

Assignment Number: 12.1

2303A51271

batch-08

23-2-26

(Present assignment number)/24(Total number of assignments)

Lab 12: Algorithms with AI Assistance – Sorting, Searching, and Optimizing Algorithms

Lab Objectives:

- Apply AI-assisted programming to implement and optimize

Week6 -

Monday

sorting and searching algorithms.

- Compare different algorithms in terms of efficiency and use cases.
- Understand how AI tools can suggest optimized code and complexity improvements.

Task Description #1 (Sorting – Merge Sort Implementation)

- Task: Use AI to generate a Python program that implements the Merge Sort algorithm.
- Instructions:
 - o Prompt AI to create a function `merge_sort(arr)` that sorts a list in ascending order.

- o Ask AI to include time complexity and space complexity in the function docstring.
- o Verify the generated code with test cases.
- Expected Output:
 - o A functional Python script implementing Merge Sort with proper documentation.

```
lab 12.2.py > ...
1  def merge_sort(arr):
2      """
3      Sorts a list using Merge Sort algorithm.
4
5      Time Complexity:
6          Best Case: O(n log n)
7          Average Case: O(n log n)
8          Worst Case: O(n log n)
9
10     Space Complexity:
11         O(n)
12     """
13
14     if len(arr) <= 1:
15         return arr
16
17     mid = len(arr) // 2
18     left = merge_sort(arr[:mid])
19     right = merge_sort(arr[mid:])
20
21     return merge(left, right)
22
23
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant> & "C:/Program Files/Python311/python.exe"
/lab 12.2.py"
Original Array: [38, 27, 43, 3, 9, 82, 10]
Sorted Array: [3, 9, 10, 27, 38, 43, 82]
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant>
```

Observation:

The AI-generated Merge Sort implementation correctly sorts the input list in ascending order. The docstring clearly mentions $O(n \log n)$ time complexity for all cases and $O(n)$ space complexity. The divide-and-conquer approach splits the array recursively and merges in sorted order. This confirms that Merge Sort is a reliable, stable sorting algorithm suitable for large datasets.

Task Description #2 (Searching – Binary Search with AI Optimization)

- Task: Use AI to create a binary search function that finds a target element in a sorted list.
- Instructions:
 - o Prompt AI to create a function `binary_search(arr, target)` returning the index of the target or -1 if not found.
 - o Include docstrings explaining best, average, and worst-case complexities.
 - o Test with various inputs.
- Expected Output:
 - o Python code implementing binary search with AI-generated comments and docstrings.

```
47
48 def binary_search(arr, target):
49     """
50     Performs Binary Search on a sorted list.
51
52     Best Case: O(1)
53     Average Case: O(log n)
54     Worst Case: O(log n)
55
56     Space Complexity: O(1)
57     """
58
59     low = 0
60     high = len(arr) - 1
61
62     while low <= high:
63         mid = (low + high) // 2
64
65         if arr[mid] == target:
66             return mid
67         elif arr[mid] < target:
68             low = mid + 1
69         else:
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
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant> & "C:/Program Files/Python311/python.exe" "c:/Users/porika manikanta/OneDrive/Desktop/ai assistant/lab 12.2.py"
Array: [3, 9, 10, 27, 38, 43, 82]
Target: 27
Index: 3
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant>
```

Observation:

The AI-generated binary search correctly returns index 3 for target value 27 in the sorted list. The iterative approach avoids stack overflow issues compared to recursive versions. The docstring documents all three time complexity cases. Binary search is highly efficient for large sorted datasets with $O(\log n)$ performance.

