9th Nov, 23

Data Structure & Algorithms

Introduction to Array

(i) It stores same type of data.

(ii) It stores at contigeous location.

```
int a[1000]
```

- `int` → data type
- `a` → array name
- `1000` → size of array

Initialization:

(1)

```
int arr1[6] = {1, 3, 4, 2, 4, 5};
    for(int i = 0; i < 6; i++)
        cout << arr1[i] << " ";
    cout << endl;
//Output : 1 3 4 2 4 5
```

(2)

```
int arr2[] = {1, 3, 4, 2, 5, 55};
    for (int i = 0; i < sizeof(arr2)/sizeof(int); i++)
        cout << arr2[i] << " ";
//Output : 1 3 4 2 5 55
```

(3)

```
int arr3[5] = {0};
    for (int i = 0; i < 5; i++)
        cout << arr3[i] << " ";
//Outpur : 0 0 0 0 0
```

(4)

```
int arr4[5] = {2};
    for (int i = 0; i < 5; i++)
        cout << arr4[i] << " ";
//Outpur : 2 0 0 0 0
```

(5)

```
int arr5[5];
    for(int i = 0; i < 5; i++)
        cin >> arr5[i];
//Input...
    for(int i = 0; i < 5; i++)
        cout << arr5[i] << " ";
//Output...
```

(6)

```
int size;
    cin >> size;
    // int arr[size]; Not recommended..
    int arr[10000];
    for (int i = 0; i < size; i++)
        cin >> arr[i];

    for(int i = 0; i < size; i++)
        cout << arr[i] << " ";
```

```
cout << sizeof(arr);//4 * 10000 = 40000
```

Maximum element in Array:

```
int arr[5] = {2, 4, 44, 0, -22};
    int max = INT_MIN;
    for(int i = 0; i < 5; i++){
        if(arr[i] > max){
            max = arr[i];
        }
    }
    cout << "Max = " << max << endl;
```

Minimum element in Array:

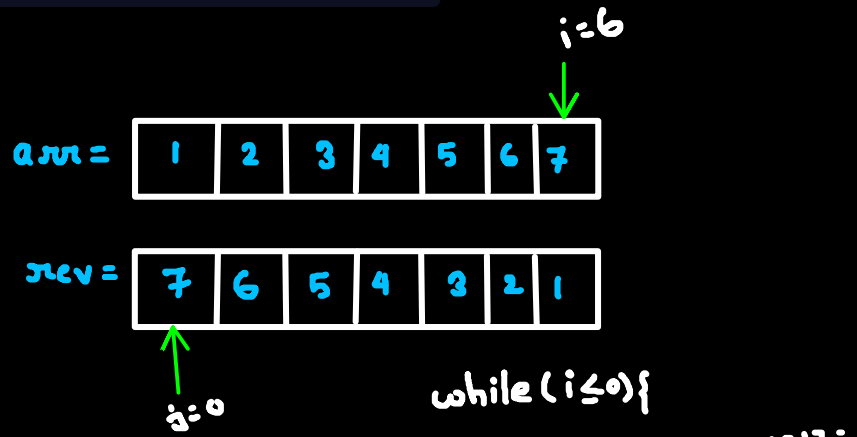
```
int arr[5] = {2, 4, 44, 0, -22};
    int min = INT_MAX;
    for(int i = 0; i < 5; i++){
        if(arr[i] < min){
            min = arr[i];
        }
    }
    cout << "Min = " << min << endl;
//Output : Min = -22
```

Linear Search:

```
int linear_search(int arr[], int N, int key){
    for(int i = 0; i < N; i++){
        if(arr[i]==key){
            return i;
        }
    }
    return -1;
}
```

