

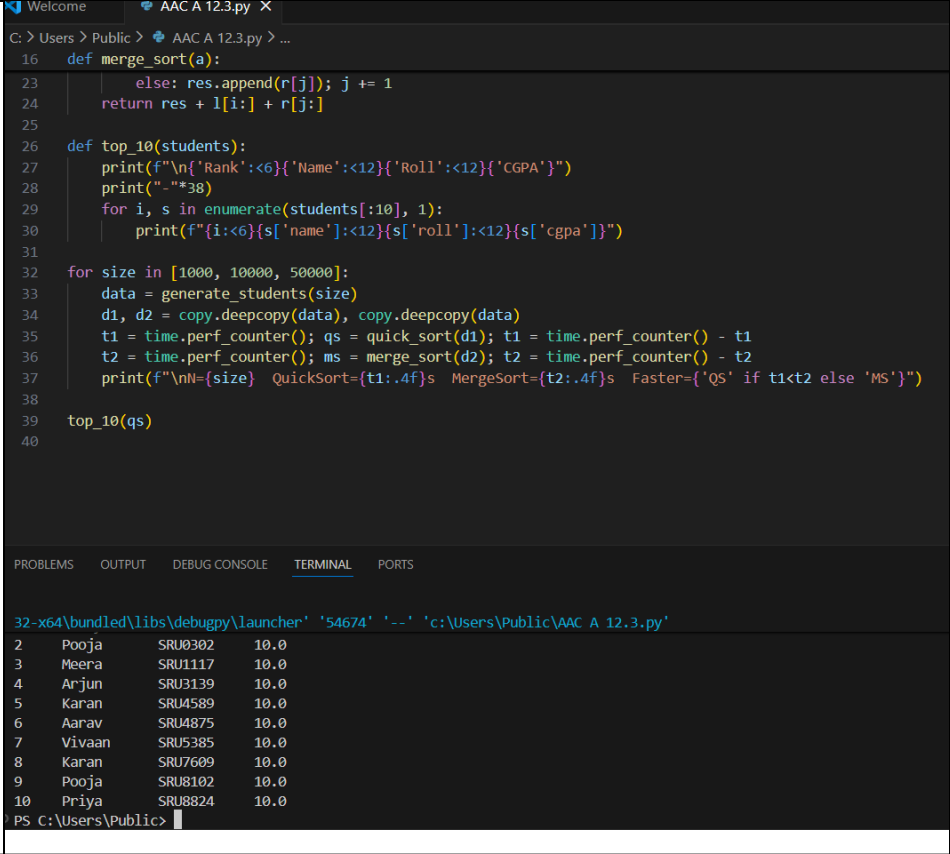
Name:I. Abhinay H.No:2303A51811 Batch:26

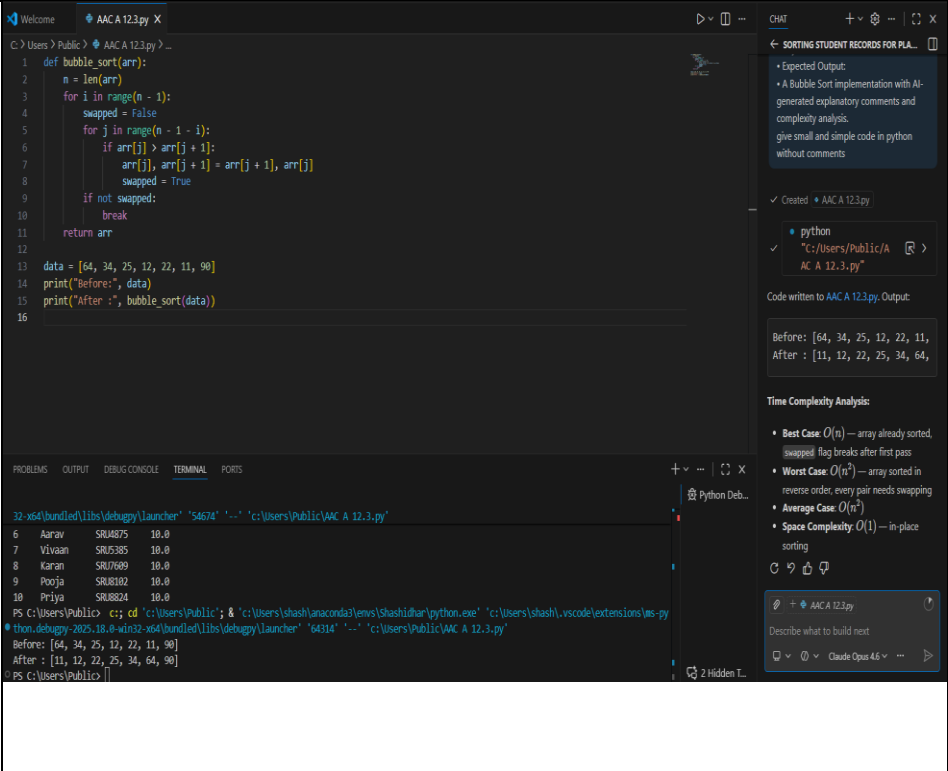
SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		Assignment Type: Lab	Academic Year:2025-2026
Course Coordinator Name		Dr. Rishabh Mittal	
Instructor(s) Name		Mr. S Naresh Kumar	
		Ms. B. Swathi	
		Dr. Sasanko Shekhar Gantayat	
		Mr. Md Sallauddin	
		Dr. Mathivanan	
		Mr. Y Srikanth	
		Ms. N Shilpa	
		Dr. Rishabh Mittal (Coordinator)	
		Dr. R. Prashant Kumar	
		Mr. Ankushavali MD	
		Mr. B Viswanath	
		Ms. Sujitha Reddy	
		Ms. A. Anitha	
		Ms. M.Madhuri	
		Ms. Katherashala Swetha	
		Ms. Velpula sumalatha	
Mr. Bingi Raju			
CourseCode	23CS002PC304	Course Title	AI Assisted Coding
Year/Sem	III/II	Regulation	R23
Date and Day of Assignment	Week6 – Wednesday	Time(s)	23CSBTB01 To 23CSBTB52
Duration	2 Hours	Applicable to Batches	All batches
Assignment Number:12.3(Present assignment number)/24(Total number of assignments)			
Q.No.	Question		Expected Time to complete
1	Lab 12: Algorithms with AI Assistance Sorting, Searching, and Algorithm Optimization Using AI Tools Lab Objectives		Week3 - Wednesday

