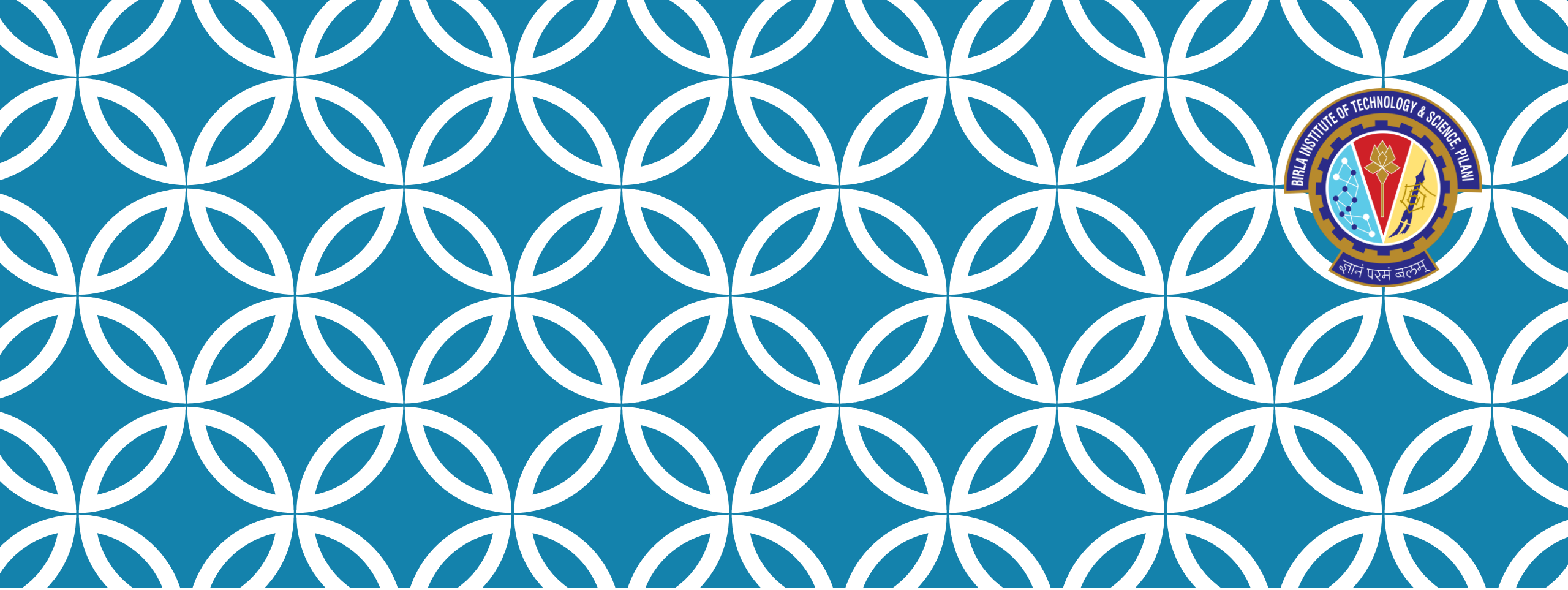


BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) ADAPTER DESIGN PATTERN

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

RECAP

Queue can also be implemented using one User stack and one Function Call Stack.

```
int deQueue() {
  if (s.empty()) {
     return -1;
  }
  int x = s.top();
  s.pop();
  if (s.empty())
    return x;
  int item = deQueue();
  s.push(x);
  return item;
}
```

```
void enQueue(int x) {
   s.push(x);
}
```

ADAPTERS DESIGN PATTERN

- What is an adapter/ a wrapper?

Deque
insertFront()
insertBack()
removeFront()
removeBack()
Size()
Empty()

```
typedef string Elem;
DequeStack {  // stack as a deque
   public:
        DequeStack();
        int size() const;
        bool empty() const;
        const Elem& top();
        void push(const Elem& e);
        void pop();
   private:
        LinkedDeque D; };
```

```
Queue is Empty!
Tim enqueued into the Queue.
Alex enqueued into the Queue.
Bob enqueued into the Queue.
FRONT -> Tim Alex Bob
Size of the Queue = 3.
Elem at the FRONT: Tim
Dequeuing...
Elem at the FRONT: Alex
FRONT -> Alex Bob
Smith enqueued into the Queue.
FRONT -> Alex Bob Smith
```

```
template <typename E>
void Queue<E>::enqueue(E elem)
{
    dq. insertBack(elem);
}

template <typename E>
void Queue<E>::dequeue()
{
    if (dq.empty())
        throw("Queue Underflows!");
    dq.removeFront();
}
```

Lab 7:next week's lab: queue using deque.

```
203 void DequeStack::push(const Elem& e)
204 {
205     D.insertFront(e);
206 }
```

```
206 void DequeStack::pop(){
207     if (empty())
208         cout<<"pop of empty stack\n";
209     D.removeFront();
210 }
```

VECTOR OR ARRAY LIST ADT

- Vector is a sequential container to store elements.