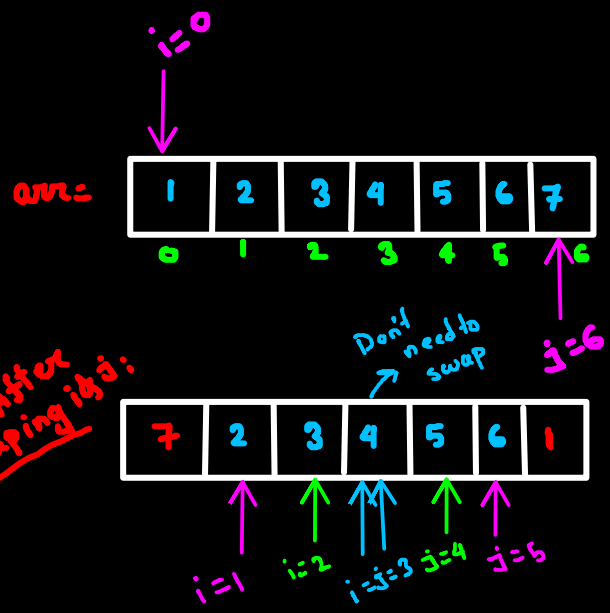
Reverse an Array:

1st:




```
while(i≤0){
    rev[j] = arr[i];
    j++;
    i--;
}
```

2nd:




```
i=0; j=6;
while(i<j){
    swap(arr[i],arr[j]);
    i++;
    j--;
}
```

```
int arr[6] = {1, 2, 3, 4, 5 , 6};

    int start = 0, end = 5;

    while (start < end)
    {
        swap(arr[start],arr[end]);
        start++;
        end--;
    };
    for(int i = 0; i < 6; i++)
        cout << arr[i] << " ";
//Output : 6 5 4 3 2 1
```

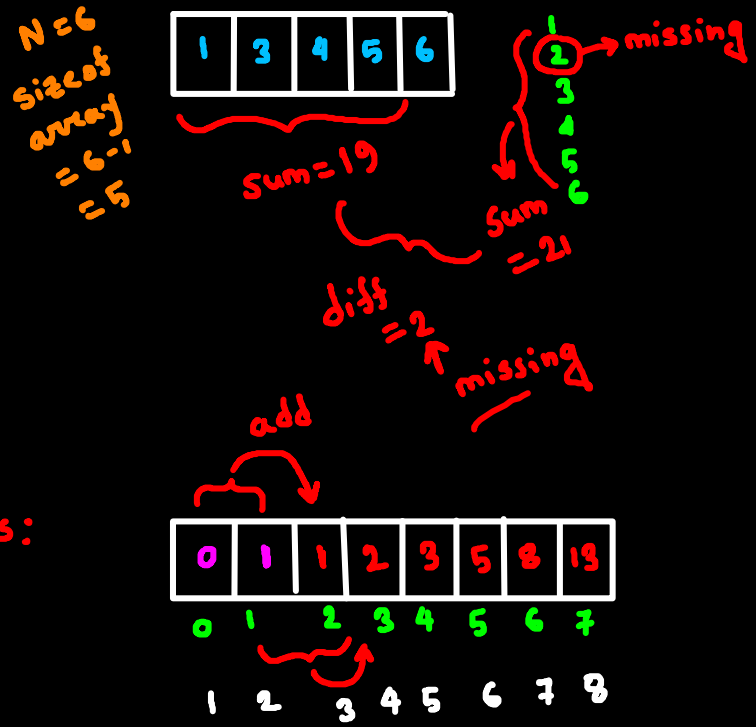
Second Maximum element:

```cpp
int arr[6] = { 2, 3, 4, 55, 222, 100};
    int max1 = INT_MIN;
    for (int i = 0; i < 6; i++)
    {
        max1 = max(max1, arr[i]);
    }
    cout << "1st Max element = " << max1 << endl;

    int max2 = INT_MIN;
    for (int i = 0; i < 6; i++)
    {
        if(arr[i]!=max1){
            max2 = max(max2, arr[i]);
        }
    }
    cout << "2nd Max element = " << max2 << endl;
//1st Max element = 222
//2nd Max element = 100
```

Missing Number:

Given an array of size N-1 such that it only contains distinct integers in the range of 1 to N. Find the missing element.



