
1227
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1235
1236
1237
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1245
1246
1247
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1256
1257
1258
1258
1259
1260
1261
1262
1263
1264
1264
1265
1266
1266
1267
1268
1268
1269
1270
1271
1272
1273
1274
1274
1275
1276
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1284
1285
1286
1286
1287
1288
1288
1289
1290
1291
1292
1293
1294
1294
1295
1296
1296
1297
1298
1298
1299
1300
1301
1302
1303
1303
1304
1305
1305
1306
1307
1307
1308
1309
1309
1310
1311
1312
1313
1314
1314
1315
1316
1316
1317
1318
1318
1319
1320
1321
1322
1323
1324
1324
1325
1326
1326
1327
1328
1328
1329
1330
1331
1332
1333
1334
1334
1335
1336
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1344
1345
1346
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1354
1355
1356
1356
1357
1358
1358
1359
1360
1361
1362
1363
1364
1364
1365
1366
1366
1367
1368
1368
1369
1370
1371
1372
1373
1374
1374
1375
1376
1376
1377
1378
1378
1379
1380
1381
1382
1383
1384
1384
1385
1386
1386
1387
1388
1388
1389
1390
1391
1392
1393
1393
1394
1395
1395
1396
1397
1397
1398
1399
1399
1400
1401
1402
1403
1403
1404
1405
1405
1406
1407
1407
1408
1409
1409
1410
1411
1412
1413
1414
1414
1415
1416
1416
1417
1418
1418
1419
1420
1421
1422
1423
1424
1424
1425
1426
1426
1427
1428
1428
1429
1430
1431
1432
1433
1434
1434
1435
1436
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1444
1445
1446
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1454
1455
1456
1456
1457
1458
1458
1459
1460
1461
1462
1463
1464
1464
1465
1466
1466
1467
1468
1468
1469
1470
1471
1472
1473
1474
1474
1475
1476
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1484
1485
1486
1486
1487
1488
1488
1489
1490
1491
1492
1493
1493
1494
1495
1495
1496
1497
1497
1498
1499
1499
1500
1501
1502
1503
1503
1504
1505
1505
1506
1507
1507
1508
1509
1509
1510
1511
1512
1513
1514
1514
1515
1516
1516
1517
1518
1518
1519
1520
1521
1522
1523
1524
1524
1525
1526
1526
1527
1528
1528
1529
1530
1531
1532
1533
1534
1534
1535
1536
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1544
1545
1546
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1554
1555
1556
1556
1557
1558
1558
1559
1560
1561
1562
1563
1564
1564
1565
1566
1566
1567
1568
1568
1569
1570
1571
1572
1573
1574
1574
1575
1576
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1584
1585
1586
1586
1587
1588
1588
1589
1590
1591
1592
1593
1593
1594
1595
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1603
1604
1605
1605
1606
1607
1607
1608
1609
1609
1610
1611
1612
1613
1614
1614
1615
1616
1616
1617
1618
1618
1619
1620
1621
1622
1623
1624
1624
1625
1626
1626
1627
1628
1628
1629
1630
1631
1632
1633
1634
1634
1635
1636
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1644
1645
1646
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1654
1655
1656
1656
1657
1658
1658
1659
1660
1661
1662
1663
1664
1664
1665
1666
1666
1667
1668
1668
1669
1670
1671
1672
1673
1674
1674
1675
1676
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1684
1685
1686
1686
1687
1688
1688
1689
1690
1691
1692
1693
1693
1694
1695
1695
1696
1697
1697
1698
1699
1699
1700
1701
1702
1703
1703
1704
1705
1705
1706
1707
1707
1708
1709
1709
1710
1711
1712
1713
1714
1714
1715
1716
1716
1717
1718
1718
1719
1720
1721
1722
1723
1724
1724
1725
1726
1726
1727
1728
1728
1729
1730
1731
1732
1733
1734
1734
1735
1736
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1744
1745
1746
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1754
1755
1756
1756
1757
1758
1758
1759
1760
1761
1762
1763
1764
1764
1765
1766
1766
1767
1768
1768
1769
1770
1771
1772
1773
1774
1774
1775
1776
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1784
1785
1786
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1794
1795
1796
1796
1797
1798
1798
1799
1800
1801
1802
1803
1803
1804
1805
1805
1806
1807
1807
1808
1809
1809
1810
1811
1812
1813
1814
1814
1815
1816
1816
1817
1818
1818
1819
1820
1821
1822
1823
1824
1824
1825
1826
1826
1827
1828
1828
1829
1830
1831
1832
1833
1834
1834
1835
1836
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1844
1845
1846
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1854
1855
1856
1856
1857
1858
1858
1859
1860
1861
1862
1863
1864
1864
1865
1866
1866
1867
1868
1868
1869
1870
1871
1872
1873
1874
1874
1875
1876
1876
1877
1878
1878
1879
1880
1881
1882
1883
1884
1884
1885
1886
1886
1887
1888
1888
1889
1890
1891
1892
1893
1893
1894
1895
1895
1896
1897
1897
1898
1899
1899
1900
1901
1902
1903
1903
1904
1905
1905
1906
1907
1907
1908
1909
1909
1910
1911
1912
1913
1914
1914
1915
1916
1916
1917
1918
1918
1919
1920
1921
1922
1923
1924
1924
1925
1926
1926
1927
1928
1928
1929
1930
1931
1932
1933
1934
1934
1935
1936
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1944
1945
1946
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1954
1955
1956
1956
1957
1958
1958
1959
1960
1961
1962
1963
1964
1964
```

```

# Test 3: Reverse sorted (worst case - O(n^2))
test3 = [5, 4, 3, 2, 1]
print("\nReverse sorted:", test3)
print("Result:", bubble_sort(test3.copy()))

# Test 4: Single element
test4 = [42]
print("\nSingle element:", test4)
print("Result:", bubble_sort(test4.copy()))

# Test 5: Duplicates
test5 = [3, 1, 3, 2, 1]
print("\nWith duplicates:", test5)
print("Result:", bubble_sort(test5.copy()))

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Unsorted: [64, 34, 25, 12, 22, 11, 90]
Sorted: [11, 12, 22, 25, 34, 64, 90]

Already sorted: [1, 2, 3, 4, 5]
Result: [1, 2, 3, 4, 5]

Reverse sorted: [5, 4, 3, 2, 1]
Result: [1, 2, 3, 4, 5]

Single element: [42]
Result: [42]

With duplicates: [3, 1, 3, 2, 1]
Result: [1, 1, 2, 3, 3]

Ln 2, Col 322 Spaces: 4 UTF-8 CRLF {} Python 3.14.2 Python 3.14.2

Task 3: Quick Sort vs Merge Sort Comparison

Prompt

Implement recursive Quick Sort and Merge Sort. Test them on random, sorted, and reverse-sorted lists. Measure runtime and print results in tabular format. Explain best, average, and worst-case complexities.

