Data Structures and Algorithms

Lab assignments

1. Implement a stack using an array. The functions to work on it are given below:
   - `size()`
   - `is_empty()`
   - `push()`
   - `top()`
   - `pop()`

   Eg:
   Input:
   ```
   push 5
   push 15
   top
   pop
   push 25
   top
   pop
   top
   size
   is_empty
   pop
   is_empty
   exit
   ```
   Output:
   ```
   15
   removed 15
   25
   removed 25
   5
   1
   no
   removed 5
   yes
   ```

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

   Eg:
   Input:
   ```
   new_head 12
   new_head 10
   new_tail 15
   total_nodes
   print_all_nodes
   find 10
   delete_node 10
   find 10
   print_all_nodes
   exit
   ```
   Output:
   ```
   3
   10 12 15
   10 found
   10 deleted
   10 not found
   12 15
   ```

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

   Eg:
   Input:
   ```
   size
   is_empty
   enqueue 10
   front
   enqueue 20
   front
   size
   is_empty
   dequeue
   dequeue
   exit
   ```

   Output:
   ```
   0
   yes
   10
   10
   2
   no
   removed 10
   removed 20
   ```

4. Implement binary search on integers and strings. Assume that the input is not sorted.
   a. Integers

      Eg:
      Input:
      ```
      5
      5 1 2 3 4
      search 2
      search 10
      search 3
      exit
      ```

      Output:
      ```
      found 2
      not found 10
      ```

```
found 3
```

b. Strings

Eg:
Input:
```
3
hello world c
search hello
search India
search WoRLd
exit
```

Output:
```
found "hello"
not found "India"
found "world"
```

5. Implement quicksort on integers.

Eg:
Input:
```
5
4 6 -1 32 4
```

Output:
```
-1 4 4 6 32
```

6. Implement Heap-Sort on *n* integers.

Eg:
Input:
```
5
4 6 -1 32 4
```

Output:
```
-1 4 4 6 32
```

7. 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.

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
```

8. Implement trie tree. Insert for (1, *word*), search for (2, *word*), delete for (3, *word*) and stop for 0.

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

Output:
```
found
not found
not found
```

9. 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.

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
```

10. 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.

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
```

11. 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.

    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.
    Eg:
    Input:

    ```
    0 1 0
    1 0 0
    0 1 0
    1 3
    3 1
    0
    ```

    Output:

    ```
    Not connected
    connected
    ```

12. 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.
    Eg:
    Input:

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

    Output:

    ```
    found
    not found
    not found
    ```