CSE222 / BİL505
Data Structures and Algorithms
Homework #6 – Report

ÇAĞRI YILDIZ

1) Selection Sort

Time Analysis	**Time Analysis:** - **Best Case:** \`O(n^2)\` because even if the array is already sorted, comparisons are still needed. - **Worst Case:** \`O(n^2)\` due to nested loops comparing each element to find the minimum. - **Average Case:** \`O(n^2)\` for similar reasons. **Example Run:** Input: \`[64, 25, 12, 22, 11]\` Output: \`[11, 12, 22, 25, 64]\` Comparison Counter: 10, Swap Counter: 4
Space Analysis	- **Space Complexity:** \`O(1)\` because only a few variables are needed for swapping.

2) Bubble Sort

Time Analysis	**Time Analysis:** - **Best Case:** \`O(n)\` when the array is already sorted (no swaps occur). - **Worst Case:** \`O(n^2)\` because all adjacent pairs must be compared. - **Average Case:** \`O(n^2)\` as nested loops compare adjacent pairs. **Example Run:** Input: \`[64, 25, 12, 22, 11]\` Output: \`[11, 12, 22, 25, 64]\` Comparison Counter: 10, Swap Counter: 9
Space Analysis	- **Space Complexity:** \`O(1)\` as it requires a constant amount of memory for swapping.

3) Quick Sort

Time Analysis	**Time Analysis:** - **Best Case:** \`O(n log n)\` when the pivot divides the array into roughly equal halves. - **Worst Case:** \`O(n^2)\` when the pivot repeatedly divides the array into unbalanced partitions. - **Average Case:** \`O(n log n)\` due to partitioning. **Example Run:** Input: \`[64, 25, 12, 22, 11]\` Output: \`[11, 12, 22, 25, 64]\` Comparison Counter: 9, Swap Counter: 7
Space Analysis	**Space Analysis:** - **Space Complexity:** \`O(log n)\` due to the recursive call stack.

4) Merge Sort

Time Analysis	**Time Analysis:** - **Best Case:** \`O(n log n)\` because the array is always split into two halves. - **Worst Case:** \`O(n log n)\` due to consistent partitioning. - **Average Case:** \`O(n log n)\` as partitioning remains consistent. **Example Run:** Input: \`[64, 25, 12, 22, 11]\` Output: \`[11, 12, 22, 25, 64]\` Comparison Counter: 6, Swap Counter: 0
Space Analysis	- **Space Complexity:** \`O(n)\` due to the temporary arrays used during merging.

General Comparison of the Algorithms

1. **Time Complexity Comparison:**
   - **Selection Sort:** \`O(n^2)\`
   - **Bubble Sort:** \`O(n^2)\`
   - **Quick Sort:** \`O(n log n)\`
   - **Merge Sort:** \`O(n log n)\`
2. **Space Complexity Comparison:**
   - **Selection Sort:** \`O(1)\`
   - **Bubble Sort:** \`O(1)\`
   - **Quick Sort:** \`O(log n)\`
   - **Merge Sort:** \`O(n)\`