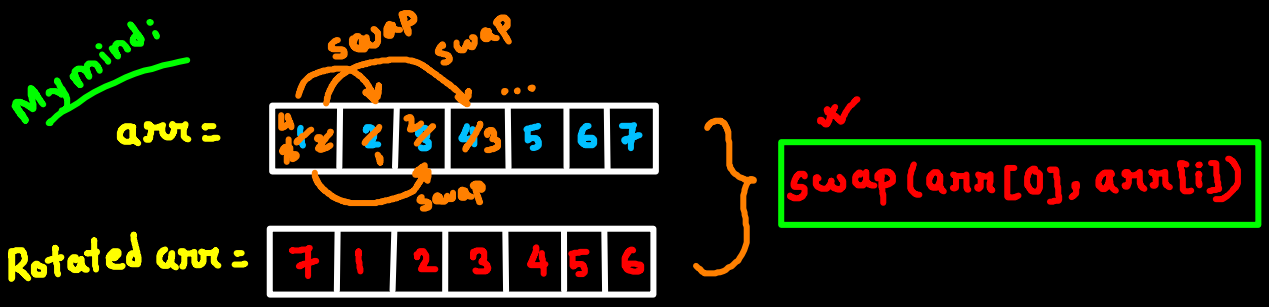
Fibonacci Series:


```cpp
arr[0] = 0;
    arr[1] = 1;
    for (int i = 2; i < n; i++)
        arr[i] = arr[i-1] + arr[i-2];

    cout << arr[n-1] << " ";
```

Array Rotation:

My mind:

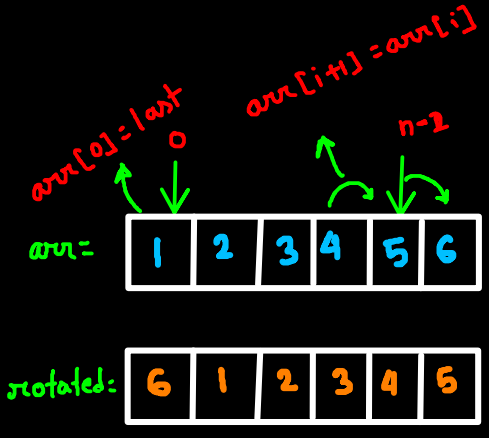



```
for (int i = 1; i < n; i++)
        swap(arr[0],arr[i]);
```

2nd Approach:

Take another array then again copy from that.

3rd Approach:



int last = arr[n-1]

```
int last = arr[n-1];
    for(int i = n-2; i >= 0; i--)
        arr[i+1] = arr[i];

    arr[0] = last;
```

Time and Space Complexity

Time complexity: It is the total time taken by an algorithm to run as a function of length of the input.

Big (O) — Worst Case

Big (Ω) — Best Case

Big (Θ) — Avg. Case

$3n+1 = O(n)$ → ignore contants

$2N^3 + N^2 + N = O(N^3)$

```
for(int i=0; i<n; i++){
    for(int j=0; j<n; j++){
        ...
    }
}
```
$O(n^2)$

```
for(int i=0; i<10; i++){
    cout<<i<<" ";
}
```
$O(10) \simeq O(1)$

```
for(int i=1; i<n; i++){
    for(int j=1; j<=i; j++){
        ...
    }
}
```

i=1	i=2	i=3	i=n
j=1	j=1-2	j=1-3	j=1-n
1time	2times	3times	ntimes

$1+2+3+\cdots n = \frac{n(n+1)}{2} = \frac{n^2+n}{2}$

$\therefore O(n^2)$

```
for(int i=1; i<=n; i++){
    for(int j=1; j<=i^2; j++){
        ...
    }
}
```

i=1	i=2	i=3	... i=n
j=1 time	j=1-4	j=1-9	j=n² times
	=4 times	=9 times	

$1+2^2+3^2+n^2 = \frac{n(n+1)(2n+1)}{6}$

$= n^3+n^2+n\cdots$

$= O(n^3)$

```
for(int i=1; i<=n; i++){
    for(int j=1; j<=m; j++){
        ...
    }
}
```
$O(nm)$

```
for(int i=1; i<=n; i++){
    for(int j=1; j<=i^2; j++){
        for(int k=1; k<=n/2; k++){
            ...
        }
    }
}
```

i=1	i=2	i=3	... i=n
j=1 time	j=4 times	j=9 times	j=n² times
k=1-n/2	k=1-n/2	k=1-n/2	k=1-n/2
=n/2 times	=4n/2 times	=9n/2 times	=n²×n/2 times