Output Explanation The

output will show:

A comparison table like:

Input Type	Quick Sort	Merge Sort
------------	------------	------------

Random	0.01 sec	0.015 sec
--------	----------	-----------

Sorted	0.05 sec	0.014 sec
--------	----------	-----------

Reverse	0.06 sec	0.016 sec
---------	----------	-----------

What the Output Shows

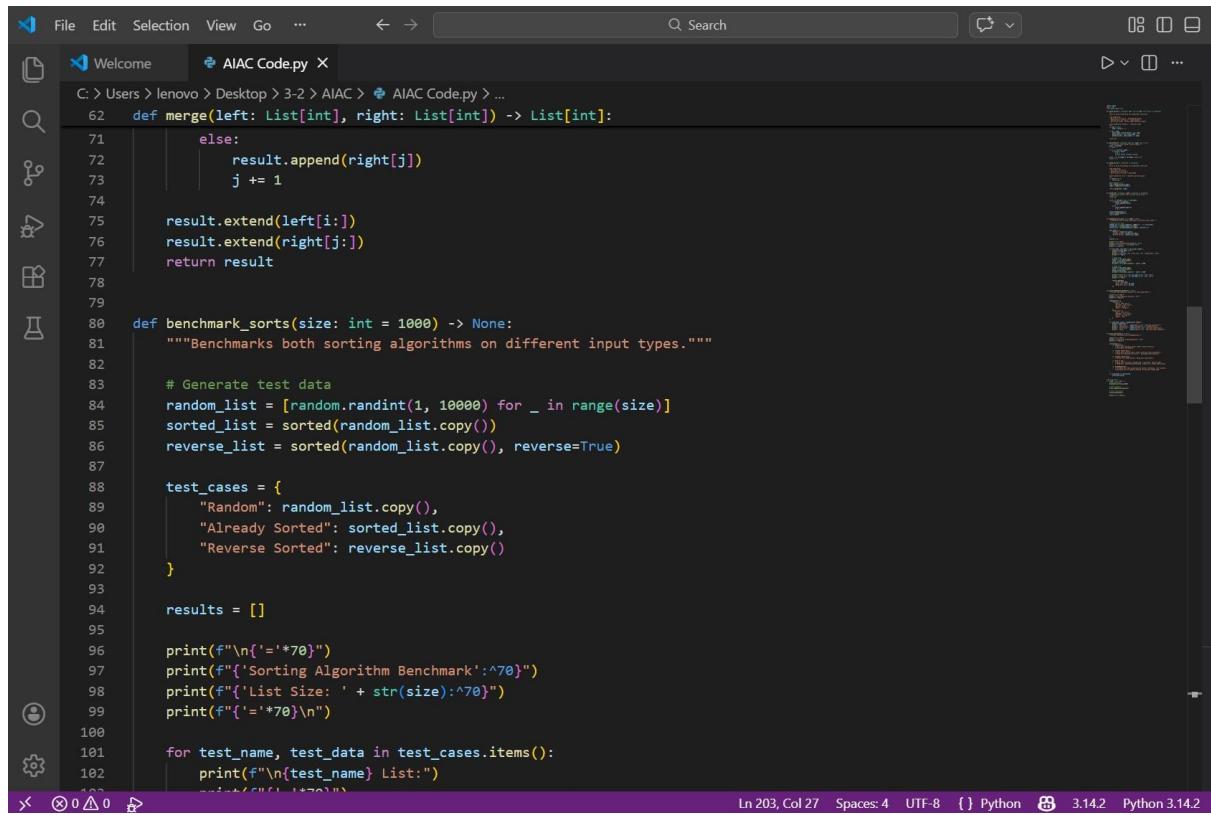
- Quick Sort performs well on random data.
- Quick Sort slows down for already sorted input (worst case).
- Merge Sort remains consistent across all inputs.
- Experimental results confirm theoretical time complexities.

```
File Edit Selection View Go ... ← → Q Search 08 □ ... Welcome AIAC Code.py C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ... 1 import random 2 import time 3 from typing import List 4 5 def quick_sort(arr: List[int], low: int = 0, high: int = None) -> List[int]: 6     """ 7         Sorts an array using Quick Sort algorithm (recursive). 8 9         Time Complexity: 10             - Best case: O(n log n) - balanced partitions 11             - Average case: O(n log n) - random pivots 12             - Worst case: O(n2) - pivot always smallest/largest 13 14         Space Complexity: O(log n) - recursion stack 15     """ 16     if high is None: 17         high = len(arr) - 1 18 19     if low < high: 20         pivot_index = partition(arr, low, high) 21         quick_sort(arr, low, pivot_index - 1) 22         quick_sort(arr, pivot_index + 1, high) 23 24     return arr 25 26 27 def partition(arr: List[int], low: int, high: int) -> int: 28     """Partitions array around a pivot element.""" 29     pivot = arr[high] 30     i = low - 1 31 32     for j in range(low, high): 33         if arr[j] < pivot: 34             i += 1 35 36     arr[i], arr[high] = arr[high], arr[i] 37 38     return i 39 40 41 def merge_sort(arr: List[int]) -> List[int]: 42     """ 43         Sorts an array using Merge Sort algorithm (recursive). 44 45         Time Complexity: 46             - Best case: O(n log n) 47             - Average case: O(n log n) 48             - Worst case: O(n log n) - guaranteed 49 50         Space Complexity: O(n) - requires auxiliary space 51     """ 52     if len(arr) <= 1: 53         return arr 54 55     mid = len(arr) // 2 56     left = merge_sort(arr[:mid]) 57     right = merge_sort(arr[mid:]) 58 59     return merge(left, right) 60 61 62 def merge(left: List[int], right: List[int]) -> List[int]: 63     """Merges two sorted arrays into one sorted array.""" 64     result = [] 65     i = j = 0 66 67     while i < len(left) and j < len(right): 68         if left[i] <= right[j]: 69             result.append(left[i]) 70             i += 1 71         else:
```

