Week-2 (13-08-2025)

Task -1 [5m]

Implement the following sorting algorithms in C:

1. Selection Sort
2. Heap Sort
3. Bubble Sort

Instructions:

1. For each algorithm, measure the execution time for different input sizes (e.g., 1000, 5000, 10000, 20000 elements).
2. Use PBPlot to plot a graph of *Execution Time (y-axis)* vs *Input Size (x-axis)* for all three algorithms on the same chart.
3. Clearly label the graph with algorithm names, axes labels, and a title.
4. Compare and discuss the growth trends in runtime for each algorithm.

Expected Outcome:
Students should observe that Bubble Sort and Selection Sort ($O(n^2)$) grow much faster than Heap Sort ($O(n \log n)$) as input size increases.

Task -2 [5m]

1. 1. Implement Quick Sort using:
    - Fixed Pivot Strategy: Choose the last element as pivot.
2. Implement Randomized Quick Sort using:
    - Random Pivot Strategy: Randomly select a pivot before partitioning.
3. Generate the following input cases:
    - Case 1: Already sorted array.
    - Case 2: Reverse sorted array.
    - Case 3: Randomized array.
4. For each case, measure execution time for:
    - n=1000,2000,5000,10000,20000 (or more, depending on system capacity).
5. Plot the results using PBPlot:
    - X-axis: Array size n

    - Y-axis: Execution time in milliseconds
    - Two curves per plot: Quick Sort vs Randomized Quick Sort
6. Analyze and explain:
    - Why Quick Sort may degrade to O(n^2) in certain inputs.
    - Why Randomized Quick Sort gives more consistent results.
    - Differences in trends for different input cases.

Expected Output:
Three runtime comparison plots — one each for sorted, reverse sorted, and random input arrays — showing Randomized Quick Sort maintaining O(nlogn) trend while Quick Sort degrades for sorted/reverse sorted arrays.

Bonus Question

Implement a Hash Table with Chaining for collision handling in C, and a Binary Search algorithm.

Instructions:

1. Generate n random integers for n = 1000, 5000, 10000.
2. Implement Hash Table Search using chaining (linked lists for buckets).
3. Implement Binary Search on a sorted array.
4. Measure average search time for both algorithms for each n over at least 1000 search operations.
5. Use PBPlot to plot *Search Time (y-axis)* vs *Number of Elements (x-axis)*, showing both algorithms on the same graph.
6. Add labels, title, and legend for clarity.
7. Write a short observation explaining why Hash Table search is O(1) average case while Binary Search is O(log n), and discuss whether your experimental results match theory.

Expected Outcome:

- Hash Table search time should remain almost constant as n increases.
- Binary Search time should grow slowly (logarithmically) with n.