Data Structures and Algorithms

1. Implement a program to handle linked list. Required functions are given below. (c89)
    - `new_head(ptr_head, int)`
    - `find_node(head, int)`
    - `delete_node(ptr_head, int)`
    - `print_all_nodes(head)`
    - `total_nodes(head)`

2. Show the following operations on the STL containers listed below. (c++98)

    Containers:
    - `vector<int>`, `vector<string>`
    - `list<int>`, `list<string>`
    - `map<int, string>`, `map<string, int>`
    - `set<int>`
    - `bitset`
    - `vector<vector<int> >`
    - `map<string, map<string, string> >`

    Operations:
    - Insert
    - Delete
    - Find
    - Print all the elements

3. Run the following STL algorithms on the STL containers listed above as applicable. (c++98)
    - `sort`
    - `binary_search`
    - `find`
    - `max_element`
    - `random_shuffle`
    - `next_permutation`

4. a. Implement binary search on integers. (c89)
b. Convert binary search as performed above to a generic data type. Use operator overloading to demonstrate binary search on a `struct` type. (c++98)

5. a. Implement quicksort on integers. (c89)
b. Convert the above code to a work on a generic data type. (c++98)

6. a. Implement a stack using a linked list. The functions to work on it are given below: (c89)
    - `size()`
    - `is_empty()`
    - `push()`
    - `top()`
    - `pop()`

    b. Use the above approach to make a generic linked list that works like STL stack for the same operations. (c++98)

7. a. Implement a queue using a linked list. The functions to work on it are given below: (c89)
    - `size()`
    - `is_empty()`
    - `enqueue()`
    - `top()`
    - `dequeue()`

    b. Use the above approach to make a generic linked list that works similar to STL queue for the same operations.

8. a. Implement Heap-Sort on *n* integers. (c89)
   b. Convert the above code to a work on a generic data type. (c++98)

9. a. Implement Binary Search Tree (BST). Construct first a BST with *n* numbers. Subsequently, perform **search** (print "found" or "not_found") for (-1 *i*), **insert** for (-2 *i*), delete for (-3 *i*), print minimum for -4, print maximum for -5 and print sorted numbers for -6 and stop for 0. (c89)
   b. Convert the above code to work on a generic data type. (c++98)
      Eg:
      Input:
```
4
5 4 2 6
-1 4
-1 3
-4
-5
-6
0
```
      Output:
```
found
not_found
2
6
2
4
5
6
```

10. Implement trie tree. Insert for (1, *word*), search for (2, *word*), delete for (3, *word*) and stop for 0. (c89)
      Eg:
      Input:
```
1 Gone
1 Wind
2 Wind
2 Width
3 Wind
2 Wind
0
```
      Output:
```
found
not_found
not_found
```

11. a. Implement disjoint set operations. The first *n* lines give elements of *n* sets. Two integers should merge two disjoint sets (stored sequentially, smaller set is merged at the end of the larger) to which they belong and print the representative of disjoint sent. Single integer should result in search for the number and print the representative (first number) if found and -1 otherwise. 0 should result in termination of the program. (c++98, no STL)
    b. Convert the above code to work on a generic data type. (c++98)
      Eg:
      Input:
```
4
1 3 4 6
2 7 8 9
10 11
12 15 32
11 32
4
9
10
20
0
```

Output:
```
10
1
2
10
-1
```

12. Implement in Floyd-Warshall algorithm for a given graph in the form of weighted adjacency. Print row-wise the shortest distance between all pairs. Also print path of the shortest distance. (c++98)

    Eg:

    Input:
    ```
    0 1 5
    1 0 2
    5 2 0
    1 3
    ```

    Output:
    ```
    0 1 3
    1 0 2
    3 2 0
    1 -> 2 -> 3
    ```

13. a. Implement Bredth First Search and print the path length (no. of edges) between nodes *i* and *j* till *i* is zero. The graph is undirected and unweighted and the adjacency matrix is given as input. (c++98)

    Eg:

    Input:
    ```
    0 1 0
    1 0 1
    0 1 0
    1 3
    0
    ```

    Output:
    ```
    2
    ```

    b. Implement Depth First Search and print if nodes *i* and *j* are connected ("connected" or "not_connected") until 0 is encountered. The graph is directed and unweighted and the adjacency matrix is given as input. (c++98)

    Eg:

    Input:
    ```
    0 1 0
    1 0 0
    0 1 0
    1 3
    3 1
    0
    ```

    Output:
    ```
    not_connected
    connected
    ```

14. Solve *n*-queen problem printing the number of possible configurations on the chess-board. (c89)

    Eg:

    Input:
    ```
    8
    ```

    Output:
    ```
    92
    ```

15. Using hashing by chaining made by an array of 100 linked lists and a hash function that adds the ASCII values and rounds it off to the range 0-99, implement **insert** (1, *word*), **find** (2, *word*) and **delete** (3, *word*). The program should terminate with input 0. (c89)

    Eg:

    Input:
    ```
    1 Gone
    1 Wind
    2 Wind
    2 Width
    3 Wind
    2 Wind
    0
    ```

    Output:
    ```
    found
    not_found
    not_found
    ```