Ln 203, Col 27 Spaces: 4 UTF-8 {} Python 3.14.2 Python 3.14.2

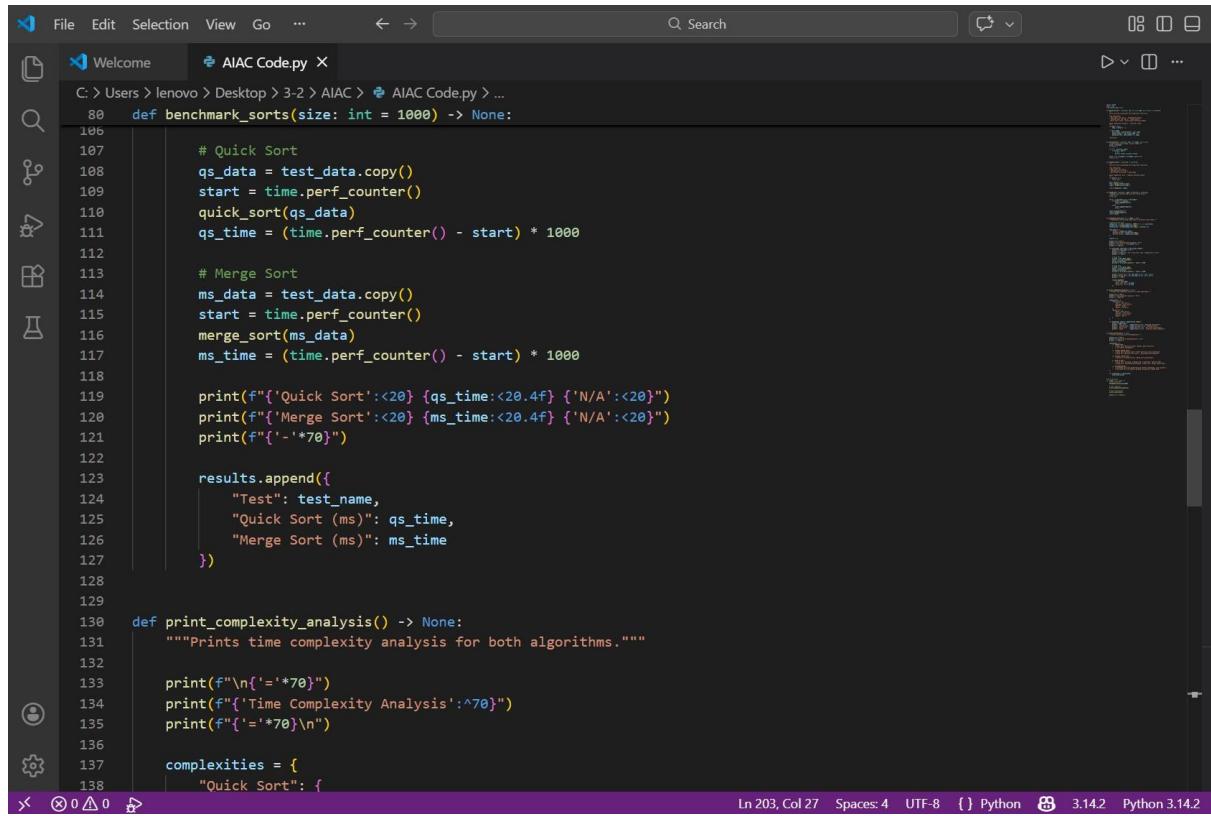
```
File Edit Selection View Go ... ← → Q Search 08 □ ... Welcome AIAC Code.py C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ... 39 40 41 def merge_sort(arr: List[int]) -> List[int]: 42     """ 43         Sorts an array using Merge Sort algorithm (recursive). 44 45         Time Complexity: 46             - Best case: O(n log n) 47             - Average case: O(n log n) 48             - Worst case: O(n log n) - guaranteed 49 50         Space Complexity: O(n) - requires auxiliary space 51     """ 52     if len(arr) <= 1: 53         return arr 54 55     mid = len(arr) // 2 56     left = merge_sort(arr[:mid]) 57     right = merge_sort(arr[mid:]) 58 59     return merge(left, right) 60 61 62 def merge(left: List[int], right: List[int]) -> List[int]: 63     """Merges two sorted arrays into one sorted array.""" 64     result = [] 65     i = j = 0 66 67     while i < len(left) and j < len(right): 68         if left[i] <= right[j]: 69             result.append(left[i]) 70             i += 1 71         else:
```

Ln 203, Col 27 Spaces: 4 UTF-8 {} Python 3.14.2 Python 3.14.2



```
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
62     def merge(left: List[int], right: List[int]) -> List[int]:
71         else:
72             result.append(right[j])
73             j += 1
74
75         result.extend(left[i:])
76     result.extend(right[j:])
77     return result
78
79
80     def benchmark_sorts(size: int = 1000) -> None:
81         """Benchmarks both sorting algorithms on different input types."""
82
83         # Generate test data
84         random_list = [random.randint(1, 10000) for _ in range(size)]
85         sorted_list = sorted(random_list.copy())
86         reverse_list = sorted(random_list.copy(), reverse=True)
87
88         test_cases = {
89             "Random": random_list.copy(),
90             "Already Sorted": sorted_list.copy(),
91             "Reverse Sorted": reverse_list.copy()
92         }
93
94         results = []
95
96         print(f"\n{'='*70}")
97         print(f"{'Sorting Algorithm Benchmark':^70}")
98         print(f"{'List Size: ' + str(size):^70}")
99         print(f"{'='*70}\n")
100
101     for test_name, test_data in test_cases.items():
102         print(f"\n{test_name} List:")
103         print(f"{'='*70}\n")
```

Ln 203, Col 27 Spaces: 4 UTF-8 [] Python 3.14.2 Python 3.14.2



```
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
80     def benchmark_sorts(size: int = 1000) -> None:
106
107         # Quick Sort
108         qs_data = test_data.copy()
109         start = time.perf_counter()
110         quick_sort(qs_data)
111         qs_time = (time.perf_counter() - start) * 1000
112
113         # Merge Sort
114         ms_data = test_data.copy()
115         start = time.perf_counter()
116         merge_sort(ms_data)
117         ms_time = (time.perf_counter() - start) * 1000
118
119         print(f"{'Quick Sort':<20} {qs_time:<20.4f} {'N/A':<20}")
120         print(f"{'Merge Sort':<20} {ms_time:<20.4f} {'N/A':<20}")
121         print(f"{'='*70}\n")
122
123         results.append({
124             "Test": test_name,
125             "Quick Sort (ms)": qs_time,
126             "Merge Sort (ms)": ms_time
127         })
128
129
130     def print_complexity_analysis() -> None:
131         """Prints time complexity analysis for both algorithms."""
132
133         print(f"\n{'='*70}")
134         print(f"{'Time Complexity Analysis':^70}")
135         print(f"{'='*70}\n")
136
137         complexities = {
138             "Quick Sort": {
```