	<p>The objectives of this laboratory exercise are to:</p> <ul style="list-style-type: none"> • Apply AI-assisted programming techniques to implement sorting and searching algorithms. • Analyze and compare algorithm efficiency using time and space complexity. • Understand how AI tools can suggest optimizations and alternative algorithmic approaches. • Strengthen problem-solving skills through real-world, data-driven scenarios. 	
	<p>Learning Outcomes After completing this lab, students will be able to:</p> <ul style="list-style-type: none"> • Implement and optimize classic algorithms using AI-assisted coding tools. • Compare multiple algorithms for the same problem and justify their selection. • Measure and analyze runtime performance using experimental data. • Critically review and refine AI-generated algorithmic solutions. 	
	<p>Task 1: Sorting Student Records for Placement Drive Scenario SR University's Training and Placement Cell needs to shortlist candidates efficiently during campus placements. Student records must be sorted by CGPA in descending order.</p> <p>Tasks</p> <ol style="list-style-type: none"> 1. Use GitHub Copilot to generate a program that stores student records (Name, Roll Number, CGPA). 2. Implement the following sorting algorithms using AI assistance: <ul style="list-style-type: none"> ○ Quick Sort ○ Merge Sort 3. Measure and compare runtime performance for large datasets. 4. Write a function to display the top 10 students based on CGPA. <p>Expected Outcome</p> <ul style="list-style-type: none"> • Correctly sorted student records. • Performance comparison between Quick Sort and Merge Sort. • Clear output of top-performing students. 	

WelcomeAAC A 12.3.py X

C:\Users\Public> AAC A 12.3.py > ...
1 import random, time, copy
2
3 def generate_students(n):
4 names = ["Aarav", "Vivaan", "Aditya", "Arjun", "Sai", "Ananya", "Diya", "Rohan", "Mehta", "Priya",
5 "Rahul", "Sneha", "Vikram", "Meera", "Karan", "Pooja", "Ishaan", "Riya", "Amit", "Navya"]
6 students = []
7 for i in range(1, n+1):
8 students.append({"name": random.choice(names), "roll": f"SRU{i:04d}", "cgpa": round(random.uniform(4.0, 10.0), 2)})
9 return students
10
11 def quick_sort(a):
12 if len(a) <= 1: return a
13 p = a[len(a)//2]["cgpa"]
14 return quick_sort([x for x in a if x["cgpa"] > p]) + [x for x in a if x["cgpa"] == p] + quick_sort([x for x in a if x["cgpa"] < p])
15
16 def merge_sort(a):
17 if len(a) <= 1: return a
18 m = len(a)//2
19 l, r = merge_sort(a[:m]), merge_sort(a[m:])
20 res, i, j = [], 0, 0
21 while i < len(l) and j < len(r):
22 if l[i]["cgpa"] >= r[j]["cgpa"]: res.append(l[i]); i += 1
23 else: res.append(r[j]); j += 1
24 return res + l[i:] + r[j:]
25
26 def top_10(students):
27 print(f"\n{'Rank':<6}{'Name':<12}{'Roll':<12}{'CGPA':<6}")
28 print("-"*38)
29 for i, s in enumerate(students[:10], 1):
30 print(f"{i:<6}{s['name']:<12}{s['roll']:<12}{s['cgpa']:<6}")
31
32 for size in [1000, 10000, 50000]:
33 data = generate_students(size)
34 d1, d2 = copy.deepcopy(data), copy.deepcopy(data)
35 t1 = time.perf_counter(); qs = quick_sort(d1); t1 = time.perf_counter() - t1
36 t2 = time.perf_counter(); ms = merge_sort(d2); t2 = time.perf_counter() - t2
37 print(f"nm={size} QuickSort={t1:.4f}s MergeSort={t2:.4f}s Faster={'QS' if t1<t2 else 'MS'}")
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
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
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
213

