

27/02/2024

Page No.	
Date	

Day 1 OF DSA

* Tasks :

Check Box # Array.

- Basic operations; Insertion, deletion, Searching.

- Simple manipulations: Reversal, rotation

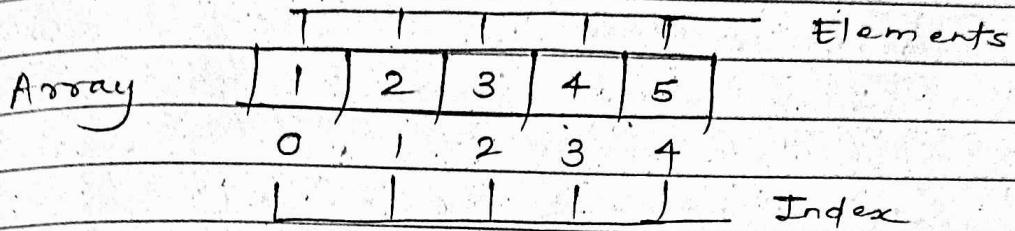
- Implement basic array operations - insertion, deletion, searching.

- Solve simple array manipulation problems.

- Raj Kashid

- Array :-

collection of items of same data type stored at contiguous memory locations.



- Basic Terms of Array:-

- **Array Index** - Array elements identified by index.
Index start from 0.
- **Array element** - Elements are items stored in an array.
- **Array Length** - length is determined by no. of elements it can contain

- Representation of Array:-

```
int arr[5];
char arr[10];
float arr[15];
```

Static Array :-

Name

int array [10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }
 data Type Elements

Dynamic Array:-

int *array = new int[5];

* Types of Arrays:-

(1) One-dimensional Arrays (only Row)

Elements	1	2	3	4	5
Index	0	1	2	3	4

(2) Two-dimensional Arrays (Row & column)

Col →		0	1	2
Row ↓	0	0	1	2
1	3	4	5	
2	6	7	8	

(3) Three-dimensional Arrays :- (2-Dimensional No. of 2D Arrays Arrays)

Col →		0	1	2
Row ↓	0	1	2	3
1	4	5	6	
2	7	8	9	

* Types of Array Operations

- Traversal - Traverse through array elements
- Insertion - Insert elements } OF An
- Deletion - Delete elements } Array
- Searching - Search an element
- Sorting - Maintain order of elements }

* Subarrays; Subsequences and Subsets in Array :-

(1) Subarray - contiguous part of array.

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

$$n = 4 = \frac{4 * (5)}{2}$$

$$= \frac{20}{2}$$

Subarrays = 10

The diagram shows the array [1, 2, 3, 4] with various double-headed arrows indicating subarrays. Arrows of length 1 point to each individual element (1, 2, 3, 4). Arrows of length 2 point to pairs (1, 2), (2, 3), and (3, 4). Arrows of length 3 point to triplets (1, 2, 3) and (2, 3, 4). A single long arrow of length 4 points to the entire array [1, 2, 3, 4].

(2) Subsequence -

A sequence by removing
0 or more elements

$$\text{Formula} = (2^n - 1)$$

$$n = 4 = (2^4 - 1)$$

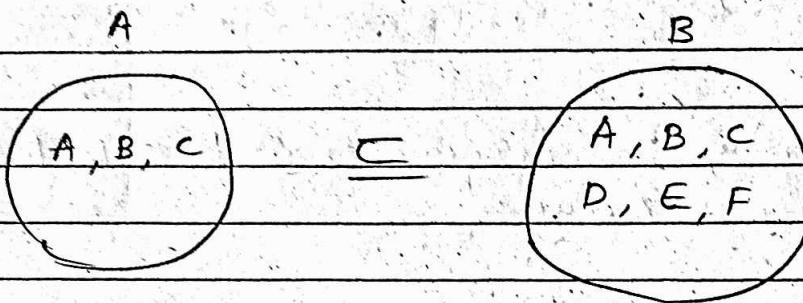
$$16 - 1$$

$$\text{Subsequence} = 15$$

⑤ Subset -

A set has all its elements belonging to other sets.

$$A \subset B$$



* Basic operations in Array:-

① Array / searching:

operatⁿ of finding a particular element or a group of elements in the array.