Ln 203, Col 27 Spaces: 4 UTF-8 [] Python 3.14.2 Python 3.14.2

```
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
130 def print_complexity_analysis() -> None:
131
132     complexities = {
133         "Quick Sort": {
134             "Best": "O(n log n)",
135             "Average": "O(n log n)",
136             "Worst": "O(n2)",
137             "Space": "O(log n)"
138         },
139         "Merge Sort": {
140             "Best": "O(n log n)",
141             "Average": "O(n log n)",
142             "Worst": "O(n log n)",
143             "Space": "O(n)"
144         }
145     }
146
147     for algorithm, cases in complexities.items():
148         print(f"\n{algorithm}:")
149         print(f" Best case: {cases['Best'][:15]} - Balanced partitions")
150         print(f" Average case: {cases['Average'][:15]} - Random input")
151         print(f" Worst case: {cases['Worst'][:15]} - Poor pivot selection")
152         print(f" Space: {cases['Space'][:15]} - Auxiliary space needed\n")
153
154     def print_conclusions() -> None:
155         """Prints conclusions and recommendations."""
156
157         print(f"\n{'*70}")
158         print(f"{'Conclusions & Recommendations':^70}")
159         print(f"{'*70}\n")
160
161         conclusions = [
162             "1. RANDOM DATA -"
163         ]
164
165         for conclusion in conclusions:
166             print(conclusion)
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
```

```
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
160 def print_conclusions() -> None:
161     ]
162
163     for conclusion in conclusions:
164         print(conclusion)
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 # Main execution
193 if __name__ == "__main__":
194     # Run benchmarks
195     benchmark_sorts(size=1000)
196
197     # Print analysis
198     print_complexity_analysis()
199
200     # Print conclusions
201     print_conclusions()
202
203     print(f"\n{'*70}\n")
```

=====

Sorting Algorithm Benchmark
List Size: 1000

=====

Random List:

Algorithm	Time (ms)	Comparisons
Quick Sort	1.4789	N/A
Merge Sort	1.6257	N/A

Already Sorted List:

Algorithm	Time (ms)	Comparisons
Quick Sort	43.7259	N/A
Merge Sort	1.2163	N/A

Reverse Sorted List:

Algorithm	Time (ms)	Comparisons
Quick Sort	32.5161	N/A
Merge Sort	1.2805	N/A

Time Complexity Analysis

Quick Sort:

Best case:	$O(n \log n)$	- Balanced partitions
Average case:	$O(n \log n)$	- Random input
Worst case:	$O(n^2)$	- Poor pivot selection
Space:	$O(\log n)$	- Auxiliary space needed

Merge Sort:

Best case:	$O(n \log n)$	- Balanced partitions
Average case:	$O(n \log n)$	- Random input
Worst case:	$O(n \log n)$	- Poor pivot selection
Space:	$O(n)$	- Auxiliary space needed

Conclusions & Recommendations

1. RANDOM DATA:

- Quick Sort typically faster (better cache locality)
- Less memory overhead

2. ALREADY SORTED DATA:

- Quick Sort performs poorly ($O(n^2)$ with poor pivot selection)
- Merge Sort maintains $O(n \log n)$ - guaranteed performance

1. RANDOM DATA:

- Quick Sort typically faster (better cache locality)
- Less memory overhead

2. ALREADY SORTED DATA:

- Quick Sort performs poorly ($O(n^2)$ with poor pivot selection)
- Merge Sort maintains $O(n \log n)$ - guaranteed performance

3. REVERSE SORTED DATA:

- Similar to already sorted - Merge Sort preferred

4. WHEN TO USE:

- Quick Sort: In-place, average case is excellent, good for RAM
- Merge Sort: Guaranteed performance, stable sort, larger memory OK

5. RECOMMENDATION:

- Use Merge Sort when predictability matters (databases, file systems)
- Use Quick Sort for general purpose sorting with random data

Task 4: Inventory Management System

Prompt

Design a Python inventory system using a dictionary for searching by Product ID and Name. Implement sorting by Price and Quantity. Provide a table mapping operation → algorithm → justification and include complexity analysis.

Output Explanation

The output will include:

- **Dictionary-based storage for products.**
- **Instant search result:**

Product Found: Laptop - ₹55000 •

**Sorted list by price or
quantity.**

- **Mapping table like:**

Operation Algorithm Complexity

Search ID Hash Map O(1)

Sort Price TimSort O(n log n)

What the Output Shows

- **Searching is extremely fast due to Hash Map.**
- **Sorting complexity depends on number of products.**
- **The table justifies algorithm selection logically.**
- **Demonstrates efficient real-world system design.**

```
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
1 #Design and implement a Python-based inventory management system for a retail store containing thousands of products.
2
3 from collections import defaultdict
4 from typing import List, Dict, Tuple
5 import time
6
7 """
8 Inventory Management System for Retail Store
9 Uses Hash Maps for O(1) lookups and efficient sorting algorithms
10 """
11
12
13 class Product:
14     """Represents a single product in inventory"""
15     def __init__(self, product_id: str, name: str, price: float, quantity: int):
16         self.product_id = product_id
17         self.name = name
18         self.price = price
19         self.quantity = quantity
20
21     def __repr__(self):
22         return f"Product(ID:{self.product_id}, Name:{self.name}, Price:${self.price}, Qty:{self.quantity})"
23
24
25 class InventorySystem:
26     """
27     Inventory Management System using Hash Maps and optimized sorting
28
29     Data Structures:
30     - products_by_id: Dict[str, Product] - O(1) lookup by ID
31     - products_by_name: Dict[str, List[Product]] - O(1) lookup by Name
32     - products_list: List[Product] - For efficient bulk sorting
33     """
34
```

```
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
25 class InventorySystem:
26     """
27     def __init__(self):
28         self.products_by_id: Dict[str, Product] = {}
29         self.products_by_name: defaultdict = defaultdict(list)
30         self.products_list: List[Product] = []
31
32     def add_product(self, product_id: str, name: str, price: float, quantity: int) -> bool:
33         """
34             Add product to inventory
35             Time Complexity: O(1) average case
36         """
37         if product_id in self.products_by_id:
38             print(f"Product ID {product_id} already exists!")
39             return False
40
41         product = Product(product_id, name, price, quantity)
42         self.products_by_id[product_id] = product
43         self.products_by_name[name.lower()].append(product)
44         self.products_list.append(product)
45         return True
46
47     def search_by_id(self, product_id: str) -> Product:
48         """
49             Search product by ID using Hash Map
50             Time Complexity: O(1) average case
51         """
52         return self.products_by_id.get(product_id, None)
53
54     def search_by_name(self, name: str) -> List[Product]:
55         """
56             Search products by name using Hash Map
57             Time Complexity: O(1) average case (hash lookup) + O(k) where k = matching products
58         """
59
```