$\therefore \frac{n}{2} + \frac{4n}{2} + \frac{9n}{2} + \cdots n^2 n/2 = \frac{n}{2}(1+2^2+3^2+\cdots n^2)$

$= \frac{n}{2}\left(\frac{n(n+1)(2n+1)}{6}\right)$

$= O(n^4)$ ✓

```
for(int i=1; i<=n; i=i*2){
...
}
```

$i=1$ (2^0) 1 time, $i=2$ (2^1) 1 time, $i=4$ (2^2) 1 time, $i=8$ (2^3) 1 time ... $i=n$ (2^k) 1 time

$(k+1)$ times 1 ✓

$n = 2^k$

$\log n = k \log 2$

$\therefore (\log_2 n + 1) = O(\log_2 n)$

$\Rightarrow k = \frac{\log n}{\log 2} = \log_2 n$

$1 + 3 + 9 + 27 + 81 \ldots$

↓

$\log_3 n$

```
for(int i=n/2; i<=n; i++){
  for(int j=1; j<=n/2; j++){
    for(int k=1; k<=n; k++){
    ...
    }
  }
}
```

$n/2$, $n/2$, n

i, j, k independent...

↓

$n/2 \times n/2 \times n = O(n^3)$

```
for(int i=n/2; i<=n; i++){
  for(int j=1; j<=n; j=j*2){
    for(int k=1; k<=n; k=k*2){
    ...
    }
  }
}
```

$n/2$, $\log_2 n$, $\log_2 n$

$\frac{n}{2} \times \log_2 n \times \log_2 n = O(n(\log_2 n)^2)$

```
for(int i=1; i<=n; i++){
  for(int j=1; j<=n; j=j+i){
  ...
  }
}
```

$i=1$: $j=1$ to n, n times

$i=2$: $j=n/2$

$i=3$: $j=n/3$

...

$i=n$: $j=n/n=1$

Harmonic series

$\therefore (n + n/2 + n/3 + n/n) = n(1 + \frac{1}{2} + \frac{1}{3} + \cdots \frac{1}{n})$

$= n \log_e n$

Space complexity: It is the total space taken by an algorithm to run as a function of length of the input.

Auxiliary Space → which I have created
Total Space } w/

```
int a, b, c;
cin >> a >> b >> c;
int d = (a+b)/2;
int e = (a+c)/2;
```

∴ auxiliary = 2 = O(1)