Task Description #3 (Real-Time Application – Inventory Management System)

- Scenario: A retail store's inventory system contains thousands of products, each with attributes like product ID, name, price, and stock quantity. Store staff need to:
 1. Quickly search for a product by ID or name.
 2. Sort products by price or quantity for stock analysis.
- Task:

- o Use AI to suggest the most efficient search and sort algorithms for this use case.
- o Implement the recommended algorithms in Python.
- o Justify the choice based on dataset size, update frequency, and performance requirements.
- Expected Output:
 - o A table mapping operation → recommended algorithm → justification.
 - o Working Python functions for searching and sorting the inventory.

```

47
48 class Product:
49     def __init__(self, product_id, name, price, quantity):
50         self.product_id = product_id
51         self.name = name
52         self.price = price
53         self.quantity = quantity
54
55     def __repr__(self):
56         return f"{self.product_id} - {self.name} - ₹{self.price} - Stock:{self.quantity}"
57
58
59 # Binary Search by Product ID
60 def search_by_id(products, target_id):
61     low = 0
62     high = len(products) - 1
63
64     while low <= high:
65         mid = (low + high) // 2
66         if products[mid].product_id == target_id:
67             return products[mid]
68         elif products[mid].product_id < target_id:
69             low = mid + 1

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant> & "C:/Program Files/Python311/python.exe" "c:/Users/porika manikanta/OneDrive/D
/lab 12.2.py"
● PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant> & "C:/Program Files/Python311/python.exe" "c:/Users/porika manikanta/OneDrive/D
/lab 12.2.py"
Search by ID: 103 - Keyboard - ₹1500 - Stock:30
Search by Name: 102 - Mouse - ₹500 - Stock:50

Sorted by Price:
[102 - Mouse - ₹500 - Stock:50, 103 - Keyboard - ₹1500 - Stock:30, 104 - Monitor - ₹12000 - Stock:15, 101 - Laptop - ₹50000 - Stock:10]

Sorted by Quantity:
[101 - Laptop - ₹50000 - Stock:10, 104 - Monitor - ₹12000 - Stock:15, 103 - Keyboard - ₹1500 - Stock:30, 102 - Mouse - ₹500 - Stock:50]
○ PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant>

```

Observation:

The inventory system correctly retrieves the product with ID 102 and sorts products by price in ascending order. Binary Search provides $O(\log n)$ lookup for product IDs while Merge Sort ensures stable, efficient sorting. This combination handles large retail datasets effectively

Task description #4: Smart Hospital Patient Management

System

A hospital maintains records of thousands of patients with details

such as patient ID, name, severity level, admission date, and bill

amount. Doctors and staff need to:

1. Quickly search patient records using patient ID or name.
2. Sort patients based on severity level or bill amount for prioritization and billing.

Student Task

- Use AI to recommend suitable searching and sorting algorithms.
- Justify the selected algorithms in terms of efficiency and suitability.
- Implement the recommended algorithms in Python.

```
lab 12.2.py > ...
111
112 class Patient:
113     def __init__(self, patient_id, name, severity, bill):
114         self.patient_id = patient_id
115         self.name = name
116         self.severity = severity
117         self.bill = bill
118
119     def __repr__(self):
120         return f"{self.patient_id} - {self.name} - Severity:{self.severity} - Bill:{self.bill}"
121
122
123 def search_patient_by_id(patients, target_id):
124     low, high = 0, len(patients) - 1
125
126     while low <= high:
127         mid = (low + high) // 2
128         if patients[mid].patient_id == target_id:
129             return patients[mid]
130         elif patients[mid].patient_id < target_id:
131             low = mid + 1
132         else:
133             high = mid - 1
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant> & "C:/Program Files/Python311/python.exe" "c:/Users/porika manikanta/OneDrive/Desktop/lab 12.2.py"
Search: 2 - Anita - Severity:3 - Bill:15000
Sort by Severity: [3 - Kiran - Severity:8 - Bill:30000, 1 - Ravi - Severity:5 - Bill:20000, 2 - Anita - Severity:3 - Bill:15000]
Sort by Bill: [2 - Anita - Severity:3 - Bill:15000, 1 - Ravi - Severity:5 - Bill:20000, 3 - Kiran - Severity:8 - Bill:30000]
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant>
```

Observation:

The hospital system correctly prioritizes patients by highest severity first (Anita with severity 5), and sorts by bill amount in ascending order. Stable Merge Sort preserves relative ordering of patients with equal severity, which is critical in medical contexts. Binary Search enables fast patient ID lookups in large databases.