	 <p>The screenshot shows a Python IDE with a file named 'AAC A 12.3.py'. The code implements a merge sort algorithm and a function to generate student data. The execution output in the terminal shows a list of 10 students with their IDs, names, and CGPAs, all having a CGPA of 10.0.</p> <pre> 16 def merge_sort(a): 23 else: res.append(r[j]); j += 1 24 return res + l[i:] + r[j:] 25 26 def top_10(students): 27 print(f"\n{'Rank':<6}{'Name':<12}{'Roll':<12}{'CGPA'}") 28 print("-"*38) 29 for i, s in enumerate(students[:10], 1): 30 print(f"{i:<6}{s['name']:<12}{s['roll']:<12}{s['cgpa']}") 31 32 for size in [1000, 10000, 50000]: 33 data = generate_students(size) 34 d1, d2 = copy.deepcopy(data), copy.deepcopy(data) 35 t1 = time.perf_counter(); qs = quick_sort(d1); t1 = time.perf_counter() - t1 36 t2 = time.perf_counter(); ms = merge_sort(d2); t2 = time.perf_counter() - t2 37 print(f"\nN={size} QuickSort={t1:.4f}s MergeSort={t2:.4f}s Faster={'QS' if t1<t2 else 'MS'}") 38 39 top_10(qs) 40 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 32-x64\bundled\libs\debugpy\launcher '54674' '--' 'c:\Users\Public\AAC A 12.3.py' 2 Pooja SRU0302 10.0 3 Meera SRU1117 10.0 4 Arjun SRU3139 10.0 5 Karan SRU4589 10.0 6 Aarav SRU4875 10.0 7 Vivaan SRU5385 10.0 8 Karan SRU7609 10.0 9 Pooja SRU8102 10.0 10 Priya SRU8824 10.0 PS C:\Users\Public> </pre>	
	<p>Task 2: Implementing Bubble Sort with AI Comments</p> <ul style="list-style-type: none"> • Task: Write a Python implementation of Bubble Sort. • Instructions: <ul style="list-style-type: none"> • Students implement Bubble Sort normally. • Ask AI to generate inline comments explaining key logic (like swapping, passes, and termination). • Request AI to provide time complexity analysis. • Expected Output: <ul style="list-style-type: none"> • A Bubble Sort implementation with AI-generated explanatory comments and complexity analysis. 	

	 <p>The screenshot displays a VS Code editor with a Python file named 'AAC A 12.3.py'. The code implements a bubble sort algorithm. The initial list is [64, 34, 25, 12, 22, 11, 90]. After sorting, the list becomes [11, 12, 22, 25, 34, 64, 90]. A chat window on the right, titled 'SORTING STUDENT RECORDS FOR PLA...', provides an expected output and a time complexity analysis.</p> <p>Expected Output:</p> <ul style="list-style-type: none">A Bubble Sort implementation with AI-generated explanatory comments and complexity analysis.give small and simple code in python without comments <p>Time Complexity Analysis:</p> <ul style="list-style-type: none">Best Case: $O(n)$ — array already sorted.Worst Case: $O(n^2)$ — array sorted in reverse order, every pair needs swapping.Average Case: $O(n^2)$Space Complexity: $O(1)$ — in-place sorting. <p>Task 3: Quick Sort and Merge Sort Comparison</p> <ul style="list-style-type: none">Task: Implement Quick Sort and Merge Sort using recursion.Instructions:<ul style="list-style-type: none">Provide AI with partially completed functions for recursion.Ask AI to complete the missing logic and add docstrings.Compare both algorithms on random, sorted, and reverse-sorted lists.Expected Output:<ul style="list-style-type: none">Working Quick Sort and Merge Sort implementations.AI-generated explanation of average, best, and worst-case complexities.	

ACA A 12.3.py

```
1 import random, time
2 def quick_sort(arr):
3     if len(arr) <= 1:
4         return arr
5     pivot = arr[len(arr) // 2]
6     left = [x for x in arr if x < pivot]
7     mid = [x for x in arr if x == pivot]
8     right = [x for x in arr if x > pivot]
9     return quick_sort(left) + mid + quick_sort(right)
10
11 def merge_sort(arr):
12     if len(arr) <= 1:
13         return arr
14     mid = len(arr) // 2
15     left = merge_sort(arr[:mid])
16     right = merge_sort(arr[mid:])
17     result, i, j = [], 0, 0
18     while i < len(left) and j < len(right):
19         if left[i] <= right[j]:
20             result.append(left[i]); i += 1
21         else:
22             result.append(right[j]); j += 1
23     return result + left[i:] + right[j:]
24
25 def measure(func, data):
26     start = time.perf_counter()
27     func(data)
28     return time.perf_counter() - start
29
30 n = 10000
31 cases = {
32     "Random": random.sample(range(n), n),
33     "Sorted": list(range(n)),
34     "Reversed": list(range(n, 0, -1))
35 }
36 print(f"Case: {c12} | Quick Sort: {c14} | Merge Sort: {c14}")
37 print("-" * 40)
38 for name, data in cases.items():
39     qt = measure(quick_sort, data)
40     mt = measure(merge_sort, data)
41     print(f"Case: {c12} | qt: {qt:.4f}s | mt: {mt:.4f}s")
42
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

Python Deb...

PS C:\Users\Public> cd 'C:\Users\Public'; & 'C:\Users\shash\anaconda3\envs\shashidhar\python.exe' 'C:\Users\shash\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '64314' '-' 'C:\Users\Public\ACA A 12.3.py'
After: [11, 12, 22, 25, 34, 64, 90]
PS C:\Users\Public> cd 'C:\Users\Public'; & 'C:\Users\shash\anaconda3\envs\shashidhar\python.exe' 'C:\Users\shash\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '51664' '-' 'C:\Users\Public\ACA A 12.3.py'
Case Quick Sort Merge Sort
Random 0.0159s 0.0238s
Sorted 0.0105s 0.0143s
Reversed 0.0111s 0.0182s

CHAT

Sorting Student Records for PLA...

Functions for recursion.

Ask AI to complete the missing logic and add docstrings.

Compare both algorithms on random, sorted, and reverse-sorted lists.

Expected Output:

Working Quick Sort and Merge Sort implementations.

AI-generated explanation of average, best, and worst-case complexities.

give simple and small code in python

Reviewed and created ACA A 12.3.py

Created ACA A 12.3.py

python

C:\Users\Public\ACA A 12.3.py

Code written to ACA A 12.3.py. Output:

Case Quick Sort Merge

Random 0.0315s 0.03

Sorted 0.0204s 0.02

Reversed 0.0263s 0.03

Complexity Analysis:

Describe what to build next

Claude Opus 4.5

ACA A 12.3.py