Ln 1, Col 555 Spaces: 4 UTF-8 CRLF {} Python 3.14.2 Python 3.14.2

```
C:\> Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
25  class InventorySystem:
68
69      def sort_by_price(self, ascending: bool = True) -> List[Product]:
70          """
71              Sort products by price using Timsort (Python's default)
72              Time Complexity: O(n log n) worst case, O(n) best case
73              Recommended for: Small to medium datasets with partial ordering
74          """
75          return sorted(self.products_list, key=lambda p: p.price, reverse=not ascending)
76
77      def sort_by_quantity(self, ascending: bool = True) -> List[Product]:
78          """
79              Sort products by quantity
80              Time Complexity: O(n log n) worst case, O(n) best case
81          """
82          return sorted(self.products_list, key=lambda p: p.quantity, reverse=not ascending)
83
84      def sort_by_multiple_criteria(self, criteria: List[Tuple[str, bool]]) -> List[Product]:
85          """
86              Sort by multiple criteria (price, then quantity, etc.)
87              criteria: List of tuples (field_name, ascending)
88              Time Complexity: O(n log n)
89          """
90
91          result = self.products_list.copy()
92
93          # Sort in reverse order of criteria for proper precedence
94          for field, ascending in reversed(criteria):
95              if field == "price":
96                  result.sort(key=lambda p: p.price, reverse=not ascending)
97              elif field == "quantity":
98                  result.sort(key=lambda p: p.quantity, reverse=not ascending)
99              elif field == "name":
100                 result.sort(key=lambda p: p.name, reverse=not ascending)
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
```

Ln 1, Col 555 Spaces: 4 UTF-8 CRLF {} Python 3.14.2 Python 3.14.2

```
C:\> Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
25  class InventorySystem:
102
103      def get_low_stock_products(self, threshold: int) -> List[Product]:
104          """
105              Get products with quantity below threshold
106              Time Complexity: O(n)
107          """
108          return [p for p in self.products_list if p.quantity < threshold]
109
110      def update_quantity(self, product_id: str, new_quantity: int) -> bool:
111          """
112              Update product quantity
113              Time Complexity: O(1)
114          """
115          if product_id not in self.products_by_id:
116              return False
117          self.products_by_id[product_id].quantity = new_quantity
118          return True
119
120      def display_inventory(self, products: List[Product] = None) -> None:
121          """
122              Display inventory in table format
123          """
124          if products is None:
125              products = self.products_list
126
127          if not products:
128              print("No products to display")
129              return
130
131          print(f"\n{'ID':<12} {'Name':<20} {'Price':<10} {'Quantity':<10}")
132          print("-" * 52)
133          for p in products:
134              print(f"{p.product_id:<12} {p.name:<20} ${p.price:<9.2f} {p.quantity:<10}")
```

Ln 1, Col 555 Spaces: 4 UTF-8 CRLF {} Python 3.14.2 Python 3.14.2

The screenshot shows a code editor with a dark theme. The top bar includes file navigation (File, Edit, Selection, View, Go, etc.), a search bar, and various tool icons. The left sidebar has icons for file operations like Open, Save, Find, and Settings.

The main area displays Python code for an 'InventorySystem' class and a function to print a justification table. The table compares different operations based on recommended algorithms, time complexity, and justification.

```
File Edit Selection View Go ... < > Search
Welcome AIAC Code.py ●
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
25     class InventorySystem:
132         print(f"{p.product_id:<12} {p.name:<20} ${p.price:<9.2f} {p.quantity:<10}")
133
134
135     def algorithm_justification_table():
136         """Operations vs Algorithm vs Justification"""
137
138         table = """
139
140             OPERATION      RECOMMENDED ALGO      TIME COMPLEXITY      JUSTIFICATION
141
142             Search by ID    Hash Map (Dict)    O(1) avg, O(n) worst    Instant lookups
143                                         for large datasets
144
145             Search by Name   Hash Map (Dict)    O(1) lookup + O(k) iteration
146                                         where k = results    Fast name-based
147                                         searches
148
149             Sort by Price or Quantity   Timsort (Python)  O(n log n) worst, O(n) best case  Optimized for
150                                         partially sorted
151                                         real-world data
152
153             Add Product      Hash Map Insert    O(1) average case    Constant time
154                                         insertion
155
156             Low Stock Filter  Linear Scan    O(n)                  No better option
157                                         without additional
158                                         indices
159
160             Update Quantity  Hash Map Access  O(1) average case    Direct access via
161                                         ID
162
163         print(table)
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
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
427
428
429
429
430
431
432
433
434
435
435
436
437
437
438
439
439
440
441
442
443
443
444
445
445
446
447
447
448
449
449
450
451
451
452
453
453
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
```

The screenshot shows the PyCharm IDE interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, ...
- Search Bar:** Search
- Toolbars:** Welcome, AIAC Code.py
- Left Sidebar:** Icons for Project, Find, Open, and Help.
- Code Editor:** Displays the Python script `AIAC Code.py`. The code implements a performance analysis for an inventory system, generating 1000 sample products and testing search and sort operations.

```
135     def algorithm_justification_table():
136         """
137
138         print(table)
139
140
141
142     def performance_analysis():
143         """Performance analysis for different dataset sizes"""
144
145
146         print("\n" + "="*70)
147         print("PERFORMANCE ANALYSIS FOR 1000 PRODUCTS".center(70))
148         print("="*70)
149
150
151         inventory = InventorySystem()
152
153
154         # Generate 1000 sample products
155         for i in range(1000):
156             inventory.add_product(
157                 f"P{i:04d}",
158                 f"Product_{i}",
159                 10.0 + (i % 100),
160                 100 + (i % 500)
161             )
162
163
164
165
166         # Test 1: Search by ID
167         start = time.time()
168         for i in range(100):
169             inventory.search_by_id(f"P{i:04d}")
170         search_id_time = (time.time() - start) * 1000
171
172
173
174
175         # Test 2: Sort by Price
176         start = time.time()
177         inventory.sort_by_price()
178         sort_price_time = (time.time() - start) * 1000
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
```

```
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
166     def performance_analysis():
167         # Test 3: Sort by Quantity
168         start = time.time()
169         inventory.sort_by_quantity()
170         sort_qty_time = (time.time() - start) * 1000
171
172         # Test 4: Low Stock Filter
173         start = time.time()
174         inventory.get_low_stock_products(150)
175         filter_time = (time.time() - start) * 1000
176
177         print(f"\nSearch by ID (100 searches) : {search_id_time:.3f}ms")
178         print(f"Sort by Price (1000 products) : {sort_price_time:.3f}ms")
179         print(f"Sort by Quantity (1000 products) : {sort_qty_time:.3f}ms")
180         print(f"Low Stock Filter (1000 products) : {filter_time:.3f}ms")
181
182     def main():
183         """Main function - Demo of Inventory System"""
184         print("*" * 70)
185         print("RETAIL STORE INVENTORY MANAGEMENT SYSTEM".center(70))
186         print("*" * 70)
187
188         # Initialize system
189         inventory = InventorySystem()
190
191         # Add sample products
192         inventory.add_product("P001", "Laptop", 899.99, 45)
193         inventory.add_product("P002", "Monitor", 299.99, 120)
194         inventory.add_product("P003", "Keyboard", 79.99, 250)
195         inventory.add_product("P004", "Mouse", 29.99, 500)
196         inventory.add_product("P005", "Headphones", 149.99, 80)
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
```

