Sorting

- arrangement of data elements either in ascending or descending order of their values
  1. Selection sort
  2. Bubble sort
  3. Insertion sort
  4. Merge sort
  5. Quick sort
  6. Heap sort

No. of comps $= 1+2+3+\cdots n-1$

$= \frac{n(n+1)}{2}$

$= \frac{1}{2}(n^2+n)$

Time $\propto \frac{1}{2}(n^2+n)$

Time $= n^2+n$

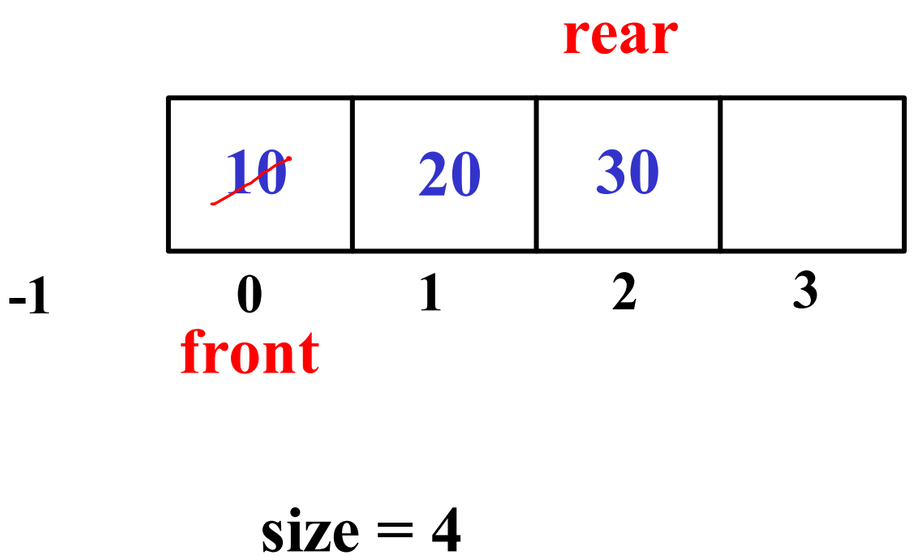
Time $= n^2$

$T(n) = O(n^2)$

	Selection Sort	Bubble Sort	Insertion Sort
Best case	$O(n^2)$	$O(n)$	$O(n)$
Average case	$O(n^2)$	$O(n^2)$	$O(n^2)$
Worst case	$O(n^2)$	$O(n^2)$	$O(n^2)$

Linear Queue

- linear data structure which is used to store similar type data
- data can be inserted from one end, that end is called rear
- data can be removed from another end, that end is called as front
- work on the principle of "First In First Out" (FIFO)



Conditions:

1. Full : rear == size - 1
2. Empty : front == rear

Operations:

1. Insert/Add/Enqueue/Push:

a. reposition the rear (inc)
b. add data at rear index

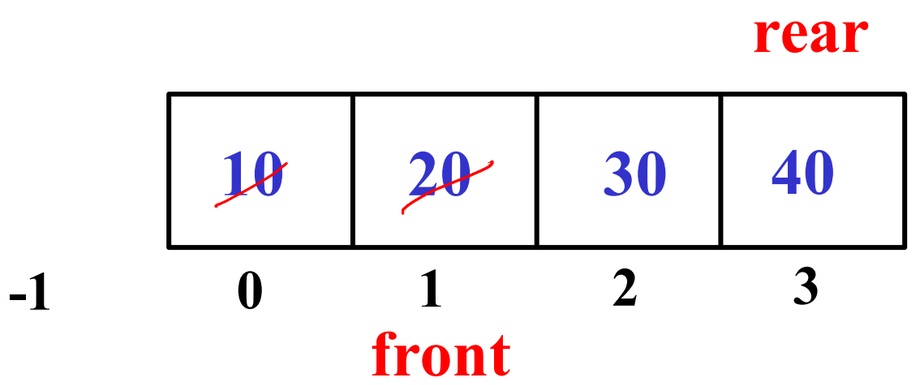
2. Remove/Delete/Dequeue/pop:

a. reposition the front (inc)