```
10 def merge_sort(arr):
11     result.append(right[j]); j += 1
12     return result + left[i:] + right[j:]
13
14 def measure(func, data):
15     start = time.perf_counter()
16     func(data)
17     return time.perf_counter() - start
18
19 n = 10000
20 cases = {
21     "Random": random.sample(range(n), n),
22     "Sorted": list(range(n)),
23     "Reversed": list(range(n, 0, -1))
24 }
25 print(f"Case: {c12} | Quick Sort: {c14} | Merge Sort: {c14}")
26 print("-" * 40)
27 for name, data in cases.items():
28     qt = measure(quick_sort, data)
29     mt = measure(merge_sort, data)
30     print(f"Case: {c12} | qt: {qt:.4f}s | mt: {mt:.4f}s")
31
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

Python Deb...

PS C:\Users\Public> cd 'C:\Users\Public'; & 'C:\Users\shash\anaconda3\envs\shashidhar\python.exe' 'C:\Users\shash\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '64314' '-' 'C:\Users\Public\ACA A 12.3.py'
After: [11, 12, 22, 25, 34, 64, 90]
PS C:\Users\Public> cd 'C:\Users\Public'; & 'C:\Users\shash\anaconda3\envs\shashidhar\python.exe' 'C:\Users\shash\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundle\libs\debugpy\launcher' '51664' '-' 'C:\Users\Public\ACA A 12.3.py'
Case Quick Sort Merge Sort
Random 0.0159s 0.0238s
Sorted 0.0105s 0.0143s
Reversed 0.0111s 0.0182s

CHAT

Sorting Student Records for PLA...

Code written to ACA A 12.3.py. Output:

Case Quick Sort Merge

Random 0.0315s 0.03

Sorted 0.0204s 0.02

Reversed 0.0263s 0.03

Complexity Analysis:

Quick Sort Merge Sort

Best $O(n \log n)$ $O(n \log n)$

Ave rag e $O(n \log n)$ $O(n \log n)$

Wor st $O(n^2)$ (sorted input, bad pivot) $O(n \log n)$

Spa ce $O(\log n)$ (call stack) $O(n)$ (merge buffer)

Quick Sort is generally faster in practice due to better cache locality, but Merge Sort guarantees $O(n \log n)$ in all cases.

Claude Opus 4.5

Describe what to build next

Claude Opus 4.5

Task 4 (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:

- Quickly search for a product by ID or name.
- Sort products by price or quantity for stock analysis.

Task:

- Use AI to suggest the most efficient search and sort algorithms for this use case.

- Implement the recommended algorithms in Python.
- Justify the choice based on dataset size, update frequency, and performance requirements.

Expected Output:

- A table mapping operation → recommended algorithm → justification.
- Working Python functions for searching and sorting the inventory.

```

1 import random, time
2
3 def generate_inventory(n):
4     return [{"id": i, "name": f"Product_{i}", "price": round(random.uniform(10, 5000), 2),
5             "qty": random.randint(0, 500)} for i in range(1, n + 1)]
6
7 def binary_search_by_id(products, target_id):
8     lo, hi = 0, len(products) - 1
9     while lo <= hi:
10         mid = (lo + hi) // 2
11         if products[mid]["id"] == target_id:
12             return products[mid]
13         elif products[mid]["id"] < target_id:
14             lo = mid + 1
15         else:
16             hi = mid - 1
17     return None
18
19 def hash_search_by_name(index, name):
20     return index.get(name)
21
22 def build_name_index(products):
23     return {p["name"]: p for p in products}
24
25
26 PS C:\Users\Public> cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\shashidhar\python.exe' 'c:\Users\shash\vscode\extensions\ms-python.debugpy-2025.18.8.win32-x64\debugpy\launcher' '57500' '-.' 'c:\Users\Public\AAC A 12.3.py'
== Search by ID (Binary Search) ==
Found: {'id': 5000, 'name': 'Product_5000', 'price': 2625.52, 'qty': 428} Time: 0.000007s
== Search by name (Hash Map) ==
Found: {'id': 5000, 'name': 'Product_5000', 'price': 2625.52, 'qty': 428} Time: 0.000002s
== Sort by Price (Quick sort) ==
Time: 0.00000s
ID Name Price Qty

