Dt 25/12/2022
Pg

Module- "Searching and Sorting"

Understanding Binary Search → O(logn) → Sorted order

Linear Search → O(n) → Random array → no order

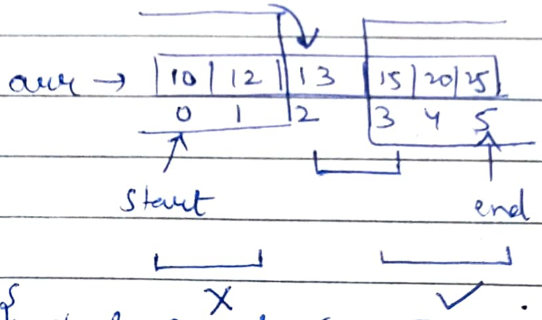
arr[] →

10	12	1	8	2	5
0	1	2	3	4	5

x
1
11

```
{
for(i=0 → n-1){
        if(arr[i] = = x) return i; }
  return -1;
}
```

Binary Search → Works on Sorted Array → O(logn)



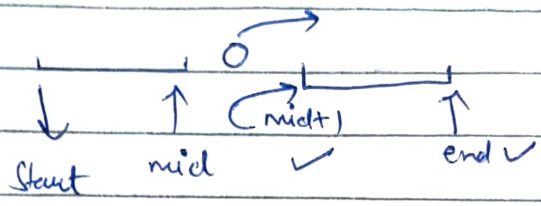
x
20 → 4
15 → 3

```
{  start=0, end= 6-1=5
while( start< = end){
        int mid= start+(end-start)/2;
        if(arr[mid] == target) return mid;
        elsif (arr[mid] > target) end = mid-1
        else   start = mid+1;
    }
  return -1;
}
```

1) input →

5	7	8	10	20	30
0	1	2	3	4	5

mid = (start + end)/2 = (0+5)/2 = 2



x	start	end	mid
8	0	5	2
10	0	5	2
	3	5	4
	3	3	3
4	0	5	2
	0	-1	0

2) arr →

10	12	15	20	30	40
0	1	2	3	4	5
		↑	↑	↑	

x	start	end	mid
50	0	5	2
	3	5	4
	5	5	5
	6	5	X

Advantages of Binary Search

input[] → 1000 (size)   if x to search is not present

Linear Search → 1000 comparisons

Binary Search → 500 comparisons
↓
250 comparisons
↓
125 comparisons
↓
62 comparisons
↓
31 comparisons
↓
15 comparisons
↓
7 comparisons
↓
3 comparisons
↓
1 → 0

$\}$ 10 comparisons

Linear Search Algorithm

arr →

1	2	5	0	9	8
0	1	2	3	4	5
↗	↗	↑	↑	↑	↑ ↑

x
~~1~~ ✓
~~5~~ ✓
15 × (-1)

#

Binary Search Algorithm

arr →

1	2	3	10	15	20
0	1	2	3	4	5

$\underbrace{\qquad\qquad}_{\times}$ $\underbrace{\qquad\qquad}_{✓}$

mid = (start + end) / 2

x	start	end	mid
10	0	5	2
	3	5	4
	3	3	3
21	0	5	2
	3	5	4
	5	5	5

①

```
                                              → int val
int   binarySearch ( int arr[], int size,){
            int start = 0, end = size - 1;
            while ( start <= end ){
                    int mid = start + (end - start)/2;
                    if (arr[mid] = = val) return mid;
                    elseif (arr[mid] < val) { start = mid+1; }
                     else { end = mid - 1; }
                }
            return -1;
          }
```

arr →

10	12	13	14	15
0	1	2	3	4

x	start	end	mid
13	0	4	2 ✓

x		start	end	mid
	15	0	4	2 ✓
		3	4	3 ✓
		4	4	4 ✓

Dt.

Pg.