Ln 1, Col 555 Spaces: 4 UTF-8 CRLF {} Python 3.14.2 Python 3.14.2

```
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
211     def main():
212         # Display all products
213         print("\n1. ALL PRODUCTS IN INVENTORY:")
214         inventory.display_inventory()
215
216         # Search by ID
217         print("\n2. SEARCH BY ID (P001):")
218         product = inventory.search_by_id("P001")
219         print(f"    Found: {product}")
220
221         # Search by Name
222         print("\n3. SEARCH BY NAME (Monitor):")
223         products = inventory.search_by_name("Monitor")
224         for p in products:
225             print(f"    Found: {p}")
226
227         # Sort by Price (Low to High)
228         print("\n4. PRODUCTS SORTED BY PRICE (Ascending):")
229         inventory.display_inventory(inventory.sort_by_price(ascending=True))
230
231         # Sort by Quantity (High to Low)
232         print("\n5. PRODUCTS SORTED BY QUANTITY (Descending):")
233         inventory.display_inventory(inventory.sort_by_quantity(ascending=False))
234
235         # Low Stock Alert
236         print("\n6. LOW STOCK ALERT (< 100 units):")
237         low_stock = inventory.get_low_stock_products(100)
238         inventory.display_inventory(low_stock)
239
240         # Algorithm Justification Table
241         print("\n7. ALGORITHM RECOMMENDATIONS TABLE:")
242         algorithm_justification_table()
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
```

Ln 1, Col 555 Spaces: 4 UTF-8 CRLF {} Python 3.14.2 Python 3.14.2

AIAC Code.py

```

C:\> Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
211 def main():
212     print("\n7. ALGORITHM RECOMMENDATIONS TABLE:")
213     algorithm_justification_table()
214
215     # Performance Analysis
216     performance_analysis()
217
218     if __name__ == "__main__":
219         main()

```

TERMINAL

```

=====
RETAIL STORE INVENTORY MANAGEMENT SYSTEM
=====

1. ALL PRODUCTS IN INVENTORY:

ID      Name       Price    Quantity
-----
P001    Laptop     $899.99   45
P002    Monitor    $299.99   120
P003    Keyboard   $79.99    250
P004    Mouse      $29.99    500
P005    Headphones $149.99   80

2. SEARCH BY ID (P001):
Found: Product(ID:P001, Name:Laptop, Price:$899.99, Qty:45)

3. SEARCH BY NAME (Monitor):
Found: Product(ID:P002, Name:Monitor, Price:$299.99, Qty:120)

```

PROBLEMS **OUTPUT** **DEBUG CONSOLE** **TERMINAL** **PORTS**

TERMINAL

```

Ln 1, Col 555 Spaces: 4 UTF-8 CRLF { } Python 3.14.2 Python 3.14.2

```

AIAC Code.py

```

C:\> Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
211 def main():
212     print("\n7. ALGORITHM RECOMMENDATIONS TABLE:")
213     algorithm_justification_table()
214
215     # Performance Analysis
216     performance_analysis()
217
218     if __name__ == "__main__":
219         main()

```

TERMINAL

```

=====
RETAIL STORE INVENTORY MANAGEMENT SYSTEM
=====

ID      Name       Price    Quantity
-----
P004    Mouse      $29.99    500
P003    Keyboard   $79.99    250
P005    Headphones $149.99   80
P002    Monitor    $299.99   120
P001    Laptop     $899.99   45

5. PRODUCTS SORTED BY QUANTITY (Descending):

ID      Name       Price    Quantity
-----
P004    Mouse      $29.99    500
P003    Keyboard   $79.99    250
P002    Monitor    $299.99   120
P005    Headphones $149.99   80
P001    Laptop     $899.99   45

6. LOW STOCK ALERT (< 100 units):

ID      Name       Price    Quantity
-----
P001    Laptop     $899.99   45
P005    Headphones $149.99   80

7. ALGORITHM RECOMMENDATIONS TABLE:



| OPERATION      | RECOMMENDED ALGO | TIME COMPLEXITY                                | JUSTIFICATION                      |
|----------------|------------------|------------------------------------------------|------------------------------------|
| Search by ID   | Hash Map (Dict)  | O(1) avg, O(n) worst                           | Instant lookups for large datasets |
| Search by Name | Hash Map (Dict)  | O(1) lookup + O(k) iteration where k = results | Fast name-based searches           |
| Sort by Price  | Timsort (Python) | O(n log n) worst, O(n) best case               | Optimized for                      |


```

PROBLEMS **OUTPUT** **DEBUG CONSOLE** **TERMINAL** **PORTS**

TERMINAL

```

Ln 1, Col 555 Spaces: 4 UTF-8 CRLF { } Python 3.14.2 Python 3.14.2

```

The screenshot shows a code editor interface with a dark theme. At the top, there's a navigation bar with File, Edit, Selection, View, Go, and a search bar. Below the navigation bar is a tab bar with PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS. On the far right of the tab bar are Python-related icons and a dropdown menu.