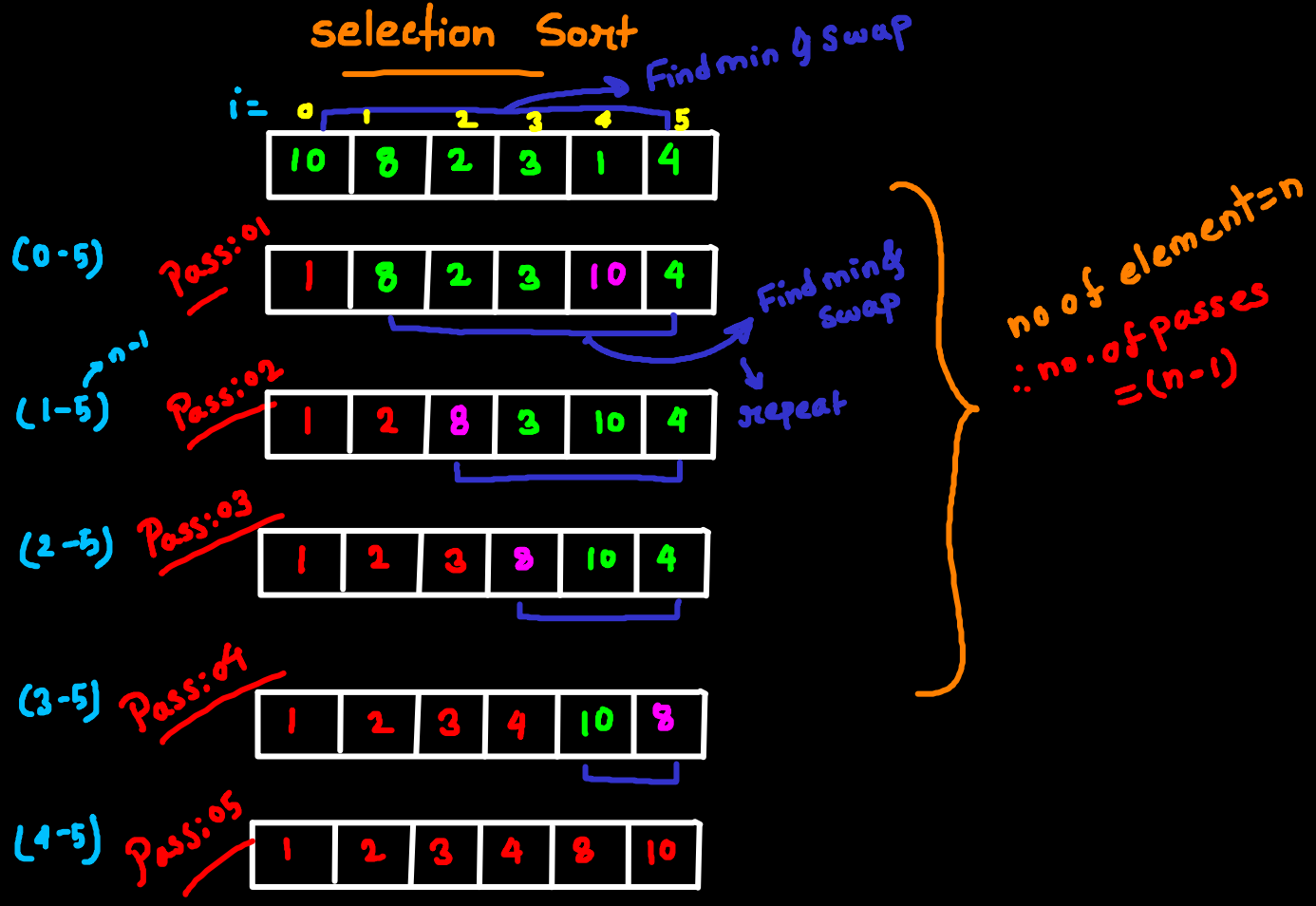
∴ Total = 5 = O(1)

Time:

$O(N!) > O(2^n) > O(n^3) > O(n^2) > O(n \log n) > O(n) > O(\sqrt{n}) > O(\log n) > O(1)$