Sorting Algorithms

↓

For sorting the array

**

** 1) Selection Sort **

** 2) Bubble Sort **

** 3) Insertion Sort **

**

SELECTION SORT

→ To do $\boxed{n-1}$ rounds

n
7

(1) Random Array →

7	8	1	2	5	9	6
0	1	2	3	4	5	6

1	2	5	6	7	8	9
0	1	2	3	4	5	6

Round-1 →

1	8	7	2	5	9	6
0	1	2	3	4	5	6

Round - 2 →

1	2	7	8	5	9	6
0	1	2	3	4	5	6

Round - 3 →

1	2	5	8	7	9	6
0	1	2	3	4	5	6

Round - 4 →

1	2	5	6	7	9	8
0	1	2	3	4	5	6

Round - 5 →

0	1	2	3	4	5	6
1	2	5	6	7	9	8

Round - 6 →

1	2	5	6	7	8	9
0	1	2	3	4	5	6

Dt

Pg

(2) arr[] →

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

Round 1 →

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

Round 2 →

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

Round 3 →

1	2	9	8	6	5
0	1	2	3	4	5

Round 4 →

1	2	4	5	6	8

5 Rounds

Start	end	i	minindex (val
0	5	0	2
1	5	1	2
2	5	2	4
3	5	3	5
4	5	4	6

```
void  SelectionSort(int arr[], int size){
        for(int i=0;i<size; i++){
          int current = arr[i], minindex=i;
           for(int j=i+1; j<size ; j++){
                  if (current > arr[j]){
                           current = arr[j];
                            minindex = j; } }
           Swap( arr[i], arr[minindex]);
        }
int main(){
      int arr[]={8, 5,1,4, 6, 2};
      SelectionSort (arr, 6);
      for(int i=0; i<6;i++){
              cout<<arr[i]<<" "; }
        cout<<endl;
      }
```

#

0,1

1,2

2,3

3,4

Dt ______

Pg ______

Selection Sort (Dry Run)

1 4

arr [ ] → { ~~4~~ | 1 | 9 | 6 | 2 | 3 }

0 1 2 3 4 5

Round 01 → | 1 | 4 | 9 | 6 | 2 | 3 |

0 | 1 2 3 4 5

Round 02 → | 1 | 2 | 9 | 6 | 4 | 3 |

0 1 | 2 3 4 5

Round 03 → | 1 | 2 | 3 | 6 | 4 | 9 |

0 1 2 | 3 4 5

Round 04 → | 1 | 2 | 3 | 4 | 6 | 9 |

0 1 2 3 | 4 5

Round 05 → | 1 | 2 | 3 | 4 | 6 | 9 |

0 1 2 3 4 | 5

i	j	current	minindex
~~0~~	(1 - 5)	4	~~0~~
~~1~~	(2-5)	|	~~1~~
~~2~~	(3-5)		~~2~~
~~3~~	(4-5)	|	~~3~~
4	(5)	|	5