- Linear search $O(N)$ →
- Binary search $O(\log_2 N)$ ↔ | ↔
- Tertiary search. $O(\log_3 N)$ ↔ | ↔ | ↔

(2) Reverse an Array :-

Input - {1, 2, 3, 4}

Output - {4, 3, 2, 1}

(I) Array Reverse Using an Extra Array :-

- Create a New Array of same size as the original array.
- Copy elements from the original array to the new Array in reverse order.

* Unlogical Approach :-

New Array

```
int revArr[size];
For( int i=0; i<size; i++) {
```

revArr[i] = arr[size - i - 1];

}

Original Array

arr =

1	2	3
0	1	2

 size = 3 .

1) $i=0$

$$\begin{aligned} \text{revArr}[0] &= \text{arr}[3 - 0 - 1]; \\ &= \text{arr}[2] \end{aligned}$$

revArr[0] = 3 revArr

3	.	.
0	1	2

2) $i=1$

$$\begin{aligned} \text{revArr}[1] &= \text{arr}[3 - 1 - 1]; \\ &= \text{arr}[1] \end{aligned}$$

revArr[1] = 2 revArr

3	2	.
0	1	2

3) $i = 2$

$$\text{rev arr}[2] = \text{arr}[3 - 2 - 1];$$

$$= \text{arr}[0]$$

$$\text{rev arr}[2] = 1$$

3	2	1
0	1	2

Time complexity = $O(n)$

Space complexity = $O(n)$.

(#) Array reverse Using a Loop (In-place)

1) two pointers (start & end)

2) swap (start & end) pointers

3) move start ++ to the end &
end -- to the start.

* Approach 1-

while (start < end).

 int temp = arr[start];

 arr[start] = arr[end];

 arr[end] = temp;

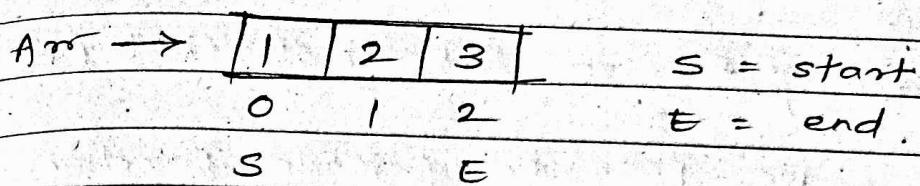
 start ++

 end --;

}

Day Run

Page No.		
Date		



① $s = 0$ & $e = 2$

$$\text{temp} = \text{arr}[0] = 1$$

$$\text{arr}[0] = \text{arr}[2] = 3$$

$$\text{arr}[2] = \text{temp} = 1$$

$s++;$

$E--;$

② $s = 1$, & $e = 1$

$$\text{temp} = \text{arr}[1] = 2$$

$$\text{arr}[1] = \text{arr}[2] = 2$$

$$\text{arr}[1] = \text{temp} = 2$$

$s++;$

$E--;$

③ $s = 2$ & $e = 0$ loop breaks ($s < e$)

Time complexity = $O(n)$

Space complexity = $O(1)$

(III) Array Reverse Inbuilt Methods (Non In-place)

we have ready made function
in C++ called reverse

Add algorithm header

Syntax :-

`reverse(arr, arr + length);`

& $length = \text{sizeof}(arr) / \text{sizeof}(arr[0]);$

Time complexity = $O(n)$

Space complexity = $O(n)$

(IV) Array Reverse Recursion (In-place or Non-In-place) :-

① Define a recursive function that takes array as input

② swap First & last elements

③ Recursively call the function with remaining subarrays.

* Approach :

```
Void reverseArray (int arr[], int start,
                   int end) {
```

Base case \rightarrow if ($start \geq end$) return;

Small calculate $\left\{ \begin{array}{l} \text{int temp} = arr[start]; \text{int arr}[start] = \\ arr[end] - temp \end{array} \right. \quad \left. \begin{array}{l} arr[end] = temp \\ arr[end]; \end{array} \right.$

Recursive call $\left\{ \begin{array}{l} \text{reverseArray} (arr, start+1, end-1); \end{array} \right.$