3. Peek (collect) :

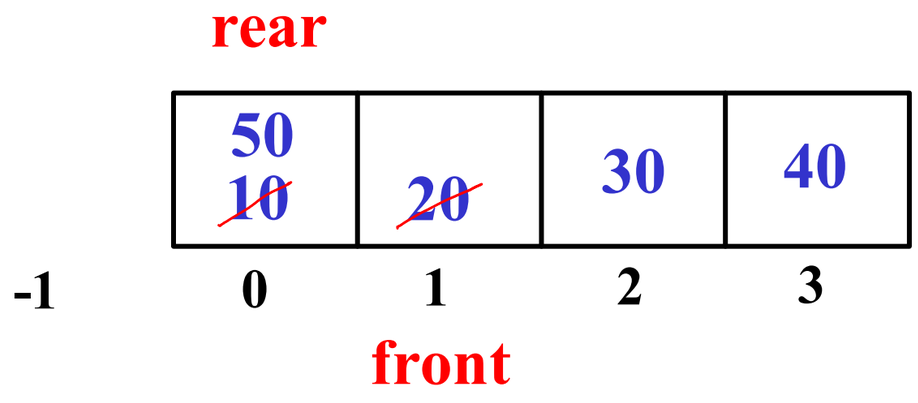
a. read/return data of front + 1 index

Linear Queue



- once rear is reached to last index of array, queue is full
- if initial few locations are free, we can not use them to add next data
- this will lead to poor memory utilization
- solution for above problem is circular queue

Circular Queue



size = 4

rear = (rear + 1) % size
front = (front + 1) % size

fornt = rear = -1

(-1 + 1) % 4 -> 0
(0 + 1) % 4 -> 1
(1 + 1) % 4 -> 2
(2 + 1) % 4 -> 3
(3 + 1) % 4 -> 0

Operations:

1. Insert/Add/Enqueue/Push:

a. reposition the rear (inc)
b. add data at rear index

2. Remove/Delete/Dequeue/pop:

a. reposition the front (inc)

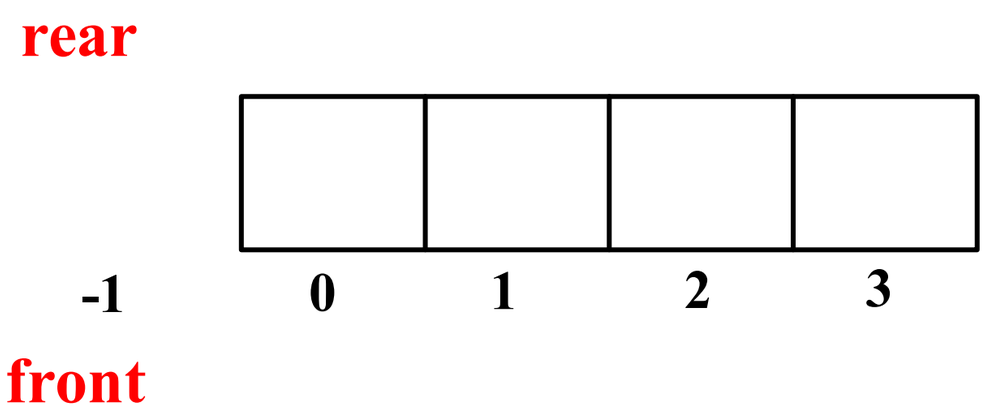
3. Peek (collect) :

