19CSE302: Design and Analysis of Algorithms
Sorting Algorithms
30.8.2023

In this lab exercise you will be implementing the sorting and graph algorithms and evaluating their performance. You can work in a team of two.

Question 1.

In the first part implement the basic sorting algorithms 1. In-place Quick Sort 2. Merge Sort 3. In-place Heap Sort 4. Insertion Sort 5. Bucket Sort 6. Radix Sort. The pseudocode for a few algorithms are attached and the heap ADT is also given. Your implementation should follow the pseudo-code and use the ADT for heapsort. Generate multiple random input test cases of various sizes (e.g 100 (min), 500, 1000) and evaluate the different algorithms based on

a. Number of comparisons (if applicable)

b. Number of swaps (if applicable)

c. Number of basic operations (other than above)

d. Running time in milliseconds

e. Memory used

Based on the above provide your observations and analysis.

To submit

1. Code files (with clear comments) . Doc files etc with pasted code will be not be accepted
2. Input data files - test cases that is used for the evaluation
3. A report with the analysis and screenshots.

Question 2.

There is scope to improve quick sort, merge sort. In quick sort, the goal is to avoid the worst case scenario. In merge sort memory can be managed more effectively. What are the current improvements to these algorithms? For instance timsort is an improvement to mergesort. In quick sort the quick sort can be hybridised with insertion sort ie run quick sort upto a point and run insertion sort on the smaller sub-problems. Implement two modifications to quick sort and merge sort, and analyse their performance with respect to the basic implementations. Provide a report with your observations and analysis.

To submit

1. The description of the modified algorithms (with source where you referred), and its theoretical analysis of running time.
2. Code files (with clear comments) . Doc files etc with pasted code will be not be accepted
3. Input data files - test cases that is used for the evaluation
4. A report with the analysis and screenshots.

Question 3:

Snakes and Ladders is a classic board game, originating in India no later than the 16th century. The board consists of an n × n grid of squares, numbered consecutively from 1 to n 2 , starting in the bottom left corner and proceeding row by row from bottom to top, with rows alternating to the left and right. Certain pairs of squares in this grid, always in different rows, are connected by either “snakes” (leading down) or “ladders” (leading up). Each square can be an endpoint of at most one snake or ladder. The goal is to reach the nxn th grid position or the paramapatam fastest.

You start with a token in cell 1, in the bottom left corner. In each move, you advance your token up to k positions, for some fixed constant k. If the token ends the move at the top end of a snake, it slides down to the bottom of that snake. Similarly, if the token ends the move at the bottom end of a ladder, it climbs up to the top of that ladder

The following are the two tasks

1. You are given a board. It is represented by a. the size nxn (n is at least 8), b. number of snakes, and number of ladders, and c. the start and destination grid positions of the snakes and ladders in the board. You have to verify if the board satisfies the following conditions:
    1. There is at least one way to reach the goal
    2. No two snakes/ladders start or end at the same grid position. (can’t process the input directly)
    3. There is no cycles formed by the combination of snakes and ladders.
    4. There is no ladder from the start position to the destination directly.
2. Find the shortest sequence of dice rolls to reach the goal from the start position on the input board above, considering that the maximum distance that can be moved is dictated by the dice aka 6 or the snake/ladder.

Implement this game along with the two tasks. Explain how you model the game and represent the game using the best representation. Justify the modeling. Explain the algorithm for the two tasks (choose the efficient algorithm for the same and those that satisfy the conditions given), the complexity of the algorithm and prove the correctness. Analyse the correctness of the algorithm using different test cases.

To submit

1. The description of the algorithms to solve both parts and theoretical analysis of running time.

3. Code files (with clear comments) . Doc files etc with pasted code will be not be accepted
4. Input data files - test cases that is used for the evaluation
5. A report with the analysis of correctness with various test cases and screenshots.

Submit a zip file with a folder each for each problem.

Deadline for submission: 5.9.2023, 11.55 pm on AUMS.