}

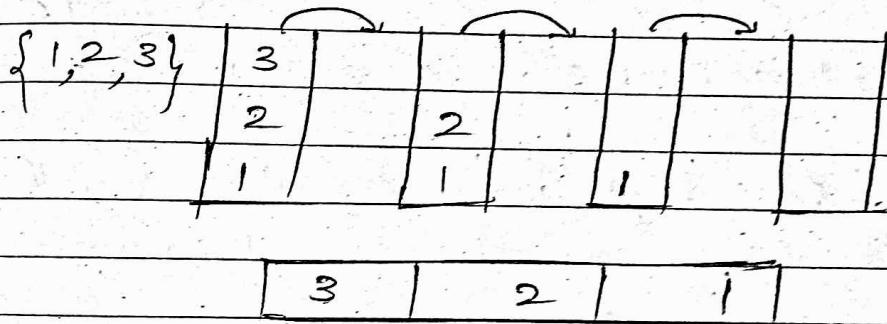
Non-In-place = Using New things
In-place = Using Existing one

Time Complexity = $O(n)$

Space complexity = $O(n)$ - Non-In-place
 $O(\log n)$ - In-place

(II) Array Reverse Stack. (Non-In-place)

- ① Push each element of the array onto a stack
- ② Pop elements from the stack to form reversed array.



* Approach :-

stack <int> stack;

```
for( int i=0; i< size; i++ )  
    stack.push( arr[i] );
```

}

```
for( int i=0; i< size; i++ )  
    arr[i] = stack.top();  
    stack.pop();
```

y

Time complexity = $O(n)$

space complexity = $O(n)$

Time Complexity = $O(N)$
Space Complexity = $O(N)$

Page No.		
Date		

* Array left rotation by d positions:-

Approach

* Three rules:-

- ① First store the elements from d to $n-1$ into the temp Array.

arr[] = { 1, 2, | 3, 4, 5, 6, 7 }, $d = 2$
 $2 \rightarrow 6$ $|d \rightarrow n-1|$
temp[] = [3, 4, 5, 6, 7]

- ② Store first d elements of the original array into the temp Array.

temp[] = [3, 4, 5, 6, 7, 1, 2] $|0 \rightarrow d|$

- ③ Copy back the elements of the temp array into the original array.

arr[] = temp[] $|arr \leftarrow temp|$
arr[] = [3, 4, 5, 6, 7, 1, 2]
X X X

- Approach:- int temp[n]; int k=0;

① { for(i=d → n)
 { temp[k] = arr[i]; $d \rightarrow n$
 k++; } }

② { for(i=0 → d)
 { temp[k] = arr[i]; $0 \rightarrow d$
 k++; } }

③ { for(i=0 → n)
 { arr[i] = temp[i]; $0 \rightarrow n$
 } }

Time complexity: $O(N \times d)$

Space complexity: $O(1)$

Page No.	

Approach :-

* Rotate one by one :-

- ① Store the first element of the array in a temp variable
 - ② Shift the rest of the elements in the original array by one place
 - ③ Update the last index of the array with temp variable.
- Repeat the above steps for the no. of left rotation

X — X — X — X —

- Approach :-

```
void Rotate(int arr[], int d, int n){  
    int p = 1;
```

```
    while(p <= d)
```

```
    {
```

```
        int last = arr[0];
```

```
        for(int i = 0; i < n - 1; i++) {  
            arr[i] = arr[i + 1];
```

```
        }
```

```
        arr[n - 1] = last;
```

```
        p++;
```