** ** ** ** **

BUBBLE SORT

** ** ** ** **

arr[ ] → | 10 | 1 | 7 | 9 | 4 | 5 |

0 1 2 3 4 5

0,1

1,2

2,3

3,4

Round-01

10

| 1 | ~~10~~ | ~~7~~ 9 | 4 | 5 |

7 9 10

| 1 | 7 | ~~10~~ | ~~9~~ | 4 | 5 |

| 1 | 7 | 5 | 4 | 10 | 5 |

| 1 | 7 | 9 | 4 | 5 | 10 |

Round-2 → | 1 | 7 | 9 | 4 | 5 | 10 |

→ | 1 | 7 | 4 | 9 | 5 | 10 |

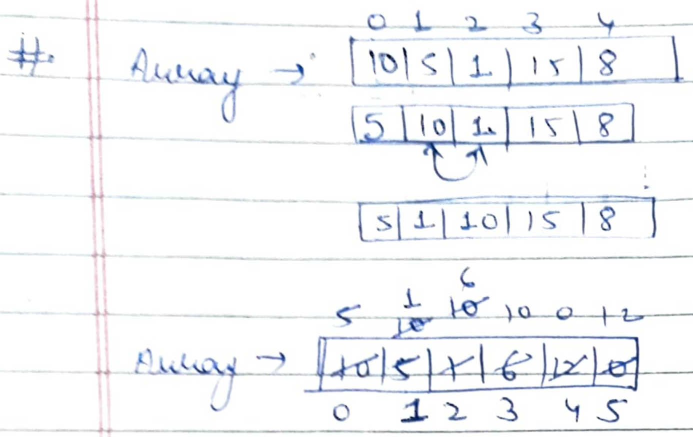
→ | 1 | 4 | 4 | 5 | 9 | 10 |

Round-3 → | 1 | 4 | 7 | 5 | 9 | 10 |

Round-4 → | 1 | 4 | 5 | 7 | 9 | 10 |

Dt________

Pg________



↗ i+1

i	j
0	1
1	2
2	3
3	4
4	5

n
6

i < n-1

j: ~~0~~ 1

i: ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5

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

int main(){
         int arr[] = { 6, 2, 3, 1, 4, 5};
         BubbleSort( arr, 6);
         for(int i=0; i<6 ;i++){ cout<< arr[i] <<" "; }
         cout << endl;
         }
```

DRY RUN - Bubble Sort Algorithm

i (Round) j (element) → (n-1-j)

i: 0, 1, 2, 3, 4, 5

Round 1

arr[] → 6|2|3|1|9|8
0 1 2 3 4 5

1
3 6 6 8 9
arr[] → 2|~~6~~|~~3~~|1|~~9~~|8

arr[] → 2|3|1|6|8|9

Round 2

~~1~~ ~~2~~ ~~1~~ 3
arr[] → 2|~~3~~|~~1~~|6|8|9

arr[] → 2|1|3|6|8|9
0 1 2 3 4 5

Round 3

0 1 2 3 4 5
arr[] → 1|2|3|6|8|9

Round 4

arr[] → 1|2|3|6|8|9
0 1 2 3 4 5

i
~~0~~
~~1~~
~~2~~
~~3~~
4

INSERTION SORT → Cardgame example

Random Array → 10|5|1|4|3|9
0 1 2 3 4 5

shifting towards right by 1 position

Round 2 → 5|10|1|4|3|5

shifting towards right by 2 position

Round 3 → 1|5|10|4|3|9

Round 4 → 1|4|5|10|3|9

Round 5 → 1|3|4|5|10|9

→ 1|3|4|5|9|10

Insertion Sort Algorithm

array →

10	5	1	8	9	4
0	1	2	3	4	5

→

5	10	1	8	9	4

→

1	5	10	8	9	4

→

1	5	8	10	9	4

	j , j--
i=1	i-1 — 0
i=2	1 — 0
i=3	2 — 0
i=4	3 — 0
i=5	4 — 0

Example →

10

10	5	3	15	2
0	1 i	2	3	4

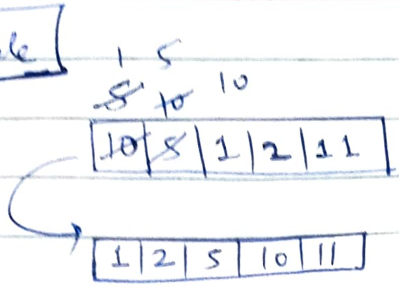
n=5

```
if (current < arr[j])
        arr[j+1] = arr[j]
    arr[j+1] = current;
```

i	j (i-1 → 0)	current
1	~~0~~ -1	5
2	~~1~~ 0	3
3	2	15
4	~~3~~ ~~2~~ ~~1~~ 0	4

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

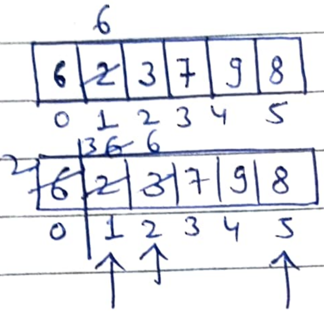
Example



, n=5

current	i	j
5	1	~~0~~ -1
1	2	~~1~~ ~~0~~ -1

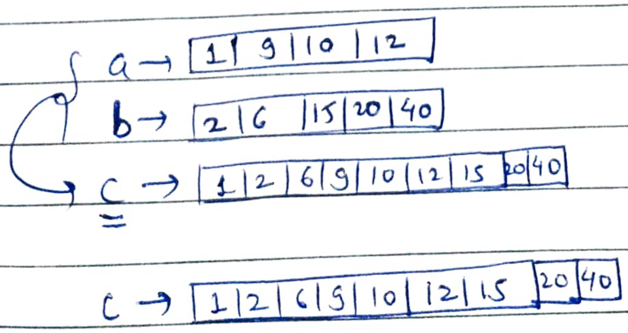
Example

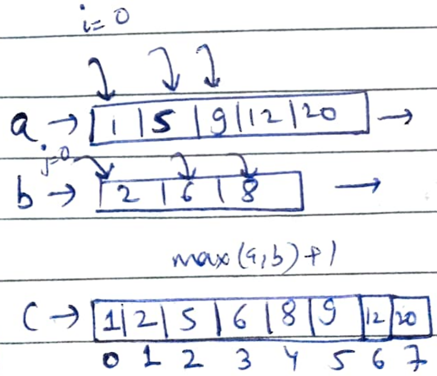


i	j	curr
1	0	6
2	1-0	2
3	2-0	
4	3-0	
5	4-0	

i	j	curr
1	0	~~2~~ 3

Merge Two Sorted Arrays


a → 1 | 9 | 10 | 12
b → 2 | 6 | 15 | 20 | 40
c → 1 | 2 | 6 | 9 | 10 | 12 | 15 | 20 | 40
c → 1 | 2 | 6 | 9 | 10 | 12 | 15 | 20 | 40



Example