Table of Operations:

OPERATION	RECOMMENDED ALGO	TIME COMPLEXITY	JUSTIFICATION
Search by ID	Hash Map (Dict)	O(1) avg, O(n) worst	Instant lookups for large datasets
Search by Name	Hash Map (Dict)	O(1) lookup + O(k) iteration where k = results	Fast name-based searches
Sort by Price or Quantity	Timsort (Python)	O(n log n) worst, O(n) best case	Optimized for partially sorted real-world data
Add Product	Hash Map Insert	O(1) average case	Constant time insertion
Low Stock Filter	Linear Scan	O(n)	No better option without additional indices
Update Quantity	Hash Map Access	O(1) average case	Direct access via ID

Performance Analysis for 1000 Products:

```

=====
PERFORMANCE ANALYSIS FOR 1000 PRODUCTS
=====

Search by ID (100 searches)      : 0.157ms
Sort by Price (1000 products)    : 0.087ms
Sort by Quantity (1000 products) : 0.052ms
Low Stock Filter (1000 products): 0.033ms
PS C:\Users\lenovo>
  
```

Ln 1, Col 555 Spaces: 4 UTF-8 CRLF { } Python 3.14.2 Python

Prompt

Simulate 100 stock records (Symbol, Opening Price, Closing Price). Calculate percentage gain/loss. Use Heap Sort to rank stocks by percentage change. Use a Hash Map for instant symbol lookup. Compare performance with Python's built-in sorted() and analyze tradeoffs.

Output Explanation

The output will include:

- Simulated stock dataset.
- Calculated percentage gain/loss.
- Ranked list of stocks using Heap Sort.
- Runtime comparison:

Heap Sort Time: 0.004 sec

Built-in sorted() Time: 0.002 sec

What the Output Shows

- Heap Sort correctly ranks stocks by performance.
- Hash Map provides O(1) stock lookup.

- Built-in sorted() may perform faster due to internal optimizations.
- Shows trade-off between custom implementation and optimized library functions.

```

1 # Develop a Python program simulating an AI-powered stock analysis tool that generates or simulates stock data (Stock
2 import random
3 import time
4 import heapq
5 from typing import List, Dict, Tuple
6
7 class StockAnalysisTool:
8     def __init__(self, num_stocks: int = 100):
9         self.num_stocks = num_stocks
10    (method) def generate_stock_data(self: Self@StockAnalysisTool) -> None
11        Generate simulated stock data for analysis.
12    def generate_stock_data(self):
13        """Generate simulated stock data for analysis."""
14        symbols = [f'STOCK{i:03d}' for i in range(self.num_stocks)]
15        for symbol in symbols:
16            opening_price = round(random.uniform(10, 500), 2)
17            closing_price = round(opening_price * random.uniform(0.9, 1.1), 2)
18            percentage_change = ((closing_price - opening_price) / opening_price) * 100
19
20            self.stocks[symbol] = {
21                'opening': opening_price,
22                'closing': closing_price,
23                'percentage_change': round(percentage_change, 2)
24            }
25
26    def heap_sort_stocks(self) -> List[Tuple[str, float]]:
27
28    def heap_sort_stocks(self) -> List[Tuple[str, float]]:
29        """Sort stocks using Heap Sort (max heap) by percentage change."""
30        # Create max heap (negative values for max heap behavior)
31        heap = [(-stock_data['percentage_change'], symbol)
32               for symbol, stock_data in self.stocks.items()]
33        heapq.heapify(heap)
34
35        sorted_stocks = []
36        while heap:
37            neg_change, symbol = heapq.heappop(heap)
38            sorted_stocks.append((symbol, -neg_change))
39
40        return sorted_stocks
41
42    def builtin_sort_stocks(self) -> List[Tuple[str, float]]:
43        """Sort stocks using Python's built-in sorted() function."""
44        sorted_stocks = sorted(
45            self.stocks.items(),
46            key=lambda x: x[1]['percentage_change'],
47            reverse=True
48        )
49
50        return [(symbol, data['percentage_change']) for symbol, data in sorted_stocks]
51
52    def lookup_stock(self, symbol: str) -> Dict:
53        """Instant lookup using Hash Map."""
54        return self.stocks.get(symbol, None)