```

Operation	Algorithm	Justification
Search by ID	Binary Search	$O(\log n)$ — IDs are sequential/sorted, minimal comparisons
Search by Name	Hash Map (dict)	$O(1)$ average — Instant lookup, ideal for frequent name queries
Sort by Price/Qty	Quick Sort	$O(n \log n)$ average — fast in practice, good cache locality

Why these choices for Inventory:

Describe what to build next

Welcome

AAC A 12.3.py X

C: > Users > Public > AAC A 12.3.py > ...

```

22 def build_name_index(products):
23     return {p["name"]: p for p in products}
24
25 def quick_sort(arr, key):
26     if len(arr) <= 1:
27         return arr
28     pivot = arr[len(arr) // 2][key]
29     left = [x for x in arr if x[key] < pivot]
30     mid = [x for x in arr if x[key] == pivot]
31     right = [x for x in arr if x[key] > pivot]
32     return quick_sort(left, key) + mid + quick_sort(right, key)
33
34 def display(products, n=10):
35     print(f"{'ID':<8}{ 'Name':<16}{ 'Price':<12}{ 'Qty':<6}")
36     print("-" * 42)
37     for p in products[:n]:
38         print(f"{p['id']:<8}{p['name']:<16}{p['price']:<12}{p['qty']:<6}")
39
40 inventory = generate_inventory(10000)
41
42 print("=== Search by ID (Binary Search) ===")
43 sorted_by_id = sorted(inventory, key=lambda x: x["id"])
44 start = time.perf_counter()

```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

PS C:\Users\Public> c::; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57920' '--' 'c:\Users\Publ

ID	Name	Price	Qty
3623	Product_3623	10.64	206
9538	Product_9538	10.67	14
122	Product_122	11.15	333
2235	Product_2235	11.41	386
6376	Product_6376	11.61	320
5937	Product_5937	11.75	324
3876	Product_3876	11.77	356

Welcome

AAC A 12.3.py X

C: > Users > Public > AAC A 12.3.py > ...

```
42 print("=== Search by ID (Binary Search) ===")
43 sorted_by_id = sorted(inventory, key=lambda x: x["id"])
44 start = time.perf_counter()
45 result = binary_search_by_id(sorted_by_id, 5000)
46 t1 = time.perf_counter() - start
47 print(f"Found: {result} Time: {t1:.6f}s")
48
49 print("\n=== Search by Name (Hash Map) ===")
50 name_index = build_name_index(inventory)
51 start = time.perf_counter()
52 result = hash_search_by_name(name_index, "Product_5000")
53 t2 = time.perf_counter() - start
54 print(f"Found: {result} Time: {t2:.6f}s")
55
56 print("\n=== Sort by Price (Quick Sort) ===")
57 start = time.perf_counter()
58 sorted_price = quick_sort(inventory, "price")
59 t3 = time.perf_counter() - start
60 print(f"Time: {t3:.4f}s")
61 display(sorted_price)
62
63 print("\n=== Sort by Quantity (Quick Sort) ===")
64 start = time.perf_counter()
65 sorted_qty = quick_sort(inventory, "qty")
66 t4 = time.perf_counter() - start
67 print(f"Time: {t4:.4f}s")
68 display(sorted_qty)
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

```
PS C:\Users\Public> c:; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda\
thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57920'
```

122	Product_122	11.15	333
2235	Product_2235	11.41	386
6376	Product_6376	11.61	320
5937	Product_5937	11.75	324
3876	Product_3876	11.77	356
6326	Product_6326	12.62	235
8471	Product_8471	13.08	105
2856	Product_2856	13.3	141

Welcome AAC A 12.3.py X

C: > Users > Public > AAC A 12.3.py > ...

```
57 start = time.perf_counter()
58 sorted_price = quick_sort(inventory, "price")
59 t3 = time.perf_counter() - start
60 print(f"Time: {t3:.4f}s")
61 display(sorted_price)
62
63 print("\n=== Sort by Quantity (Quick Sort) ===")
64 start = time.perf_counter()
65 sorted_qty = quick_sort(inventory, "qty")
66 t4 = time.perf_counter() - start
67 print(f"Time: {t4:.4f}s")
68 display(sorted_qty)
69
70 print("\n=== Algorithm Recommendation Table ===")
71 print(f'{"Operation":<25}{"Algorithm":<20}{"Justification"}')
72 print("-" * 75)
73 print(f'{"Search by ID":<25}{"Binary Search":<20}{"O(log n), data sorted by ID"}')
74 print(f'{"Search by Name":<25}{"Hash Map (dict)":<20}{"O(1) average lookup"}')
75 print(f'{"Sort by Price/Qty":<25}{"Quick Sort":<20}{"O(n log n) avg, fast in practice"}')
76
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Public> c:; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\Public\AAC A 12.3.py'

```
=== Sort by Quantity (Quick Sort) ===
Time: 0.0557s
ID      Name      Price    Qty
-----
85      Product_85  4139.68  0
441     Product_441 4613.7   0
1338    Product_1338 3868.26  0
1408    Product_1408 56.08    0
```