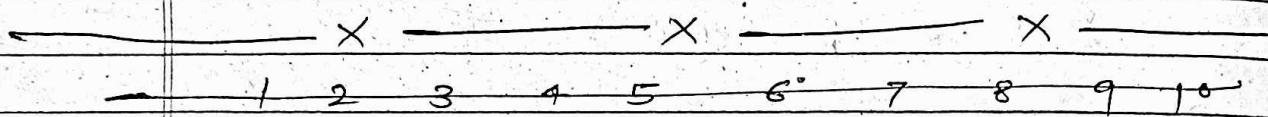
```
}
```

~~# Approach 3~~

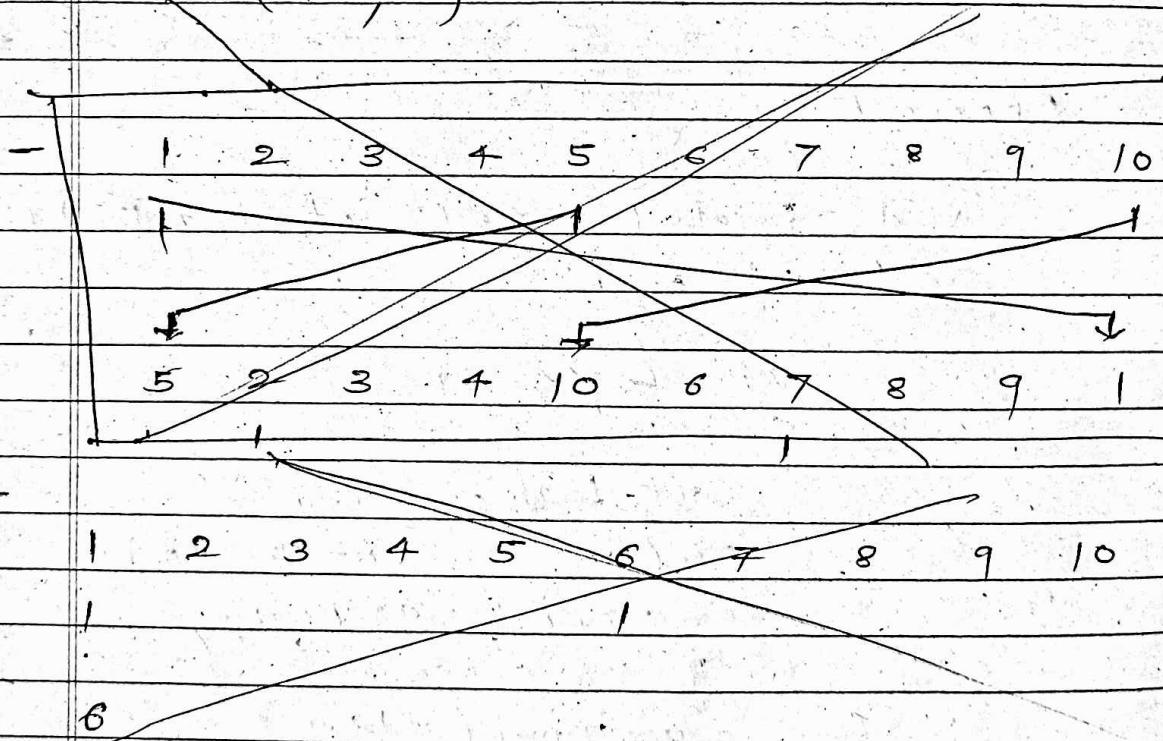
* A Juggling Algorithm :-

① Take a GCD of (array length, d places)

② Find the gap between two elements
& move it.



