Data Structures Laboratory

B.Tech Fourth Semester, Jan. 2011 - Apr. 2011)

Experiment Set I

Batch I Jan 5, Wed & Batch II Jan 6, Thu:

Priority queue using heap: Implement "extract_min" and "decrease_key" in O(log n).
Heap sort - The implementation must run in O(n logn) worst case.

Batch III Jan 12, Wed & Batch IV Jan 13, Thu:

Heap sort - The implementation must run in O(n logn) worst case.

Quick sort Implementation: Your recursive implementation must run in O(nlogn) average case. Remove the tail recursion and implement quick sort to use only O(log n) recursion depth in the worst case.

Batch I Jan 19, Wed & Batch II Jan 20, Wed:

Quick sort Implementation: Your recursive implementation must run in O(nlogn) average case. Remove the tail recursion and implement quick sort to use only O(log n) recursion depth in the worst case.

Binary Search Tree: Using linked list. Implement "Insert", "Delete" and "Search" Procedures.

Batch III Feb 02, Wed & Batch IV Feb 03, Wed

Binary Search Tree: Using linked list. Implement "Insert", "Delete" and "Search" Procedures

Expression Tree: Read and infix expression, convert to postfix and construct expression tree. Print the in-order and post-order traversals.

Batch I Feb 09, Wed & Batch II Feb 10, Wed:

Expression Tree: Read and infix expression, convert to postfix and construct expression tree. Print the in-order and post-order traversals.

Graph Search : Graph must be represented as adjacency list. Implement DFS and find the **bi-connected components** of the (un-directed) graph in O(n+e) time.

Batch III Feb 16, Wed & Batch IV Feb 17, Wed:

Graph Search : Graph must be represented as adjacency list. Implement DFS and BFS. and find the **strongly connected components** of the (directed) graph using Kosaraju's algorithm. All algorithms must run in O(n+e) time.

Dijkstra's Algorithm: Use binary heap to yield an O((n+e)log n) implementation with adjacency list representation of the Graph.

Batch I Feb 23, Wed & Batch II Feb 24, Thu:

Prim's Algorithm: Use binary heap to yield an O((n+e)log n) implementation with adjacency list representation of the Graph.

Kruskal's algorithm: Use "Union_Find" data structure to yield an implementation in O((n+e)log n) complexity for graph represented by adjacency list.

Batch III and Batch IV, March 03 Thursday:

Kruskal's algorithm: Use "Union_Find" data structure to yield an implementation in O((n+e)log n) complexity for graph represented by adjacency list.

March 09, Wed: Buffer day for all batches to submit pending experiments.

Batch I March 23, Wed, Batch II March 24, Thu:

Merge Sort on Files: For n elements in a file, you can use 2n disk space and only constant amount of main memory space. Ensure that your program runs in O(log n) complexity. Use a random number generator to generate a file with 10000, 100000 and 1000000 numbers and sort using your merge sort implementation.

Batch III March 30, Wed, Batch IV March 31, Thu:

Merge Sort on Files: For n elements in a file, you can use 2n disk space and only constant amount of main memory space. Ensure that your program runs in O(log n) complexity. Use a random number generator to generate a file with 10000, 100000 and 1000000 numbers and sort using your merge sort implementation.

Experiment Set II

Batch I Jan 5, Wed & Batch II Jan 6, Thu:

Stack and Queue using array: Implementation of stack and (circular) queue using array.

Batch III Jan 12, Wed & Batch IV Jan 13, Thu:

Stack and Queue using array: Implement stack and circular queue using array.
Bubble Sort and Binary Search: Generate 1000 random numbers, store them in an array and sort using bubble sort. Implement linear and binary search algorithms for search in the array.

Batch I Jan 19, Wed & Batch II Jan 20, Wed:

Insertion sort and Binary Search: Generate 1000 random numbers, store them in an array and sort using insertion sort. Implement linear and binary search algorithms for search in the array.

Stack and Queue using Linked list with constant time for insert and delete.

Batch III Feb 02, Wed & Batch IV Feb 03, Wed

Stack and Queue using Linked list with constant time for insert and delete.

Doubly linked list implementation with "Insert", "Delete" and "Search".

Batch I Feb 09, Wed & Batch II Feb 10, Wed:

Doubly linked list implementation with "Insert", "Delete" and "Search".

Infix to Postfix: Conversion must be done in linear time.

Batch III Feb 16, Wed & Batch IV Feb 17, Wed:

Infix to Prefix: Conversion must be done in linear time.

Binary Search Tree with "Insert", "Delete" and "Search".

Batch I Feb 23, Wed & Batch II Feb 24, Thu:

Binary Search Tree with "Insert", "Delete" and "Search".

DFS: Non-recursive implementation of Depth First Search to find the connected components of a given graph. Graph may be represented by adjacency matrix. The algorithm must run in quadratic time in the number of vertices.

Batch III and Batch IV, March 03 Thursday:

BFS: Implementation of Breadth First Search algorithm in quadratic time. The graph may be represented by its adjacency matrix.

March 09, Wed: Buffer day for all batches to submit pending experiments.

Batch I March 23, Wed, Batch II March 24, Thu:

Dijkstra's algorithm: Assume adjacency matrix representation for the graph. Running time shall be at most quadratic in the number of vertices.

Batch III March 30, Wed, Batch IV March 31, Thu:

*Prim's algorithm:*Assume adjacency matrix representation for the graph. Running time shall be at most quadratic in the number of vertices.

March 06, Wed & March 07 Thu: Buffer day for all batches to submit pending experiments.

Page Maintained by:

K. Murali Krishnan, Faculty CSED, NITC. Email: kmurali@nitc.ac.in