Queue-> FIFO

**Applications:**

1. Disk Scheduling,BF’s
2. Asynchronous data transfer(IO buffer, pipes, file IO)
3. Semaphores, FCFS,Spooling,Buffer,Router/Switches,Mail queues
4. Circular queue->memory management, traffic system, cpu schedulling

Time complexity: enqueue , dequeue , front, rear-> O(1)

Pros: Easy implementation

Cons: Fixed data structure, once a space get dequeued we can’t fill it again unless using circular queue.

Priority Queue: Every element has some priority and the one with higher priority is dequeued first in case of same priority they are dequeued according to their order.

**insert(item, priority):** O(1)   
**getHighestPriority():** O(n)  
**deleteHighestPriority():** use linear search and then moving each item one position back more effiecent when we use linked list.

But if we use heap the time taken will be O(1),O(logn),O(logn)

**Application:** Cpu Scheduling, Dijikstra, prim’s , load balancing os

Priority\_queue<int> q;

**Circular Queue/Ring Buffer:**

**Breadth First Search:** gfg book mark code

Traversing through layers and to avoid loops and cycle maintain a Boolean array of visited nodes.

**Level order Traversal:** Almost same as BF”s only diff is that uses queue<Node\*> and recursive call.

**Construct Binary tree from LL:** Generally given level order traversal first element is root and next 2 are left and right child .

Push-> curr element parent enqueuer 2 more elements in queue.

void convert\_to\_Binary\_from LL(Node \*head, TreeNode \*&root) {

queue<TreeNode\*> q;

if(head==NULL)

{

root=null;

return;

}

root=new TN(head->data)

q.push(root)

head=head->next;

while(head)

{

TN \* parent=q.front();

q.pop();

TN\* leftchild=rightchild=null;

leftchild=new TN(head->data);

q.push(leftchild);

head=head->next;

if(head)

{

rightchild=new TN(head->next);

q.push(rightchild);

head=head->next;

}

parent->left=leftchild;

parent->right=rightchild;

}

}

**Check for complete Binary tree:** O(n)

if(root==NULL)

return true;

queue<Node\*> q;

q.push(root);

bool flag=false;

while(!q.empty())

{

Node \*temp=q.front();

q.pop();

if(temp->left)

{

if(flag==true)

return false;

q.push(temp->left);

}

else

{

flag=true;

}

if(temp->right)

{

if(flag==true)

return false;

q.push(temp->right);

}

else

flag=true;

}

return true;

**\*Number of siblings of n-ary tree:**

if (root == NULL)

        return 0;

    queue<Node\*> q;

    q.push(root);

    while (!q.empty()) {

        Node\* p = q.front();

        q.pop();

        for (int i = 0; i < p->child.size(); i++) {

            if (p->child[i]->key == x)

                return p->child.size() - 1;

            q.push(p->child[i]);}}}

**Zig-Zag Tree Traversal:** O(n) ->time and space notes.

* FIFO approach is used in data structures , disk scheduling, communication and networking.
* LIFO used in data structure, extracting latest info.

Priority queue using heap/linked list, implement k queue (method 2), sorting a queue without using extra space, reserve a path in BST.