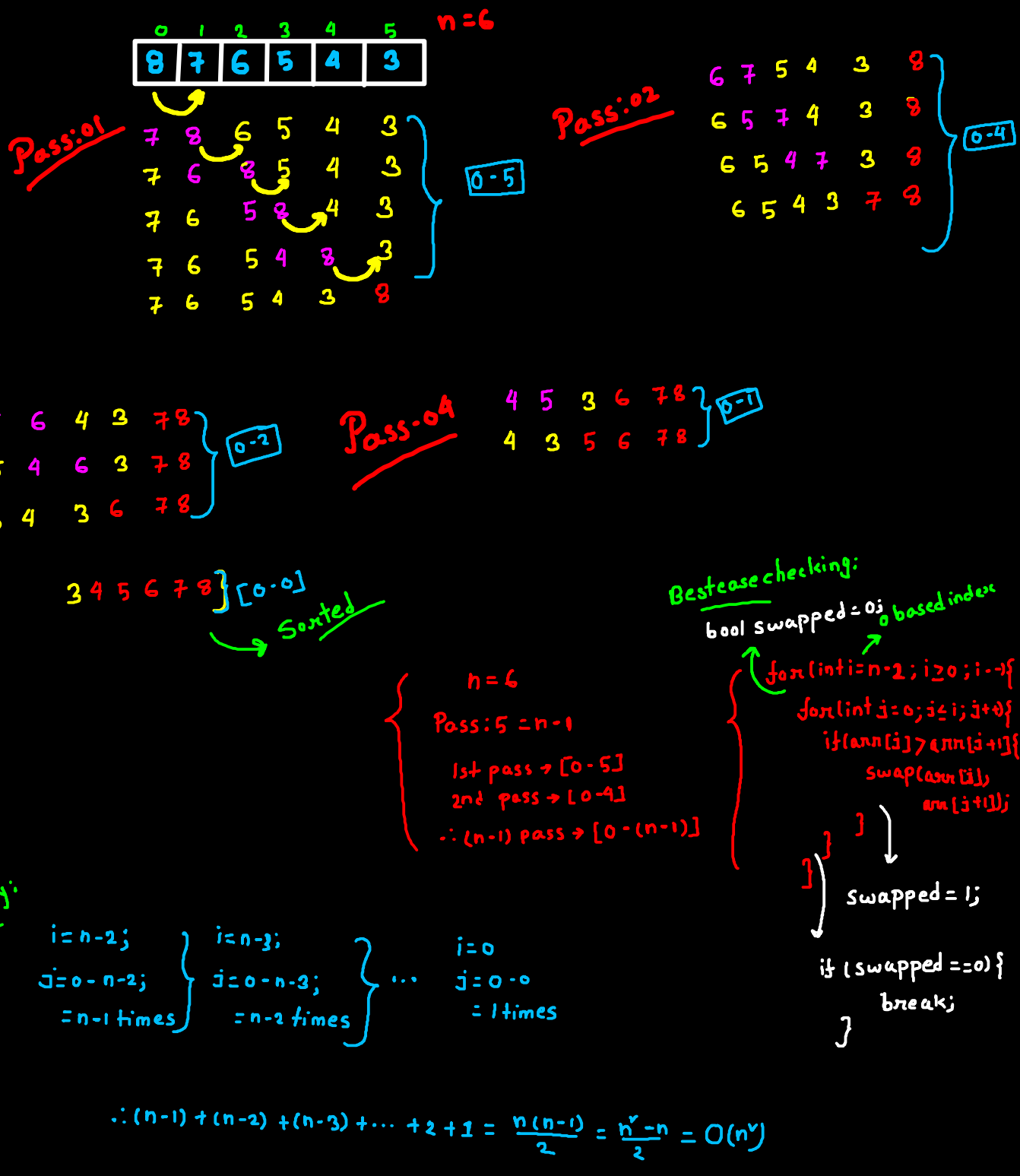
Sorting

selection Sort




```java
//O(n*n)
    for(int i = 0; i < n-1; i++){
        int index = i;
        for(int j = i+1; j < n; j++){
            if(arr[j]<arr[index])
                index = j;
        }
        swap(arr[i], arr[index]);
    }
```

Bubble Sort



n = 6

Pass : 5 = n-1

1st pass → [0-5]

2nd pass → [0-4]

∴ (n-1) pass → [0-(n-1)]

Besteasechecking:

```
bool swapped = 0;   // 0 based index
for(int i=n-2; i>=0; i--){
    for(int j=0; j<=i; j++){
        if(arr[j]>arr[j+1]){
            swap(arr[j], arr[j+1]);
            swapped = 1;
        }
    }
    if(swapped==0){
        break;
    }
}
```

Time Complexity:

i = n-2; j = 0 - n-2; = n-1 times

i = n-3; j = 0 - n-3; = n-2 times

...

i = 0 j = 0 - 0 = 1 times

$$\therefore (n-1)+(n-2)+(n-3)+\cdots+2+1 = \frac{n(n-1)}{2} = \frac{n^2-n}{2} = O(n^2)$$

```
void BUBBLE_SORT(int arr[], int n){
    for(int i = n-2; i >= 0; i--){
        bool swapped = false;
        for(int j = 0; j <= i; j++){
            if(arr[j]>arr[j+1]){
                swap(arr[j], arr[j+1]);
                swapped = true;
            }
        }
        if(swapped==false){
            break;
        }
    }
}
```

Insertion Sort

Let's already sorted...