- Vector is dynamic in nature.

- Which one takes more time to access elements? Arrays or Vectors.

Main methods:

at(i), set(i, o), insert(i, o), erase(i), size(), empty()

Assuming that the vector is initially empty, find out the contents of V for the following operations?

insert (0, 60)

at (1)

insert (1, 70)

insert (0, 40)

erase (1)

set (1, 50)

SIMPLE ARRAY-BASED IMPLEMENTATION

Use an array ***A*** of size ***N***

A variable ***n*** keeps track of the size of the array list (number of elements stored)

How will you implement?

at (i)

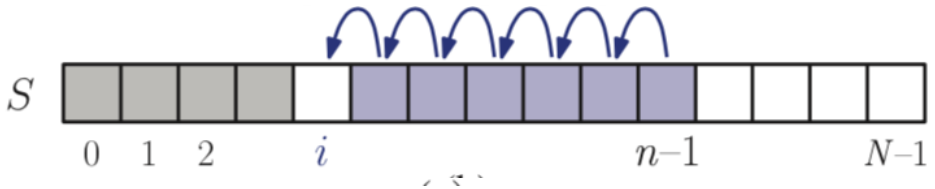
set (i, o)

insert (i, o)

erase (i)

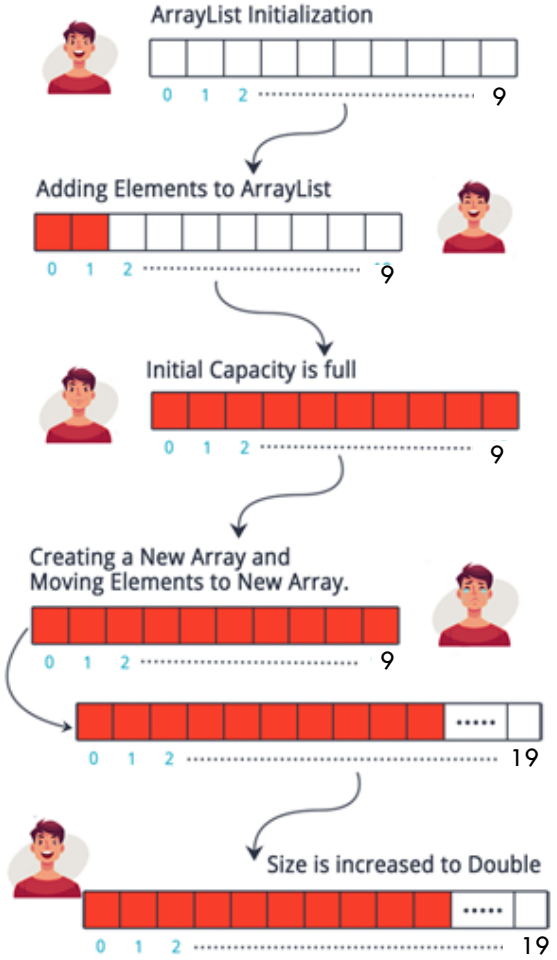
Algorithm insert(*i*,o):

Algorithm erase(*i*):



What would be the performance of a vector realized by an array?

```
1 : Insert
2 : Erase
3 : Get element at index i
4 : Get size
5 : Check if vector is empty
6 : Exit
1
Enter index and element :
0 10
3
Enter index :
0
10
1
Enter index and element :
1 20
4
Getting size
2
5
Vector is not empty
2
Enter index :
1
4
Getting size
1
```



AMORTIZATION (A DESIGN PATTERN)

- We compare the incremental strategy and the doubling strategy by analyzing the total time $T(n)$ needed to perform a series of n insert(o) operations.
- We assume that we start with an empty vector represented by an array of size 1
- We call **amortized time** of an insert operation as the average time taken by an insert over the series of operations, i.e., $T(n)/n$