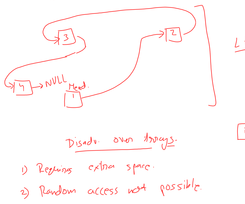
1, 2, 3, 4, 5.
x x x x ...

Arrays



L.L.

Disadv. over Arrays.

1) Requires extra space.
2) Random access not possible.
3) Not cache-friendly.

Adv. over Arrays.

1) Insert at beginning take O(1) time.
2) No continuous set of memory need to store whatever length of L.L. we want to keep.

```
Node {
    int value;
    Node* next;

}
```

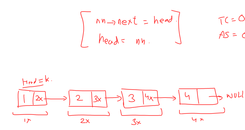
Insert at Begin.

i/p: 2 → 3 → 4 → 5 → N , 1

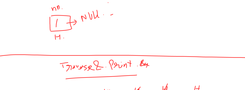
o/p. 1 → 2 → 3 → 4 → 5 → N

```
[ nn->next = head
  Head = nn ]
```

TC = O(1)
AS = O(1).



H → NULL. Value = 1.



Traverse Point to

H H H H H H
1 → 2 → 3 → 4 → 5 → NULL. X

1 2 3 4 5
H.
1 → 2 → 3 → 4 → 5 → NULL.
t t t t t t

TC = O(n)
AS = O(1)

1 2 3 4 5

Insert at End.

i/p: H.
1 → 2 → 3 → 4 → 5 → N. 6.

o/p = H.
1 → 2 → 3 → 4 → 5 → 6 → N

H T t
1 → 2 → 3 → 4 → 5 → N.
t → 6
nn.

t → next = nn.

H.
1 → 2 → 3 → 4 → 5 → N
t t t t t

temp = head
while (temp.next != NULL)
temp = temp.next

TC = O(n)
AS = O(1).

H.
1 → 2 → N.
t
2.

Insert at Pos

i/p: 1 → 2 → 3 → 4 → 5 → 6 → N.

o/p: 1 → 2 → 3 → 4 → 7 → 5 → 6 → N.

pos = 4 , value = 7.

t t2
1 → 2 → 3 → 4 → 5 → 6 → N.
7

make temp by (pos-1) hops

```
[ t2 = t->next.
  t->next = nn
  nn->next = t2 ]
```

H.
1 → 2 → 3 → 4 → 5 → 6 → N
t t t t

pos = 4.
hops = 4-1 = 3

TC = O(min(pos, n)).
AS = O(1).

t t t t t2
1 → 2 → 3 → 4 → N , pos = 5.
5 value = 5

hops = 4
t
t
3

```
Node temp = head;
int hops = 0;
while (hops < pos - 1 && temp->next != NULL) {
    temp = temp->next;
    hops++;
}
```

T T
← 1 → 2 → 3 → 4 → 5 → .

i/p: H
1 → 2 → 3 → 4 → 5 → N.
T

o/p: H.
1 → 2 → 3 → 4 → N.
T

Delete at Begin.

i/p: H.
1 → 2 → 3 → 4 → N

o/p. H
2 → 3 → 4 → N.

t H
2 → 3 → 4 → N.
t.

```
[ t = H.
  H = H->next
  delete(t) ]
```

TC = O(1)
AS = O(1).

Delete at End.

i/p: 1 → 2 → 3 → 4 → 5 → N

o/p: 1 → 2 → 3 → 4 → N.

TC = O(n).
AS: O(1).

t t2
1 → 2 → 3 → 4 → 5 → N.
N.

```
[ t2 = t->next
  t->next = NULL
  delete(t2). ]
```