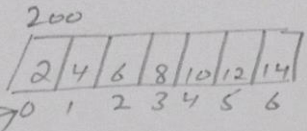
P8 1053.

# Array vs Linked List.

## Array

1) **Cost of Accessing an Element:**

int A[7]



base Address 200

Cout<< A[4];

↳ 10

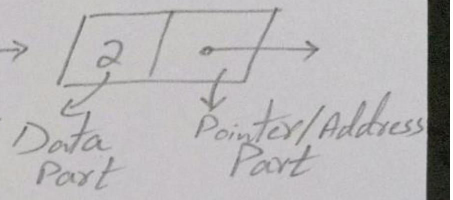
* Algorithmic Complexity

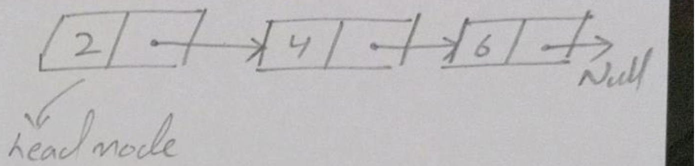
→ Constant Time

O(1).

## Linked List

Node →





* To access an element in the linked list at a Particular Position, we first need to start at the head node, then we go to the second node and see the address of the third node.

* In the worst case, to access the last element in the list, we will be traversing all the element in the list.

* Algorithmic Complexity

→ Linear Time

O(n).

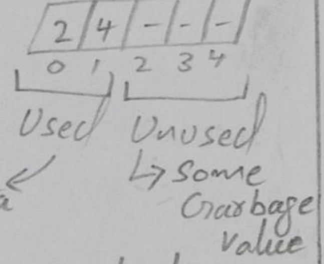
## Array

2) Memory Requirement:

→ Fixed Size

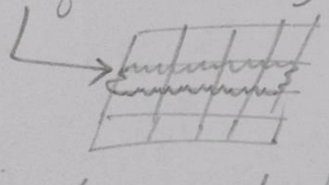
* We need to know the size of array before Creating it.

int A[5]



→ 5 × 4 = 20 bytes

* it is Created as one Contigous clock of memory

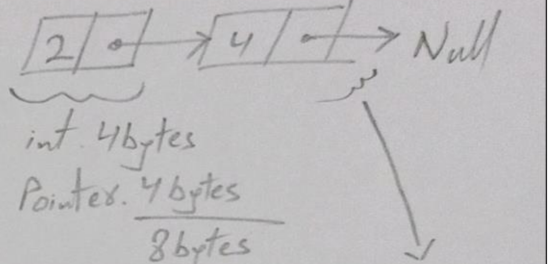


* Memory should be available as one large block.

## Linked List

→ No unused Memory

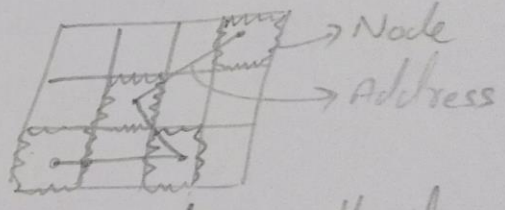
→ Extra memory requirement For Pointer Variable.



int. 4bytes
Pointer. 4 bytes
8bytes

A node takes 8bytes

2 × 8 = 16b



* Stored in scattered Memory block.

## Array

## Linked List

3) **Cost of Inserting an Element:**

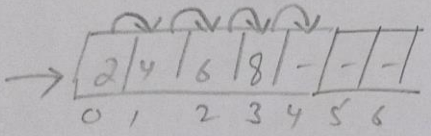
int A[7]

| 2 | 4 | 6 | 8 | - | - | - |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |

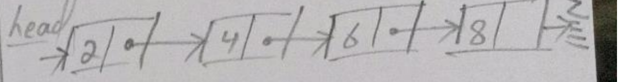
→ If array is not full.

→ 3 Scenario For Insertion

| Position | Algorithmic Complexity |
|---|---|
| a) at beginning | O(n) |
| b) at end | O(1) |
| c) at ith Position | O(n) |

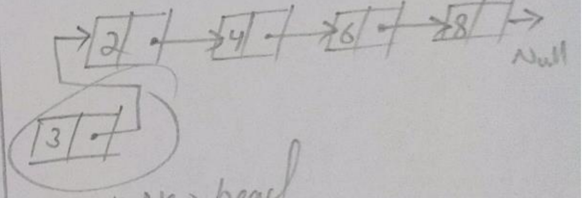


* we have to shift all element for insertion at beginning.



→ 3 Scenario For Insertion

| Position | Algorithmic Complexity |
|---|---|
| a) at beginning | O(1) |
| b) at end | O(n) |
| c) at ith Position | O(n) |



* New head

4) **Ease of use**

→ Array is Simple

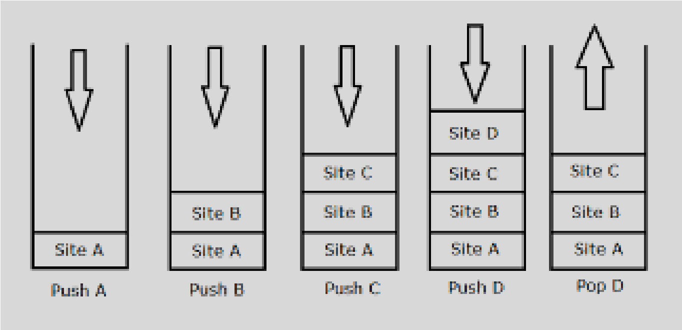
→ Linked List is Complex

### What Is a Stack?

A stack is a linear data structure which allows the adding and removing of elements in a particular order.

### Implementing Stack Functionalities Using a Linked List

Stack can be implemented using both arrays and linked lists. The limitation, in the case of an array, is that we need to define the size at the beginning of the implementation. This makes our stack static. It can also result in *"stack overflow"* if we try to add elements after the array is full. So, to alleviate this problem, we use a linked list to implement the stack so that it can grow in real time.



Stack: LIFO