7. Comparison of Sorting Algorithms

7.1 Performance Summary

```
// Test all sorting algorithms
#include <time.h>

void testSortingAlgorithms() {
    int sizes[] = {100, 1000, 10000};

    for (int s = 0; s < 3; s++) {
        int n = sizes[s];
        printf("\n=== Array Size: %d ===\n", n);

        // Test each algorithm
        int *arr = generateRandomArray(n);

        clock_t start = clock();
        bubbleSort(arr, n);
        clock_t end = clock();
        printf("Bubble Sort: %.4f seconds\n",
               (double)(end - start) / CLOCKS_PER_SEC);

        // Repeat for other algorithms...
        free(arr);
    }
}
```

7.2 When to Use Which Algorithm

Decision Tree:

```
Is n < 50?
├─ Yes → Use Insertion Sort (simple, fast for small n)
└─ No  → Is stability required?
         ├─ Yes → Use Merge Sort
         └─ No  → Is memory limited?
                  ├─ Yes → Use Quick Sort (in-place)
                  └─ No  → Use Merge Sort (guaranteed O(n log n))
```

By Data Characteristics:

```
Nearly sorted →Insertion Sort (O(n))
Reverse sorted → Any O(n log n) algorithm
Random data → Quick Sort (fastest in practice)
Many duplicates → Quick Sort 3-way
Small data → Insertion Sort
Large data → Merge Sort or Quick Sort
Linked list → Merge Sort (O(1) space)
Stability needed → Merge Sort or Insertion Sort
Memory limited → Quick Sort or Heap Sort
```

7.3 Hybrid Sorting Approaches

IntroSort (Introspective Sort):

```
void introSort(int arr[], int low, int high, int depthLimit) {
    int n = high - low + 1;

    // Use Insertion Sort for small arrays
    if (n < 16) {
        insertionSort(arr + low, n);
        return;
    }

    // Switch to Heap Sort if recursion too deep
    if (depthLimit == 0) {
        heapSort(arr + low, n);
        return;
    }

    // Use Quick Sort
```

```
    int pi = partition(arr, low, high);
    introSort(arr, low, pi - 1, depthLimit - 1);
    introSort(arr, pi + 1, high, depthLimit - 1);
}

// Wrapper function
void sort(int arr[], int n) {
    int depthLimit = 2 * log2(n);
    introSort(arr, 0, n - 1, depthLimit);
}
```

TimSort (Python's default sort):

- Combination of Merge Sort and Insertion Sort
- Identifies natural runs in data
- Uses Insertion Sort for small runs
- Merges runs using Merge Sort

Revision #1
Created 2025-11-03 02:07:50 UTC by DS
Updated 2025-11-03 02:10:11 UTC by DS