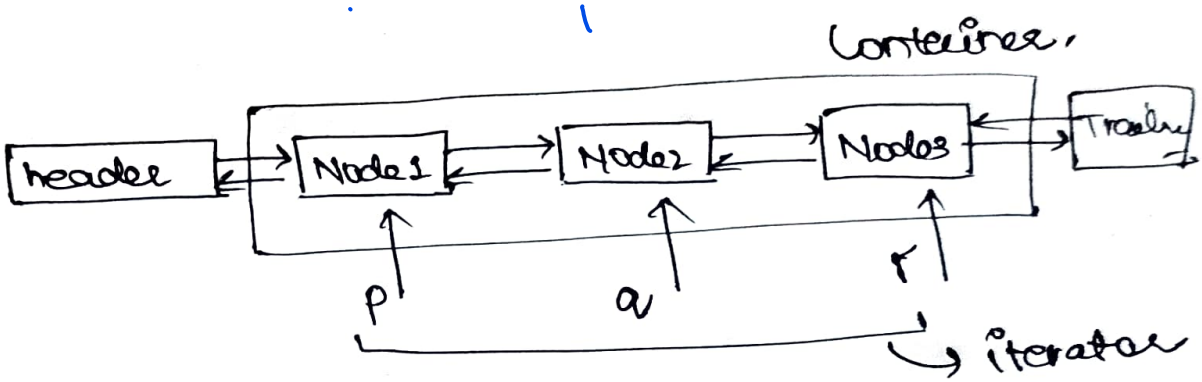
Sequences using DLL :

We have iterator (extension of position class) class which has

*p (dereferencing)

++p (next)

--p (previous)



In this case,

If we have following operation,

```
atIndex(int i)
    =
  ↳ Begins with iterator p = begin()
    For(j=0; j<i; j++) ++p;
    return p;
  }
```

Ex: what is atIndex(1)?

~~iterator p = Node1->p~~

~~For(j=0; j<1; j++)~~

j	p
0	p
1	q
2	r

So, it returns q.

=

Clearly, O(n)

```
indexOf( Iterator p) {
    Iterator q = begin()
    int j = 0;
    while( q != p ) {
        ++q ; ++j
    }
    return j
}.
```

Q4 Find indexOf(r).?

Start with $q' = p$ (true for $q' == r$)

Set $j = 0$.

Check if $q' == r$? —Yes→ return j ($j = 2$ = index)

No (No for $q' == p$, $q' == q$)

↓

Increment q' and j

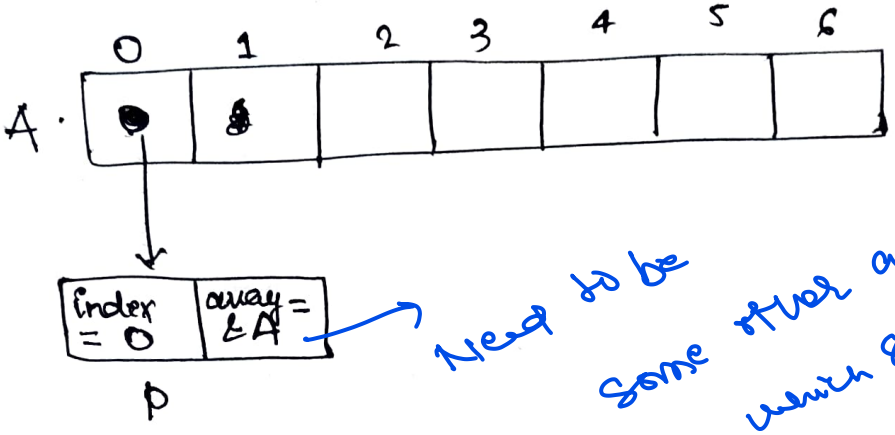
n time

clearly $O(n)$ complexity.

What if we implemented sequences with an array?.

Implementing Sequence with array.

position class {
    (index (i)
    reference to array (say A)
}



Need to be some other array which stores element's

$*p = A[index]$

But what is position of p?

No way to reference.

If insertion/deletion occurs,

rank changed not positions ✓

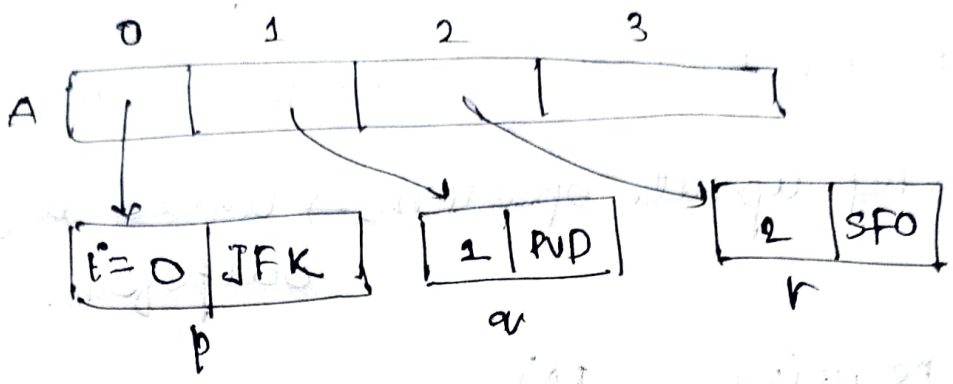
If there is no reference to position

how to update position?

Naive approach:

position {
    index i
    element e
}

So, $*p = e$



Now, accessing index of position object is O(1)

ex. Index of p ?

return p.index = 0 O(1)

and ~~position~~ which position is at index 2 ?

~~just iterate over array A,~~

~~and~~ return A[2] O(1)

But while inserting / deleting -
position need to be updated
so, O(n) time in worst case.

What if array is circular ?

→ Insert front would take O(1)
Back

Comparison.

Array-based implementation → rank based access

Equal on all other access like insert, del.

Regarding update operations → Linked list outperform.

Array requires O(N)
↳ size of array.

Linked List - O(n)
↳ no of ~~elem~~ nodes.