Welcome AAC A 12.3.py X

C:\Users\Public> AAC A 12.3.py > ...

```
start = time.perf_counter()
58 sorted_price = quick_sort(inventory, "price")
59 t3 = time.perf_counter() - start
60 print(f"Time: {t3:.4f}s")
61 display(sorted_price)
62
63 print("\n=== Sort by Quantity (Quick Sort) ===")
64 start = time.perf_counter()
65 sorted_qty = quick_sort(inventory, "qty")
66 t4 = time.perf_counter() - start
67 print(f"Time: {t4:.4f}s")
68 display(sorted_qty)
69
70 print("\n=== Algorithm Recommendation Table ===")
71 print(f"{'Operation':<25}{'Algorithm':<20}{'Justification'}")
72 print("-" * 75)
73 print(f"{'Search by ID':<25}{'Binary Search':<20}{'O(log n), data sorted by ID'}")
74 print(f"{'Search by Name':<25}{'Hash Map (dict)':<20}{'O(1) average lookup'}")
75 print(f"{'Sort by Price/Qty':<25}{'Quick Sort':<20}{'O(n log n) avg, fast in practice'}")
76
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Public> c:: cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe'
thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57920' '--' 'c:\Users\Public\AAC A 12

441 Product_441 4613.7 0
1338 Product_1338 3868.26 0
1408 Product_1408 56.08 0
1740 Product_1740 660.14 0
2077 Product_2077 2768.81 0
2597 Product_2597 299.16 0
2837 Product_2837 3044.18 0
3438 Product_3438 3553.94 0
3474 Product_3474 3410.59 0

```
C:\Users\Public> AAC A 12.3.py > ...
57 start = time.perf_counter()
58 sorted_price = quick_sort(inventory, "price")
59 t3 = time.perf_counter() - start
60 print(f"Time: {t3:.4f}s")
61 display(sorted_price)
62
63 print("\n=== Sort by Quantity (Quick Sort) ===")
64 start = time.perf_counter()
65 sorted_qty = quick_sort(inventory, "qty")
66 t4 = time.perf_counter() - start
67 print(f"Time: {t4:.4f}s")
68 display(sorted_qty)
69
70 print("\n=== Algorithm Recommendation Table ===")
71 print(f"{'Operation':<25}{'Algorithm':<20}{'Justification'}")
72 print("-" * 75)
73 print(f"{'Search by ID':<25}{'Binary Search':<20}{'O(log n), data sorted by ID'}")
74 print(f"{'Search by Name':<25}{'Hash Map (dict)':<20}{'O(1) average lookup'}")
75 print(f"{'Sort by Price/Qty':<25}{'Quick Sort':<20}{'O(n log n) avg, fast in practice'}")
76
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Public> c;; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' '
thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57920' '--' 'c:\Users\Public\AAC A 12.3
3474 Product_3474 3410.59 0

