***TIME COMPLEXITIES***

**Singly linked list:**

Insertion of the first node 🡪 O(1)

Deletion of the head node 🡪 O(1)

Insertion of the node at the end🡪 O(1)

Deletion of the last node 🡪 O(n)

*[the list has to be traversed in order to do this]*

**Stacks and queues:**

to search a node 🡪 O(n)

to insert/delete a node 🡪 O(1)

**Binary search tree:**

to search /insert/delete 🡪 O(log(n))

**Graphs:**

adjacency list 🡪 O(V+E)

adjacency matrix 🡪 O(V^2) ( *where V=vertex; E=edge*)

**OUTPUTS**

output for given graph using DFS:

**ABDHGCEJFI**

output for given graph using BFS:

**ABCDFHIEGJ**

**GRAPHS**

**DFS FOR A GRAPH:**

The DFS algorithm works as follows:

1.Start by putting any one of the graph's vertices on top of a stack.

2.Take the top item of the stack and add it to the visited list.

3.Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of stack.

4.Keep repeating steps 2 and 3 until the stack is empty.

Dfs(G,s) //g=graph,s=source vertex,S=stack,w=neighbours

S.push(s) //inserting s in stack and mark s as visited

While(S!=empty) //pop and print a vertex to visit the next

V=S.top()

S.pop()

If w is not visited:

S.push(w) //mark w as visited

Dfs(,s) //mark s as visited

If w is not visited:

Dfs(G,s)

***BFS FOR A GRAPH:***

1. Start by putting any one of the graph's vertices at the back of a queue.
2. Take the front item of the queue and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the back of the queue.
4. Keep repeating steps 2 and 3 until the queue is empty.

*Create a queue*

*Mark v as visited and put v into Q*

*While Q is non empty*

*Remove the head u of Q*

*Mark and enqueue all neighbours of u and print*