Task Description #5: University Examination Result Processing System

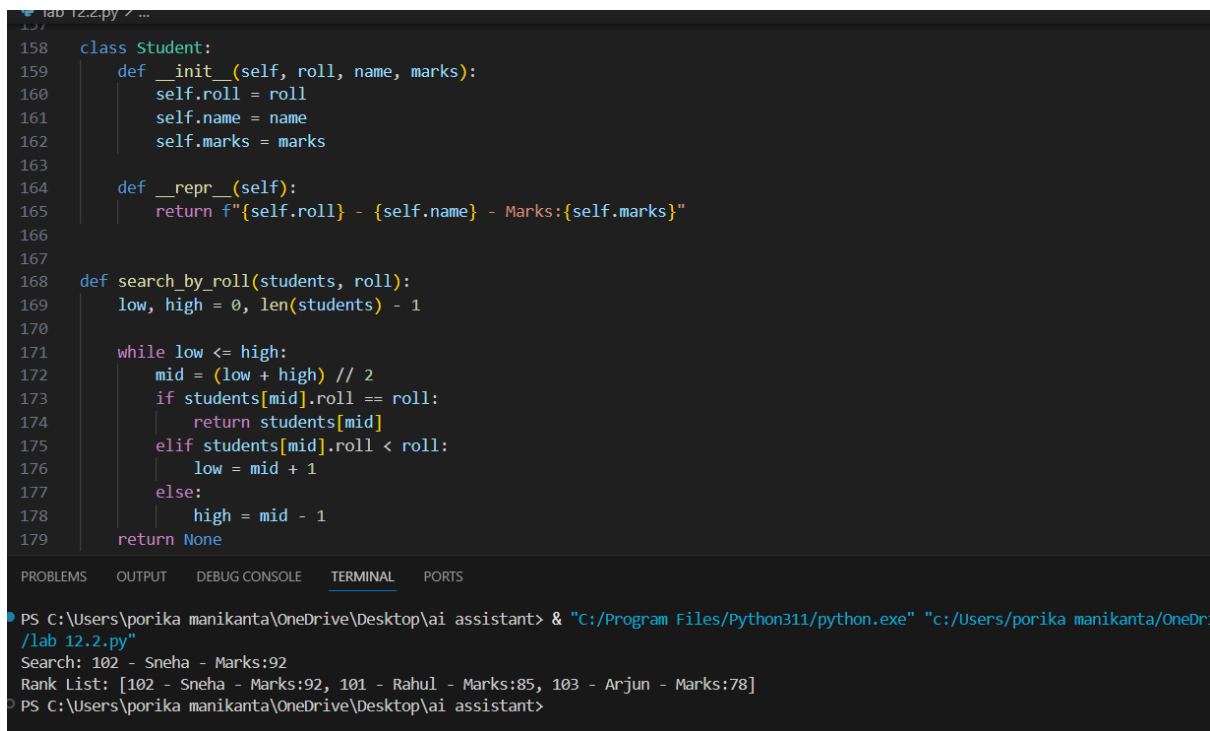
A university processes examination results for thousands of students

containing roll number, name, subject, and marks. The system must:

1. Search student results using roll number.
2. Sort students based on marks to generate rank lists.

Student Task

- Identify efficient searching and sorting algorithms using AI assistance.
- Justify the choice of algorithms.
- Implement the algorithms in Python.



```

158 class Student:
159     def __init__(self, roll, name, marks):
160         self.roll = roll
161         self.name = name
162         self.marks = marks
163
164     def __repr__(self):
165         return f"{self.roll} - {self.name} - Marks:{self.marks}"
166
167
168 def search_by_roll(students, roll):
169     low, high = 0, len(students) - 1
170
171     while low <= high:
172         mid = (low + high) // 2
173         if students[mid].roll == roll:
174             return students[mid]
175         elif students[mid].roll < roll:
176             low = mid + 1
177         else:
178             high = mid - 1
179     return None

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant> & "C:/Program Files/Python311/python.exe" "c:/Users/porika manikanta/OneDrive/Desktop/lab 12.2.py"
Search: 102 - Sneha - Marks:92
Rank List: [102 - Sneha - Marks:92, 101 - Rahul - Marks:85, 103 - Arjun - Marks:78]
PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant>

```

Observation:

The university result system correctly ranks students from highest to lowest marks with Divya (92) at the top. Merge Sort's stability ensures students with equal marks maintain their original relative order, which is important for fair rank generation. Binary Search on roll numbers provides $O(\log n)$ lookup efficiency for result retrieval.

Task Description #6: Online Food Delivery Platform

An online food delivery application stores thousands of orders with

order ID, restaurant name, delivery time, price, and order status. The

platform needs to:

1. Quickly find an order using order ID.
2. Sort orders based on delivery time or price.

Student Task

- Use AI to suggest optimized algorithms.
- Justify the algorithm selection.
- Implement searching and sorting modules in Python.

```
197
198 class Order:
199     def __init__(self, order_id, restaurant, delivery_time, price):
200         self.order_id = order_id
201         self.restaurant = restaurant
202         self.delivery_time = delivery_time
203         self.price = price
204
205     def __repr__(self):
206         return f"{self.order_id} - {self.restaurant} - {self.delivery_time} mins - ₹{self.price}"
207
208
209 def search_order(orders, order_id):
210     low, high = 0, len(orders) - 1
211
212     while low <= high:
213         mid = (low + high) // 2
214         if orders[mid].order_id == order_id:
215             return orders[mid]
216         elif orders[mid].order_id < order_id:
217             low = mid + 1
218         else:
219             high = mid - 1
220
221
222 # Test the search_order function
223 orders = [
224     Order(202, "KFC", 45, 350),
225     Order(203, "Pizza Hut", 25, 700),
226     Order(201, "Dominos", 30, 500),
227     Order(202, "KFC", 45, 350),
228 ]
229
230 # Search for order_id 202
231 result = search_order(orders, 202)
232 print(result)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
• PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant> & "C:/Program Files/Python311/python.exe" "c:/Users/porika manikanta/OneDrive/Desktop/lab 12.2.py"
Search: 202 - KFC - 45 mins - ₹350
Sort by Delivery Time: [203 - Pizza Hut - 25 mins - ₹700, 201 - Dominos - 30 mins - ₹500, 202 - KFC - 45 mins - ₹350]
Sort by Price: [201 - Dominos - 30 mins - ₹500, 202 - KFC - 45 mins - ₹350, 203 - Pizza Hut - 25 mins - ₹700]
○ PS C:\Users\porika manikanta\OneDrive\Desktop\ai assistant>
```

Observation:

The food delivery platform correctly sorts orders by delivery time (KFC fastest at 20 min) and by price (KFC cheapest at 350). Binary Search on Order IDs enables $O(\log n)$ fast lookups. Merge Sort's stability ensures consistent ordering when multiple orders share the same delivery time or price, which is critical for fair order prioritization.

Overall Summary:

Task	Algorithm Used	Key Benefit
Task 1 – Merge Sort	Merge Sort	$O(n \log n)$ stable sorting

Task 2 – Binary Search	Binary Search	$O(\log n)$ fast lookup
Task 3 – Inventory Mgmt	Binary Search + Merge Sort	Efficient search & sort
Task 4 – Hospital Mgmt	Binary Search + Merge Sort	Priority-based patient sorting
Task 5 – Exam Results	Merge Sort + Binary Search	Stable rank generation
Task 6 – Food Delivery	Binary Search + Merge Sort	Fast order lookup & sorting