0	1	2	3	4	5
6	5	4	3	2	1

n = 6

Pass: 01 — 5 6 | 4 3 2 1 } [1-0]

Pass: 02 — 5 4 6 3 2 1 } [2-0]
4 5 6 | 3 2 1

Pass: 03 — 4 5 3 6 2 1 } [3-0]
4 3 5 6 2 1
3 4 5 6 | 2 1

Pass: 04 — 3 4 5 2 6 1 } [4-0]
3 4 2 5 6 1
3 2 4 5 6 1
2 3 4 5 6 | 1

Pass: 05 — 2 3 4 5 1 6 } [5-0]
2 3 4 1 5 6
2 3 1 4 5 6
2 1 3 4 5 6
1 2 3 4 5 6

↓ Sorted

n = 6
pass = 5 = n-1
in each pass = n-1 to 0

```
void INSERTION_SORT(int arr[], int n){
    for(int i = 1; i < n; i++){
        for(int j = i; j > 0; j--){
            if(arr[j]<arr[j-1]){
                swap(arr[j],arr[j-1]);
            }else{
                break;
            }
        }
    }
}
```

```
for(i=1;i<n;i++){
  for(j=i;j>0;j--){
    if(arr[j]<arr[j-1]){
      swap(arr[j], arr[j-1]);
    }
    else{
      break;
    }
  }
}
```

Time Complexity:

i = 1;	i = 2;	i = 3;	...	i = n-1;
j = 1-1;	j = 2-1	j = 3-1;		j = (n-1) - 1
= 1 times	= 2 times	= 3 times		= n-1 times

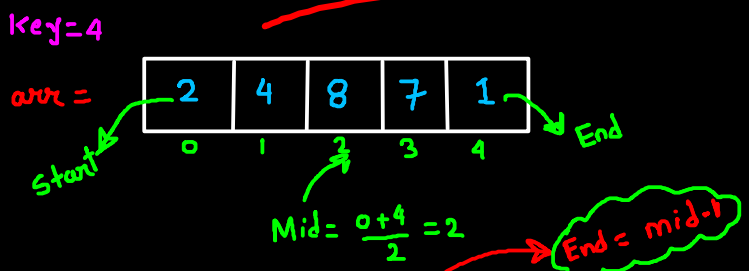
$1+2+3+4+\dots+(n-1) = \frac{n(n-1)}{2}$

$= O(n^2)$

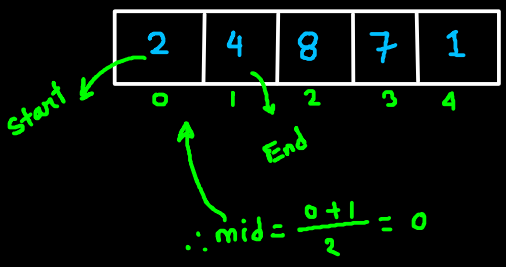
Best case complexity, $O(n)$, $\Omega(n)$ ✓

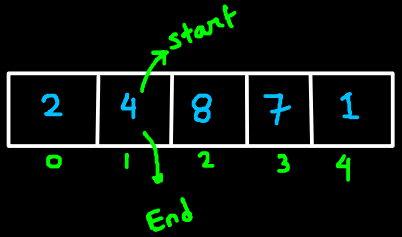
Avg-case = Sum of all cases / Total no. of cases

Binary Search



mid ≠ key, check in left as mid > key.





mid < key, check in right.

Start = mid+1

∴ mid = key

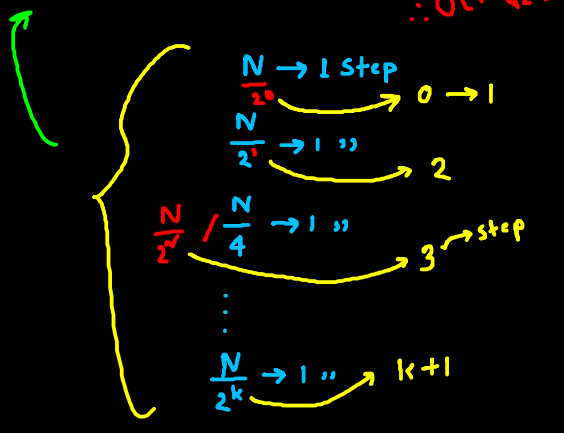
Found...

$N = 2^k$

$\Rightarrow \log N = k \log 2$

$\therefore k = \frac{\log N}{\log 2} = \log_2 N$

$\therefore O(\log_2 N)$




```
Start = 0
End = n-1
while (Start ≤ End){
    Mid = Start + End / 2
    if (key == arr[Mid]){
        found
        break
    }else if (key > arr[Mid]){
        Start = Mid + 1;
    }else{
        End = Mid - 1;
    }
}
```

imp — use otherwise may occure overflow...