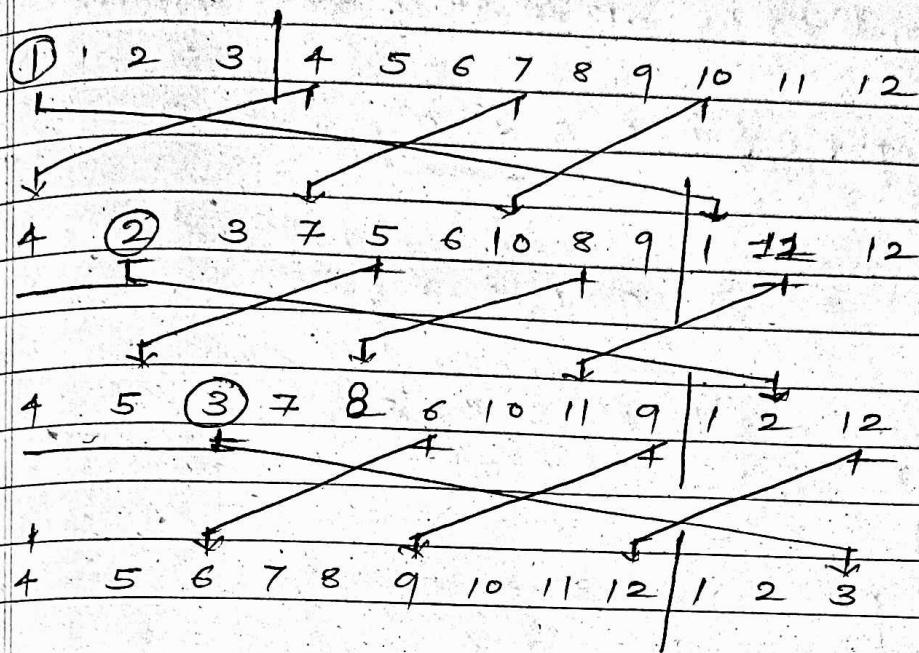
~~$\text{GCD}(10, 5) = 5$~~



$k=3$

$\text{GCD}(12, 3) \rightarrow 3$

Page No.		
Date		



Time complexity : $O(N)$

Space complexity : $O(1)$

* Print array after it is right rotated

Input = { 1, 3, 5, 7, 9 } , k = 2

Output = { 7, 9, 1, 3, 5 }

Input = { 1, 2, 3, 4, 5 } , k = 4

Output = { 2, 3, 4, 5, 1 }

* Approach :- By 2 formulae.

$$k = k \% n;$$

For (int i=0; i < n; i++) {

 if (i < k)

 cout << a[n+i-k] << " ";

 } else

 {

 cout << (a[i-k]) << " ";

 }

}

$$arr[j] = \{ 1, 2, 3, 4 \}, k=2$$

0 1 2 3

① $i=0$

$$0 < 2 = arr[4+0-2] \quad [n+i-k]$$

$$arr[2] = 3$$

② $i=1$

$$1 < 2 = arr[4+1-2] \quad [n+i-k]$$

$$arr[3] = 3$$

③ $i=2$

$$2 < 2 \quad \times$$



$$arr[2-2] = arr[0] = 3$$

$[i-k]$

④ $i=3$

$$3 < 2 \quad \times$$



$$arr[3-2] = arr[1] = 3$$

$[i-k]$

Time complexity = $O(n)$

Space complexity = $O(1)$

// Approach 2) - Reversing the Array.

- Three Rules - $0 \quad k \quad n$

1) reverse last k elements

$$\begin{array}{c|ccccc} 1 & 2 & 3 & 4 \\ \hline & & & & \end{array}$$

2) reverse first k elements

$$\begin{array}{c|ccccc} 2 & 1 & 4 & 3 \\ \hline & & & & \end{array}$$

3) Reverse Entire Array.

$$\begin{array}{ccccc} 3 & 4 & 1 & 2 & \checkmark \\ \hline & & & & \end{array}$$

```
int k = k % n;
```

```
int i, j;
```

```
for( i = n - k, j = n - 1; i < j; i++, j--) {
```

Step 1 :-

```
    int temp = arr[i];
```

```
    arr[i] = arr[j];
```

```
    arr[j] = temp;
```

}

```
for( i = 0, j = n - k - 1; i < j; i++, j--) {
```

```
    int temp = arr[i];
```

```
    arr[i] = arr[j];
```

```
    arr[j] = temp;
```

}

Step 2:-

for (int i = 0; j = n - 1; i < j; i++, j--) {

```

int temp = arr[i];
arr[i] = arr[j];
arr[j] = temp;
    
```

}

Time complexity = $O(N)$

Space complexity = $O(1)$

Approach 3 :- Recursive Approach

$K = K \% n;$

1st part.

reverse (arr, arr + n - k);

2nd part

reverse (arr + n - k, arr + n);

Entire array.

reverse (arr, arr + n);

void rotateArray (int arr[], int n, int k) {

if ($k = 0$) { return; }

int temp = arr[n - 1];

for (int i = n - 1; i > 0; i--) {

arr[i] = arr[i - 1];

y

arr[0] = temp;

rotateArray (arr, n, k - 1);

y

1	2	3	4
0	1	2	3

$k = 2$
 $n = 4$

Recursive method :-

1)

$$\text{temp} = \text{arr}[3]$$

For

$$① \quad i = 4 - 3$$

$$\text{arr}[3] = \text{arr}\left[\frac{3}{2}\right];$$

1	2	3	3
---	---	---	---

②

$$i = 2$$

$$\text{arr}[2] = \text{arr}[1]$$

1	2	2	3
---	---	---	---

③

$$i = 1$$

$$\text{arr}[1] = \text{arr}[0]$$

1	1	2	3
---	---	---	---

~~④~~ ~~i = 0~~

$$\text{arr}[0] = 4$$

4	1	2	3
0	1	2	3

2)

rotateArray (arr, 4, 1)

$$\text{temp} = \text{arr}[3] = 3$$

For

$$① \quad i = 3$$

$$\text{arr}[3] = \text{arr}[2]$$

4	1	2	2
---	---	---	---

Prog No.	
Date	

② $i = 2$

$$\text{arr}[2] = \text{arr}[1]$$

4 1 1 3

③ $i = 1$

$$\text{arr}[1] = \text{arr}[0]$$

4 4 1 2

$$\text{arr}[0] = \text{temp}$$

3 4 1 2

3) rotateArray (arr, 4, 0)

if ($k == 0$) return;

