

set10.py

```

1 def merge_sort(arr):
2     if len(arr) > 1:
3         mid = len(arr) // 2
4         left = arr[:mid]
5         right = arr[mid:]
6         print(f"Divide: {arr} -> {left} and {right}")
7         left = merge_sort(left)
8         right = merge_sort(right)
9         return merge(left, right)
10    return arr
11
12 def merge(left, right):
13     s = []
14     i = j = 0
15     comparisons = []
16     while i < len(left) and j < len(right):
17         if left[i] < right[j]:
18             comparisons.append(f"{left[i]}<{right[j]}")
19             s.append(left[i])
20             i += 1
21         else:
22             comparisons.append(f"{left[i]}>{right[j]}")
23             s.append(right[j])
24             j += 1
25     while i < len(left):
26         s.append(left[i])
27         if j > 0:
28             comparisons.append(f"{left[i]}>{right[j-1]}")
29             i += 1
30     while j < len(right):
31         s.append(right[j])
32         if i > 0:
33             comparisons.append(f"{right[j]}>{left[i-1]}")
34             j += 1
35     print(f"Merge: {left} and {right} -> {s} (comparisons: {', '.join(comparisons)})")
36     return s
37
38 arr = [38, 27, 43, 3, 9, 82, 10]
39 print("Merge Sort Steps:")
40 res = merge_sort(arr)
41 print(f"Sorted array: {res}")
42 print("Time complexity: O(n log n), Space: O(n)")
43
44
45
46 def partition(arr, low, high):
47     pivot = arr[high]
48     i = low - 1

```

```
49     comparisons = []
50
51     for j in range(low,high):
52         if(arr[j] < pivot):
53             comparisons.append(f"{arr[j]}<{pivot} swap")
54             i+=1
55             arr[i],arr[j] = arr[j],arr[i]
56         else:
57             comparisons.append(f"{arr[j]}>{pivot}")
58
59     arr[i+1],arr[high] = arr[high],arr[i+1]
60     print(f"Partition: {arr[low:high+1]} (pivot: {pivot}) -> {arr[low:i+1]} {arr[i+2:high+1]}")
61     pivot at index {i+1-low} (comparisons: {', '.join(comparisons)}"))
62     return i + 1
63
64 def quick_sort(arr,low = 0,high = None):
65     if(high == None):
66         high = len(arr)-1
67
68     if(low < high):
69         pivot_idx = partition(arr,low,high)
70         quick_sort(arr,low,pivot_idx-1)
71         quick_sort(arr,pivot_idx+1,high)
72
73     return arr
74
75 arr = [38, 27, 43, 3, 9, 82, 10]
76
77 print("Quick sort steps:")
78 res = quick_sort(arr)
79 print(f"Sorted array: {res}")
80 print("Time complexity: O(n log n) average, O(n^2) worst; Space: O(log n)")
81
82
83 def insertion_sort(arr):
84     s = []
85     for i in range(len(arr)):
86         s.append(arr[i])
87         print(f"Insert {arr[i]}:", end=" ")
88         s = insert(s, i)
89
90     return s
91
92 def insert(arr, i):
93     comparisons = []
94     while i > 0:
95         if arr[i - 1] > arr[i]:
96             comparisons.append(f"{arr[i]}<{arr[i-1]}, shift {arr[i-1]}")
97             arr[i], arr[i - 1] = arr[i - 1], arr[i]
98         else:
99             comparisons.append(f"{arr[i]}>{arr[i-1]}, no shift")
```

```
98         break
99     i -= 1
100    if comparisons == []:
101        print(arr)
102    else:
103        print("compare", ', '.join(comparisons), "->", arr)
104    return arr
105
106 arr = [38, 27, 43, 3, 9, 82, 10]
107 res = insertion_sort(arr)
108 print("Insertion Sort Steps:")
109 print(f"Sorted array: {res}")
110 print("Time complexity: O(n^2), Space: O(1)")
111
112
113
114 def bfs(graph, start):
115     visited_vertices = set()
116     level = 0
117     queue = [start]
118     distances = {start: 0}
119     print("BFS Traversal from A:")
120     while queue != []:
121         vertices_in_level = len(queue)
122         level_nodes = []
123         print(f"L{level}:", end=" ")
124         for i in range(vertices_in_level):
125             vertex = queue.pop(0)
126             visited_vertices.add(vertex)
127             level_nodes.append(vertex)
128
129             for child in graph[vertex]:
130                 if child not in queue and child not in visited_vertices:
131                     queue.append(child)
132                     distances[child] = distances[vertex]+1
133             print(f"L{level}:", ', '.join(level_nodes))
134         level+=1
135     print("Distances:", distances)
136     return distances
137
138 def shortest_path(graph, start, end):
139     queue = [start]
140     visited = {start}
141     parent = {start: None}
142     while queue != []:
143         vertex = queue.pop(0)
144         if vertex == end:
145             break
146         else:
147             for child in graph[vertex]:
```

```
148         if child not in visited:
149             visited.add(child)
150             parent[child] = vertex
151             queue.append(child)
152
153     path = []
154     curr = end
155     while curr != None:
156         path.append(curr)
157         curr = parent.get(curr)
158     path.reverse()
159
160     print(f"Shortest path from {start} to {end}:", ' -> '.join(path))
161     return path
162
163 graph = {
164     'A': ['B', 'C'],
165     'B': ['A', 'D', 'E'],
166     'C': ['A', 'F'],
167     'D': ['B'],
168     'E': ['B', 'F'],
169     'F': ['C', 'E']
170 }
171 start = 'A'
172 res = bfs(graph,start)
173 path = shortest_path(graph,'A','F')
174 print("Time complexity: O(V + E), Space: O(V)")
```