$Mid = Start + \frac{End - Start}{2}$

```c
int BINARY_SEARCH(int arr[], int n, int key){
    int start = 0;
    int end = n-1;
    while(start <= end){
        int mid = (start+end)/2;
        if(arr[mid] == key){
            return mid;
        }else if(arr[mid] < key){
            start = mid + 1;
        }else{
            end = mid - 1;
        }
    }
    return -1;
}
```

Binary Search Problems

34. Find First and Last Position of Element in Sorted Array

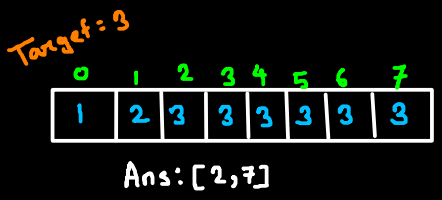
Medium 19.4K 468

Companies

Given an array of integers `nums` sorted in non-decreasing order, find the starting and ending position of a given `target` value.

If `target` is not found in the array, return `[-1, -1]`.

You must write an algorithm with `O(log n)` runtime complexity.



Pass: 01

start: 0, end: 7

$mid = \frac{0+7}{2} = 3$

arr[mid]: 3 ✓

But it maybe not the first

first must be in the leftside

So, when, arr[mid] = Target

then, first = mid;

end = mid-1;

By this way, we can find last position moving rightside.

when, arr[mid] = Target

then, last = mid;

start = mid+1;

```cpp
class Solution {
public:
    vector<int> searchRange(vector<int>& nums, int target) {
        int first = -1,last = -1;
        int start = 0, end = nums.size()-1,mid;

        //Finding 1st :
        while(start <= end){
            mid = start + (end - start)/2;
            if(nums[mid]==target){
                first = mid;
                end = mid-1;
            }else if(nums[mid]<target){
                start = mid+1;
            }else{
                end = mid -1;
            }
        }

        //Finding 2nd :
        start = 0, end = nums.size()-1;
        while(start <= end){
            mid = start + (end - start)/2;
            if(nums[mid]==target){
                last = mid;
                start = mid+1;
            }else if(nums[mid]<target){
                start = mid+1;
            }else{
                end = mid -1;
            }
        }
        vector<int>a(2);
        a[0] = first;
        a[1] = last;
        return a;
    }
};
```

35. Search Insert Position

Easy 15.1K 657

Companies

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

```
Input: nums = [1,3,5,6], target = 5
Output: 2
```

Example 2:

```
Input: nums = [1,3,5,6], target = 2
Output: 1
```

Example 3:

```
Input: nums = [1,3,5,6], target = 7
Output: 4
```

Target = 7

n = 8

0	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7

start = 0, end = 7, mid = $\frac{0+7}{2}$ = 3, index = n

arr[mid] = 3 < Target = 7

So, position of Target will be in right.

But may'nt be next pos. because they may still less num. No need to update the index as it is initialized as n.

Now, start = mid + 1, end = 7, mid = $\frac{7+4}{2}$ = 5

= 4

arr[mid] = 6 < Target = 7

Start = 5+1, end = 7, mid = $13/2$ = 6

∴ Found ✓

Target = 1

start = 0, end = 7, mid = $\frac{0+7}{2}$ = 3, index = n

arr[mid] = 3 > Target = 1

So, move left.

Update, index = mid = 3

as target is less so its position may be mid or less than that.

start = 0, end = 2, mid = 1

arr[mid] = 2 > Target = 1

index = 1 ✓ Done

```cpp
class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {
    int start = 0;
    int end = nums.size()-1;
    int index = nums.size();
    while(start <= end){
        int mid = start + (end - start)/2;
        if(nums[mid] == target){
            return mid;
        }else if(nums[mid] < target){
            start = mid + 1;
        }else{
            index = mid;
            end = mid - 1;
        }
    }
    //Evabe korleo hobe...
    // nums.insert(nums.begin() + start, target);
        return index;
    }
};
```