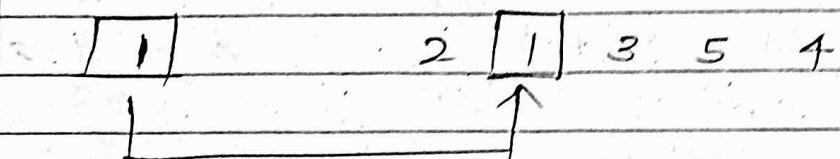
Time complexity : $O(N)$

space complexity : $O(1)$

* Search, Insert and Delete in an Unsorted Array.

Search operation -

key Array.



searching can be performed by
linear traversal.

```
for (int i=0; i<n; i++) {
```

```
    if (arr[i]==key) {
```

```
        return i;
```

}

```
    return -1;
```

Time complexity = $O(n)$

Space complexity = $O(1)$

Insert At the end.

```

    if (n >= capacity)
        return n;
    Index given
    by User
    arr size
    arr[n] = key;
    return (n+1);
  
```

Time complexity : $O(1)$
 Space complexity : $O(1)$

Insert At any position:-

```
For( int i = n-1; i >= pos; i-- ) {
```

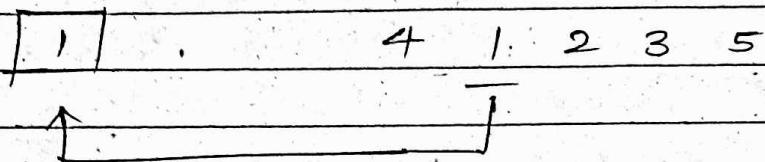
```
    arr[i+1] = arr[i];
```

y

```
    arr[pos] = x;
```

Time complexity : $O(N)$
 Space complexity : $O(1)$

// Delete operation :-



4 2 3 5

←

int pos = Find Element (arr, n, key);
 Find element index
 By Using linear search

if (pos == -1)
 cout << "Element Not found";

return n;

}

for (int i = pos; i < n - 1; i++) {

arr[i] = arr[i + 1];

y It will override the pos.
 value

return n - 1;

y

Time Complexity: O(N)

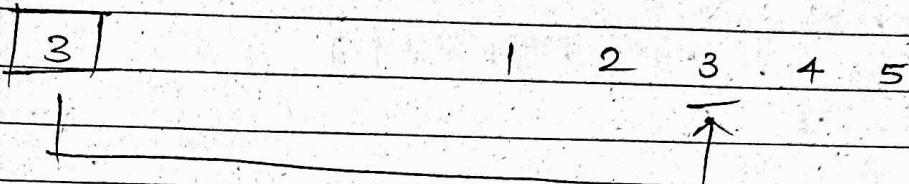
Space Complexity: O(1)

* Search, Insert and Delete in an sorted Array

①

search -

We Used Binary search.



```

int binarySearch(int arr[], int low, int high, int key)
{
    if (high < low)
        return -1;

    int mid = (low + high) / 2;
    if (key == arr[mid])
        return mid;
    if (key > arr[mid])
        return binarySearch(arr, (mid+1), high,
                            key);
    else
        return binarySearch(arr, low, (mid-1), key);
}

```

Time complexity : $O(\log(n))$
 space complexity : $O(\log(n))$

(2)

Insert:-

26 26 Break
40 26

```
For(i=n-1; i>=0 && arr[i] > key); i--
```

```
arr[i+1] = arr[i];
```

```
y
```

```
arr[i+1] = key;
```

```
return (n+1);
```

Time Complexity: O(N)

Space Complexity: O(1)

(3)

Delete:-

Just do

```
int pos = Element's index;
```

```
For(int i = pos; i < n-1; i++) {
```

```
arr[i] = arr[i+1];
```

```
y
```

```
return n-1;
```

Time complexity: O(N)

Space complexity: O(1)