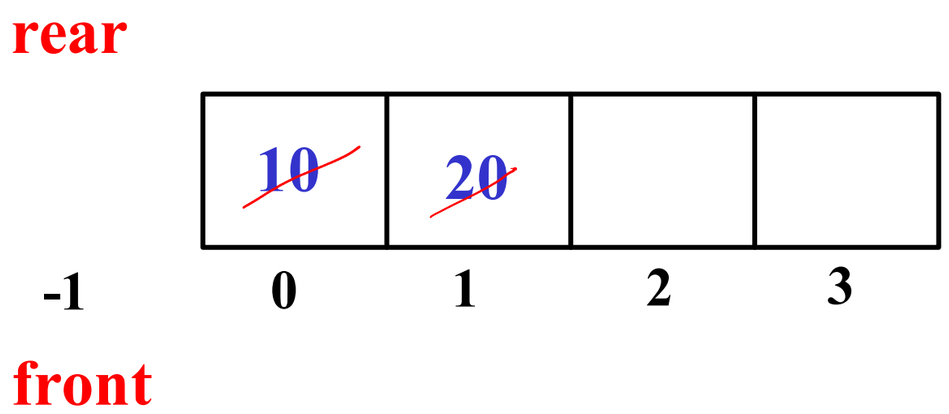
a. read/return data of front + 1 index

Conditions

1. Full : count == SIZE
2. Empty : count == 0

Circular Queue - Empty

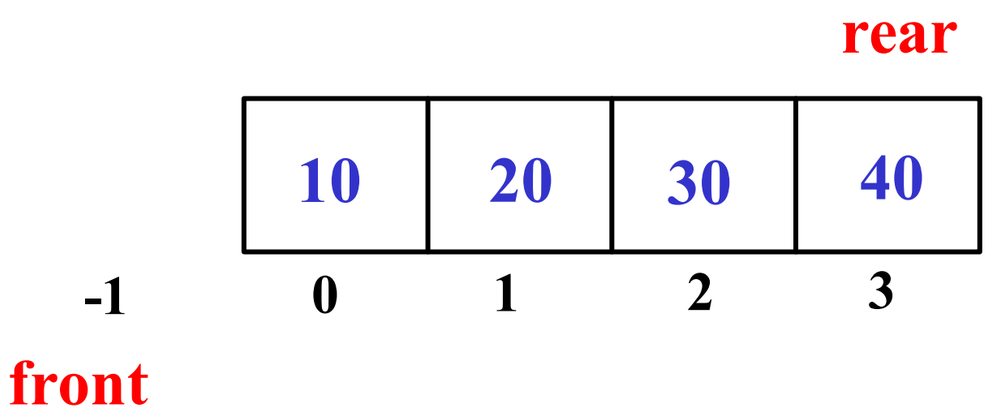


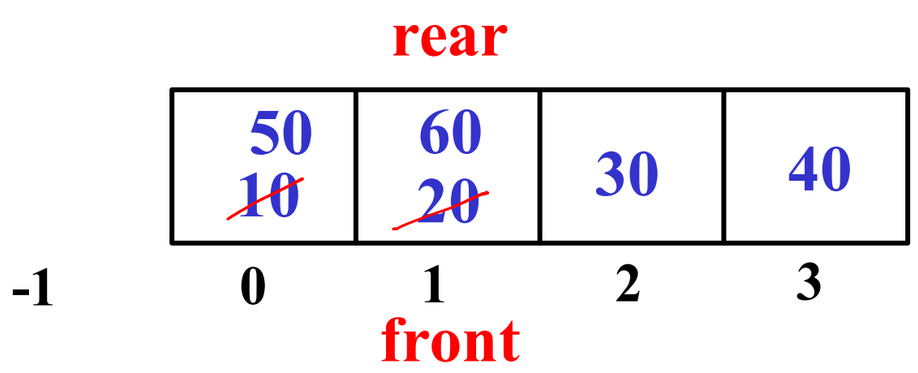


front == rear && rear == -1

```
pop(){
    fornt = (front + 1) % SIZE;
    if(front == rear)
        front = rear = -1;
}
```

Circular Queue - Full





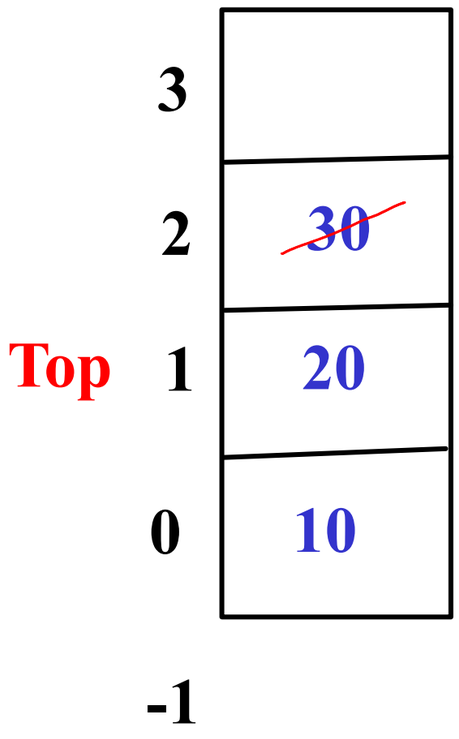
rear == SIZE -1 && front == -1

front == rear && rear != -1

(rear == SIZE -1 && front == -1) || (front == rear && rear != -1)

Stack

- linear data structure which is used to store similar type of data
- insert and remove of data is allowed from only one end (top)
- works on the principle of "Last In First Out" (LIFO)



Operations:

1. Add/Insert/push:

a. reposition top (inc)
b. add data at top index

2. Delete/Remove/pop:

a. reposition top (dec)

3. peek (Collect):

a. read/return data of top index

Condition:

1. Full : top == SIZE -1
2. Empty : top == -1