set10.py

```python
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left = arr[:mid]
        right = arr[mid:]
        print(f"Divide: {arr} -> {left} and {right}")
        left = merge_sort(left)
        right = merge_sort(right)
        return merge(left, right)
    return arr

def merge(left, right):
    s = []
    i = j = 0
    comparisons = []
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            comparisons.append(f"{left[i]}<{right[j]}")
            s.append(left[i])
            i += 1
        else:
            comparisons.append(f"{left[i]}>{right[j]}")
            s.append(right[j])
            j += 1
    while i < len(left):
        s.append(left[i])
        if j > 0:
            comparisons.append(f"{left[i]}>{right[j-1]}")
        i += 1
    while j < len(right):
        s.append(right[j])
        if i > 0:
            comparisons.append(f"{right[j]}>{left[i-1]}")
        j += 1
    print(f"Merge: {left} and {right} -> {s} (comparisons: {', '.join(comparisons)})")
    return s

arr = [38, 27, 43, 3, 9, 82, 10]
print("Merge Sort Steps:")
res = merge_sort(arr)
print(f"Sorted array: {res}")
print("Time complexity: O(n log n), Space: O(n)")



def partition(arr,low,high):
    pivot = arr[high]
    i = low - 1
```

```
    comparisons = []

    for j in range(low,high):
        if(arr[j] < pivot):
            comparisons.append(f"{arr[j]}<{pivot} swap")
            i+=1
            arr[i],arr[j] = arr[j],arr[i]
        else:
            comparisons.append(f"{arr[j]}>{pivot}")

    arr[i+1],arr[high] = arr[high],arr[i+1]
    print(f"Partition: {arr[low:high+1]} (pivot: {pivot}) -> {arr[low:i+1]} {arr[i+2:high+1]} pivot at index {i+1-low} (comparisons: {', '.join(comparisons)})")
    return i + 1

def quick_sort(arr,low = 0,high = None):
    if(high == None):
        high = len(arr)-1

    if(low < high):
        pivot_idx = partition(arr,low,high)
        quick_sort(arr,low,pivot_idx-1)
        quick_sort(arr,pivot_idx+1,high)
    return arr

arr = [38, 27, 43, 3, 9, 82, 10]

print("Quick sort steps:")
res = quick_sort(arr)
print(f"Sorted array: {res}")
print("Time complexity: O(n log n) average, O(n^2) worst; Space: O(log n)")



def insertion_sort(arr):
    s = []
    for i in range(len(arr)):
        s.append(arr[i])
        print(f"Insert {arr[i]}:", end=" ")
        s = insert(s, i)
    return s

def insert(arr, i):
    comparisons = []
    while i > 0:
        if arr[i - 1] > arr[i]:
            comparisons.append(f"{arr[i]}<{arr[i-1]}, shift {arr[i-1]}")
            arr[i], arr[i - 1] = arr[i - 1], arr[i]
        else:
            comparisons.append(f"{arr[i]}>{arr[i-1]}, no shift")
```

```
            break
        i -= 1
    if comparisons == []:
        print(arr)
    else:
        print("compare", ', '.join(comparisons), "->", arr)
    return arr

arr = [38, 27, 43, 3, 9, 82, 10]
res = insertion_sort(arr)
print("Insertion Sort Steps:")
print(f"Sorted array: {res}")
print("Time complexity: O(n^2), Space: O(1)")



def bfs(graph, start):
    visited_vertices = set()
    level = 0
    queue = [start]
    distances = {start: 0}
    print("BFS Traversal from A:")
    while queue != []:
        vertices_in_level = len(queue)
        level_nodes = []
        print(f"L{level}:",end=" ")
        for i in range(vertices_in_level):
            vertex = queue.pop(0)
            visited_vertices.add(vertex)
            level_nodes.append(vertex)

            for child in graph[vertex]:
                if child not in queue and child not in visited_vertices:
                    queue.append(child)
                    distances[child] = distances[vertex]+1
        print(f"L{level}:",', '.join(level_nodes))
        level+=1
    print("Distances:",distances)
    return distances

def shortest_path(graph, start, end):
    queue = [start]
    visited = {start}
    parent = {start: None}
    while queue != []:
        vertex = queue.pop(0)
        if vertex == end:
            break
        else:
            for child in graph[vertex]:
```

```
                if child not in visited:
                    visited.add(child)
                    parent[child] = vertex
                    queue.append(child)

    path = []
    curr = end
    while curr != None:
        path.append(curr)
        curr = parent.get(curr)
    path.reverse()

    print(f"Shortest path from {start} to {end}:",' -> '.join(path))
    return path

graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B', 'F'],
    'F': ['C', 'E']
}
start = 'A'
res = bfs(graph,start)
path = shortest_path(graph,'A','F')
print("Time complexity: O(V + E), Space: O(V)")
```