when to use which implementation  
Use **Array Implementation**:

* + When the maximum stack size is known in advance.
  + If memory overhead must be minimized.
  + When simplicity in implementation is preferred.

Use **Linked List Implementation**:

* + When stack size is unknown or unbounded.
  + If frequent push/pop operations are expected and memory is not a constraint.
  + To avoid resizing overhead.

difference between implementation

| Feature |  |  |  |  |  |  |  | Array Implementation |  |  |  |  |  | Linked List Implementation |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Memory |  |  |  |  |  |  |  | Fixed SIZE (STATIC) |  |  |  |  |  | Dynamic (GROWS/SHRINKS) |
| Resizing |  |  |  |  |  |  |  | Requires RESIZING |  |  |  |  |  | No RESIZING NEEDED |
| Flexibility |  |  |  |  |  |  |  | Less FLEXIBLE |  |  |  |  |  | Highly FLEXIBLE |
| Extra memory |  |  |  |  |  |  |  | None |  |  |  |  |  | Extra memory for pointers |
| Complexity |  |  |  |  |  |  |  | Easier TO IMPLEMENT |  |  |  |  |  | Slightly MORE COMPLEX |
| Overflow |  |  |  |  |  |  |  | Can OCCUR |  |  |  |  |  | No OVERFLOW (UNLESS MEMORY EXHAUSTED) |
| Underflow |  |  |  |  |  |  |  | Can OCCUR |  |  |  |  |  | Can OCCUR |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| QUEUE |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

**offer()** is a **non-blocking** method used to add elements to a queue. If the queue is full, it will return **false** instead of throwing an error, which helps in scenarios where you want to handle such situations gracefully.  
likewise poll() also

| **Queue Operation** | **When to Use It** | **Example Scenario** |
| --- | --- | --- |

|  |  |  |
| --- | --- | --- |
| **isEmpty** | When you want to check if the queue is empty | **Before dequeuing**, **check for tasks to process** |

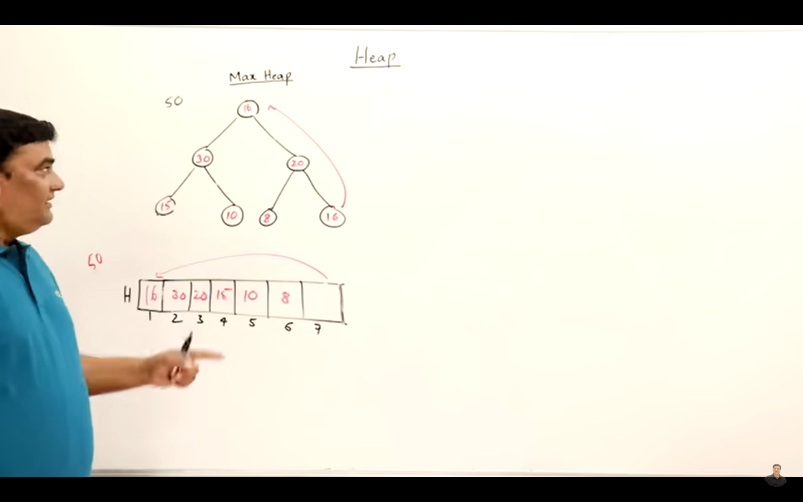
|  |  |  |
| --- | --- | --- |
| **isFull** | When the queue has a fixed size and you need to check capacity | **Buffer overflow prevention**, **fixed-size queues** |

|  |  |  |
| --- | --- | --- |
| **Size** | When you need to track the current number of elements | **Monitor pending jobs**, **check active users** |

**Enqueue**: Element ko queue ke end pe insert karo.  
**Dequeue**: Front se element remove karo.

* Queue has front and rear where both are initialized as -1 and when any value is inserted is goes to 0 then it but rear is variable while front is fixed when any value is removed rear get -1.

TYPES OF QUEUES:  
1.liner- has two operations enqueue and dequeue  
2.circular  
3.deque( double ended queue) types: 1-entry restricted , 2-exit restricted  
 deque can be represented using a doubly linked list and a circular array.  
4.priority queue : 1. Min priority queue 2.Maxpriorityqueue  
 jo element highest priority ka hota hai, woh sabse pehle dequeue hota hai, chahe woh queue mein sabse pehle aya ho ya nahi.

HEAP  
heap is a complete binary tree.(bcz order is necessary in heap otherwise queue is marked empty) (level order traversing is done here).  
max heap generates maxpriority queue: bcz root element is the greatest and so on.  
min heap generates minpriority queue  
log(n)  
  
how heap is performed  
  
isme 50 root tha pehle wahi dlt hoga or 16 upr a jayega sbse for arrangement hoga aise hi hota rhega.