=== Algorithm Recommendation Table ===
Operation                Algorithm                Justification
-----
Search by ID             Binary Search           O(log n), data sorted by ID
Search by Name           Hash Map (dict)         O(1) average lookup
Sort by Price/Qty        Quick Sort              O(n log n) avg, fast in practice
PS C:\Users\Public>
```

Task 5: Real-Time Stock Data Sorting & Searching

Scenario:

An AI-powered **FinTech Lab** at SR University is building a tool for analyzing **stock price movements**. The requirement is to quickly **sort stocks by daily gain/loss** and search for specific stock symbols efficiently.

- Use **GitHub Copilot** to fetch or simulate stock price data (Stock Symbol, Opening Price, Closing Price).
- Implement sorting algorithms to rank stocks by **percentage change**.
- Implement a **search function** that retrieves stock data instantly when a stock symbol is entered.
- Optimize sorting with **Heap Sort** and searching with **Hash Maps**.
- Compare performance with standard library functions (sorted(), dict lookups) and analyze trade-offs.

WelcomeAAC A 12.3.py X

C:\Users > Public > AAC A 12.3.py > ...
1 import random, time, heapq
2
3 symbols = ["AAPL", "GOOG", "MSFT", "AMZN", "ISLA", "META", "NFLX", "NVDA", "INCY", "TCS",
4 "RELIANCE", "HDFC", "WIPRO", "KCLTECH", "SBIN", "LICICI", "BAJAJ", "SUNPHARMA",
5 "LT", "MARUTI", "TATAMOTORS", "ITC", "AXISBANK", "KOTAK", "ADANI"]
6
7 def generate_stocks(n):
8 stocks = []
9 for i in range(n):
10 sym = symbols[i % len(symbols)] + (f"{i//25}" if i >= 25 else "")
11 open_p = round(random.uniform(100, 5000), 2)
12 close_p = round(open_p * random.uniform(0.9, 1.1), 2)
13 pct = round((close_p - open_p) / open_p * 100, 2)
14 stocks.append({"symbol": sym, "open": open_p, "close": close_p, "pct_change": pct})
15 return stocks
16
17 def heap_sort_by_pct(stocks):
18 heap = [(-s["pct_change"], i, s) for i, s in enumerate(stocks)]
19 heapq.heapify(heap)
20 return [heapq.heappop(heap)[2] for _ in range(len(heap))]
21
22 def hash_search(index, symbol):
23 return index.get(symbol)
24

PROBLEMSOUTPUTDEBUG CONSOLETERMINALPORTS
PS C:\Users\Public> c;; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\.vscode\extensions\ms-py
thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58347' '-.' 'c:\Users\Public\AAC A 12.3.py'
=== Heap Sort vs sorted() - Rank by % Change ===

Heap Sort : 0.0091s
sorted() : 0.0018s
Faster : sorted()

Top 10 Gainers:
SymbolOpenCloseChange %

NFLX_1192147.642362.279.99
AXISBANK_149313.62344.949.99
META_235160.33175.349.99

CHAT
SORTING STUDENT RECORDS FOR PL...
stock price data (Stock Symbol, Opening Price, Closing Price).
• Implement sorting algorithms to rank stocks by percentage change.
• Implement a search function that retrieves stock data instantly when a stock symbol is entered.
• Optimize sorting with Heap Sort and searching with Hash Maps.
• Compare performance with standard library functions (sorted(), dict lookups) and analyze trade-offs.
give simple and small code in python
✓ Reviewed AAC A 12.3.py
✓ Created AAC A 12.3.py
python
✓ "c:\Users\Public\AAC A 12.3.py"
Code written to AAC A 12.3.py. Results:
Sorting (10,000 stocks by % change):
• Heap Sort: 0.0056s
• sorted() (Timsort): 0.0030s — faster because Timsort is a highly optimized hybrid algorithm
AAC A 12.3.py
Describe what to build next
Claude Opus 4.5

WelcomeAAC A 12.3.py X

C:\Users > Public > AAC A 12.3.py > ...
28 def display(stocks, n=10):
32 print(f"{s['symbol']:<15}{s['open']:<12}{s['close']:<12}{s['pct_change']:<10}")
33
34 data = generate_stocks(10000)
35
36 print("=== Heap Sort vs sorted() - Rank by % Change ===\n")
37
38 start = time.perf_counter()
39 heap_sorted = heap_sort_by_pct(data)
40 t_heap = time.perf_counter() - start
41
42 start = time.perf_counter()
43 lib_sorted = sorted(data, key=lambda x: -x["pct_change"])
44 t_lib = time.perf_counter() - start
45
46 print(f"Heap Sort : {t_heap:.4f}s")
47 print(f"sorted() : {t_lib:.4f}s")
48 print(f"Faster : {'Heap Sort' if t_heap < t_lib else 'sorted()'}\n")
49
50 print("Top 10 Gainers:")
51 display(heap_sorted)
52 print("\nTop 10 Losers:")
53 display(heap_sorted[-10:][::-1])
54

PROBLEMSOUTPUTDEBUG CONSOLETERMINALPORTS
PS C:\Users\Public> c;; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\.vscode\extensions\m
thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58347' '-.' 'c:\Users\Public\AAC A 12.3.py'

NFLX_1192147.642362.279.99
AXISBANK_149313.62344.949.99
META_235160.33175.349.99

Welcome AAC A 12.3.py X

C: > Users > Public > AAC A 12.3.py > ...

```
28 def display(stocks, n=10):
32     print(f"{s['symbol']:<15}{s['open']:<12}{s['close']:<12}{s['pct_change']:<10}")
33
34     data = generate_stocks(10000)
35
36     print("=== Heap Sort vs sorted() - Rank by % Change ===\n")
37
38     start = time.perf_counter()
39     heap_sorted = heap_sort_by_pct(data)
40     t_heap = time.perf_counter() - start
41
42     start = time.perf_counter()
43     lib_sorted = sorted(data, key=lambda x: -x["pct_change"])
44     t_lib = time.perf_counter() - start
45
46     print(f"Heap Sort : {t_heap:.4f}s")
47     print(f"sorted() : {t_lib:.4f}s")
48     print(f"Faster : {'Heap Sort' if t_heap < t_lib else 'sorted()'}\n")
49
50     print("Top 10 Gainers:")
51     display(heap_sorted)
52     print("\nTop 10 Losers:")
53     display(heap_sorted[-10:][::-1])
54
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Public> c;; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' thon_debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher '58347' '--' 'c:\Users\Public\AAC A 12

NFLX_119	2147.64	2362.27	9.99
AXISBANK_149	313.62	344.94	9.99
META_225	160.32	176.34	9.99

Top 10 Losers:

Symbol	Open	Close	Change %
ITC_339	4372.67	3935.42	-10.0
BAJAJ_316	3547.91	3193.18	-10.0