```

```
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
7   class StockAnalysisTool:
54     def benchmark_sorting(self):
55       """Compare performance of both sorting methods."""
56       # Heap Sort
57       start_time = time.perf_counter()
58       heap_result = self.heap_sort_stocks()
59       heap_time = time.perf_counter() - start_time
60
61       # Built-in Sort
62       start_time = time.perf_counter()
63       builtin_result = self.builtin_sort_stocks()
64       builtin_time = time.perf_counter() - start_time
65
66       return heap_time, builtin_time, heap_result, builtin_result
67
68     def display_analysis(self):
69       """Display analysis results and complexity explanation."""
70       print("=" * 70)
71       print("AI-POWERED STOCK ANALYSIS TOOL")
72       print("=" * 70)
73
74       # Generate data and sort
75       heap_time, builtin_time, heap_result, builtin_result = self.benchmark_sorting()
76
77       # Display top 10 stocks
78       print("\nTOP 10 STOCKS BY PERCENTAGE GAIN:")
79       print("=" * 70)
```

```
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
7   class StockAnalysisTool:
68     def display_analysis(self):
81       print(f"\n{i:2d}. {symbol}: {change:+.2f} | "
82           f"Open: ${self.stocks[symbol]['opening']:.2f} | "
83           f"Close: ${self.stocks[symbol]['closing']:.2f}")
84
85     # Display performances
86     print("\n" + "=" * 70)
87     print("PERFORMANCE COMPARISON:")
88     print("-" * 70)
89     print(f"Heap Sort Time: {heap_time * 1000:.4f} ms")
90     print(f"Built-in Sort Time: {builtin_time * 1000:.4f} ms")
91     print(f"Difference: {abs(heap_time - builtin_time) * 1000:.4f} ms")
92     print(f"Speed Ratio: {builtin_time / heap_time:.2f}x")
93
94     # Complexity Analysis
95     print("\n" + "=" * 70)
96     print("COMPLEXITY ANALYSIS:")
97     print("-" * 70)
98     print(f"Number of Stocks: {self.num_stocks}")
99     print("\nHEAP SORT:")
100    print(f"  Time Complexity: O(n log n) - {self.num_stocks} * log({self.num_stocks}) ≈ {self.num_stocks} * len(")
101    print(f"  Space Complexity: O(n)")
102    print(f"  Advantages:")
103    print(f"    - Guaranteed O(n log n) performance")
104    print(f"    - Suitable for real-time streaming data")
105    print(f"    - Can process data as it arrives")
106
107    print("\nBUILT-IN SORT (Timsort):")
108    print(f"  Time Complexity: O(n log n) average, O(n) best case")
109    print(f"  Space Complexity: O(n)")
110    print(f"  Advantages:")
111    print(f"    - Highly optimized for practical datasets")
```

```
C: > Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > ...
7   class StockAnalysisTool:
119     def display_analysis(self):
120       print(f"    - Instant symbol lookup (O(1))")
121       print(f"    - Essential for real-time systems")
122       print(f"    - Avoids need for binary search")
123
124       # Sample lookup
125       print("\n" + "=" * 70)
126       print("SAMPLE HASH MAP LOOKUP:")
127       print("-" * 70)
128       test_symbol = heap_result[0][0]
129       stock_data = self.lookup_stock(test_symbol)
130       print(f"Symbol: {test_symbol}")
131       print(f"Data: {stock_data}")
132       print(f"Lookup Time: O(1) - Instant access")
133
134       print("\n" + "=" * 70)
135       print("WHY THESE STRUCTURES FOR REAL-TIME SYSTEMS:")
136       print("-" * 70)
137       print("1. Hash Maps: Enables O(1) lookups - critical for handling")
138       print(" thousands of concurrent stock quote requests")
139       print("2. Heap Sort: Predictable O(n log n) performance without")
140       print(" worst-case degradation (unlike quicksort)")
141       print("3. Combined: Can maintain sorted rankings while allowing")
142       print(" instant individual stock access")
143       print("=" * 70)
144
145   if __name__ == "__main__":
146     tool = StockAnalysisTool(num_stocks=100)
147     tool.display_analysis()
```

```
Ln 1, Col 473 Spaces: 4 UTF-8 CRLF { } Python 3.14.2
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
=====
AI-POWERED STOCK ANALYSIS TOOL
=====

TOP 10 STOCKS BY PERCENTAGE GAIN:
-----
1. STOCK020: +9.99% | Open: $431.95 | Close: $475.10
2. STOCK074: +9.84% | Open: $47.54 | Close: $52.22
3. STOCK039: +9.56% | Open: $234.48 | Close: $256.90
4. STOCK014: +9.52% | Open: $235.42 | Close: $257.83
5. STOCK043: +9.44% | Open: $28.50 | Close: $31.19
6. STOCK068: +8.92% | Open: $108.10 | Close: $117.74
7. STOCK026: +8.60% | Open: $412.93 | Close: $448.45
8. STOCK057: +8.55% | Open: $181.53 | Close: $110.21
9. STOCK001: +8.40% | Open: $74.60 | Close: $80.87
10. STOCK040: +8.10% | Open: $74.57 | Close: $80.61

=====
PERFORMANCE COMPARISON:
-----
Heap Sort Time: 0.0875 ms
Built-in Sort Time: 0.0791 ms
Difference: 0.0084 ms
Speed Ratio: 0.90x

=====
COMPLEXITY ANALYSIS:
-----
Number of Stocks: 100

HEAP SORT:
Time Complexity: O(n log n) - 100 * log(100) ≈ 900 operations
Space Complexity: O(n)
Advantages:
- Guaranteed O(n log n) performance
- Suitable for real-time streaming data
- Can process data as it arrives
```

BUILT-IN SORT (Timsort):

- Time Complexity: $O(n \log n)$ average, $O(n)$ best case
- Space Complexity: $O(n)$
- Advantages:
 - Highly optimized for practical datasets
 - Adaptive to partially sorted data
 - Better cache locality

HASH MAP (Dictionary Lookup):

- Time Complexity: $O(1)$ average case
- Space Complexity: $O(n)$
- Advantages:
 - Instant symbol lookup ($O(1)$)
 - Essential for real-time systems
 - Avoids need for binary search

SAMPLE HASH MAP LOOKUP:

```
Symbol: STOCKK20
Data: {'opening': 431.95, 'closing': 475.1, 'percentage_change': 9.99}
Lookup Time: O(1) - Instant access
```

WHY THESE STRUCTURES FOR REAL-TIME SYSTEMS:

1. Hash Maps: Enables $O(1)$ lookups - critical for handling thousands of concurrent stock quote requests
2. Heap Sort: Predictable $O(n \log n)$ performance without worst-case degradation (unlike quicksort)
3. Combined: Can maintain sorted rankings while allowing instant individual stock access

```
PS C:\Users\lenovo>
```

AIAC Code.py

```
C:\> Users > lenovo > Desktop > 3-2 > AIAC > AIAC Code.py > Student
38
39
40
41
42
43
44
45
46
47
48
49
50
51
```

```
students = generate_students(1000)

# Sort by CGPA in descending order
sorted_students = sort_by_cgpa(students)

# Display top 10 students
print("Top 10 Students by CGPA:")
print("-" * 60)
for i, student in enumerate(sorted_students[:10], 1):
    print(f"{i}. {student.name} (Roll: {student.roll_number}) - CGPA: {student.cgpa}")

print("\n\nBottom 10 Students by CGPA:")
print("-" * 60)
for i, student in enumerate(sorted_students[-10:], 1):
    print(f"{i}. {student.name} (Roll: {student.roll_number}) - CGPA: {student.cgpa}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python 3.14.2

```
PS C:\Users\lenovo>

2. Neha Mishra (Roll: 352) - CGPA: 5.03
3. Neha Singh (Roll: 110) - CGPA: 5.02
4. Rohit Verma (Roll: 136) - CGPA: 5.02
5. Zara Sharma (Roll: 690) - CGPA: 5.02
6. Zara Sharma (Roll: 843) - CGPA: 5.02
7. Arjun Gupta (Roll: 859) - CGPA: 5.02
5. Zara Sharma (Roll: 690) - CGPA: 5.02
6. Zara Sharma (Roll: 843) - CGPA: 5.02
7. Arjun Gupta (Roll: 859) - CGPA: 5.02
8. Vivaan Reddy (Roll: 874) - CGPA: 5.02
9. Aditya Sharma (Roll: 983) - CGPA: 5.02
10. Priya Mishra (Roll: 688) - CGPA: 5.01
```

Ln 4, Col 15 Spaces: 4 UTF-8 CRLF {} Python 3.14.2 Python 3.14.2 Go Live

Conclusion

In this lab, we implemented and compared various sorting and searching algorithms using AI assistance. Quick Sort and Merge Sort were tested on large datasets and their performance differences were analyzed. Bubble Sort helped in understanding basic comparison-based sorting and its time complexity. Hash Maps demonstrated efficient $O(1)$ searching in real-time applications like inventory and stock systems.

Heap Sort was useful for ranking stock performance. Overall, the lab improved understanding of algorithm efficiency, data structure selection, and practical performance analysis while highlighting benefits of AI-assisted coding.