VS Welcome AAC A 12.3.py X

C: > Users > Public > AAC A 12.3.py > ...

```
28 def display(stocks, n=10):
32     print(f"{s['symbol']:<15}{s['open']:<12}{s['close']:<12}{s['pct_change']:<10}")
33
34     data = generate_stocks(10000)
35
36     print("=== Heap Sort vs sorted() - Rank by % Change ===\n")
37
38     start = time.perf_counter()
39     heap_sorted = heap_sort_by_pct(data)
40     t_heap = time.perf_counter() - start
41
42     start = time.perf_counter()
43     lib_sorted = sorted(data, key=lambda x: -x["pct_change"])
44     t_lib = time.perf_counter() - start
45
46     print(f"Heap Sort : {t_heap:.4f}s")
47     print(f"sorted() : {t_lib:.4f}s")
48     print(f"Faster : {'Heap Sort' if t_heap < t_lib else 'sorted()'}\n")
49
50     print("Top 10 Gainers:")
51     display(heap_sorted)
52     print("\nTop 10 Losers:")
53     display(heap_sorted[-10:][::-1])
54
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Public> c::; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe'
thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58347' '--' 'c:\Users\Public\AAC A 12.

ADANI_258	3536.14	3182.58	-10.0
HCLTECH_254	3068.46	2761.7	-10.0
SUNPHARMA_222	3697.36	3327.72	-10.0
MARUTI_93	1783.28	1604.99	-10.0
WIPRO_334	861.71	775.65	-9.99
ITC_327	2201.07	1981.19	-9.99
GOOG_285	2888.79	2600.15	-9.99
NFLX_256	2957.21	2661.79	-9.99

Welcome AAC A 12.3.py X

C: > Users > Public > AAC A 12.3.py > ...

```
49
50 print("Top 10 Gainers:")
51 display(heap_sorted)
52 print("\nTop 10 Losers:")
53 display(heap_sorted[-10:][::-1])
54
55 print("\n=== Hash Map vs dict lookup - Search by Symbol ===\n")
56
57 index = build_hash_index(data)
58
59 start = time.perf_counter()
60 r1 = hash_search(index, "TSLA")
61 t_hash = time.perf_counter() - start
62
63 start = time.perf_counter()
64 r2 = next((s for s in data if s["symbol"] == "TSLA"), None)
65 t_linear = time.perf_counter() - start
66
67 print(f"Hash Map Search : {t_hash:.6f}s Result: {r1}")
68 print(f"Linear Search : {t_linear:.6f}s Result: {r2}")
69 print(f"Faster : {'Hash Map' if t_hash < t_linear else 'Linear'}")
70
71 print("\n=== Performance Comparison Table ===")
72 print(f"{'Operation':<20}{ 'Custom':<18}{ 'Standard Lib':<18}{ 'Trade-off'}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Public> c;; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58347' '--' 'c:\Users\Public\AAC A 12.3.py'
```

=== Hash Map vs dict lookup - Search by Symbol ===

```
Hash Map Search : 0.000003s Result: {'symbol': 'TSLA', 'open': 3265.84, 'close': 2963.07, 'pct_change': -9.27}
Linear Search : 0.000013s Result: {'symbol': 'TSLA', 'open': 3265.84, 'close': 2963.07, 'pct_change': -9.27}
Faster : Hash Map
```

=== Performance Comparison Table ===

Operation	Custom	Standard Lib	Trade-off
-----------	--------	--------------	-----------

		Start	
--	--	-------	--

Welcome AAC A 12.3.py X

C: > Users > Public > AAC A 12.3.py > ...

```
60 r1 = hash_search(index, "TSLA")
61 t_hash = time.perf_counter() - start
62
63 start = time.perf_counter()
64 r2 = next((s for s in data if s["symbol"] == "TSLA"), None)
65 t_linear = time.perf_counter() - start
66
67 print(f"Hash Map Search : {t_hash:.6f}s Result: {r1}")
68 print(f"Linear Search : {t_linear:.6f}s Result: {r2}")
69 print(f"Faster : {'Hash Map' if t_hash < t_linear else 'Linear'}")
70
71 print("\n=== Performance Comparison Table ===")
72 print(f"{'Operation':<20}{ 'Custom':<18}{ 'Standard Lib':<18}{ 'Trade-off'}")
73 print(f"{'-' * 75}")
74 print(f"{'Sort by % change':<20}{ 'Heap Sort':<18}{ 'sorted()':<18}{ 'Timsort is hybrid, often faster'}")
75 print(f"{'Search by symbol':<20}{ 'Hash Map O(1)':<18}{ 'Linear O(n)':<18}{ 'Hash uses more memory'}")
76
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\Public> c;; cd 'c:\Users\Public'; & 'c:\Users\shash\anaconda3\envs\Shashidhar\python.exe' 'c:\Users\shash\thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58347' '--' 'c:\Users\Public\AAC A 12.3.py'
```

```
Linear Search : 0.000013s Result: {'symbol': 'TSLA', 'open': 3265.84, 'close': 2963.07, 'pct_change': -9.27}
Faster : Hash Map
```

=== Performance Comparison Table ===

Operation	Custom	Standard Lib	Trade-off
Sort by % change	Heap Sort	sorted()	Timsort is hybrid, often faster
Search by symbol	Hash Map O(1)	Linear O(n)	Hash uses more memory

PS C:\Users\Public>

	<p>Note: Report should be submitted as a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots.</p>	
--	---	--