```
            0 1 2 3 4
arr1 →  [1|5|9|12|20] , size1 = 5
arr2 →  [2|6|8] ,       size2 = 3
         0 1 2
output → [1|2|5|6|8|9|12|20]     int i=0,j=0;
          0 1 2 3 4 5 6  7
int output = new int[size1+size2]; int k=0;
while(i<size1 && j<size2){
        if(arr1[i]< arr2[j]){
                output[k]=arr[i];
                i++;
                k++; }

        else {
                output[k] = arr[j];
                   k++;
                   j++;
        }

}

while(i<size1){
        output[k++] = arr1[i++]; }
while(j<size2){
        output[k++] = arr2[j++]; }
for(int i=0; i<p; i++)
        cout << output[i]<<" ";
```

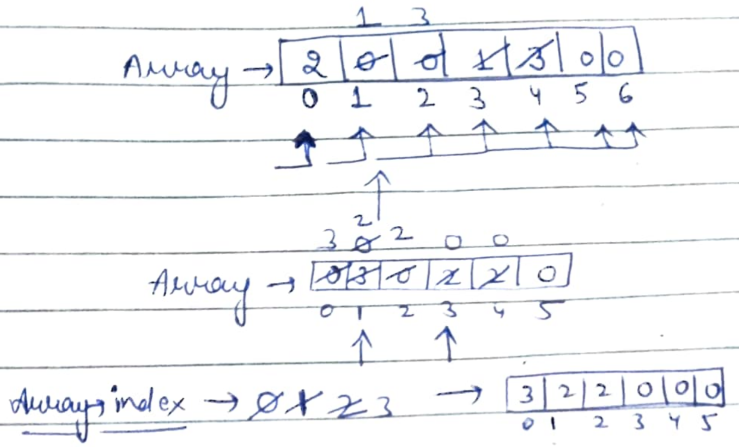
Binary Search

→ Faster & easier to implement

→ No extra space

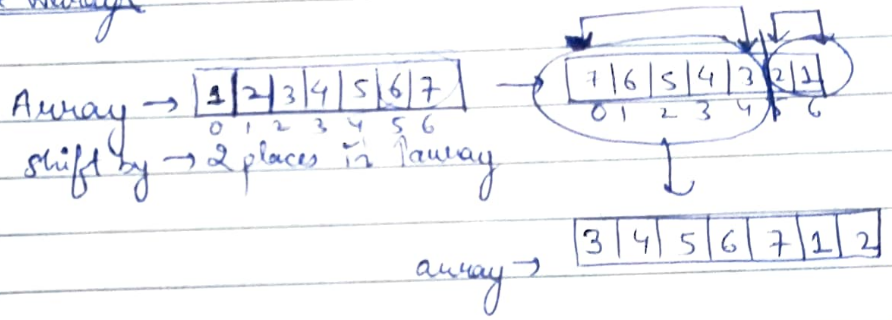
Reduces time complexity to a greater extent

Problem - Pushes zeros to the end

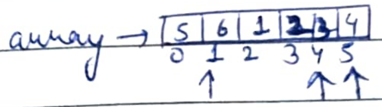


```
{
  int index=0;
  for(int i= 0; i<n; i++){
        if (arr[i]!=0){
          swap(arr[i],
                 arr[index]);
          index++;
        }
  }
```

Problem - Rotate Array



Problem - Check Array Rotation



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

Problem- Sort 012

Array →

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

output →

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

Problem - Sum of Two Arrays

1

arr1[] →

6	2	4
0	1	2

arr2[] →

7	5	6
0	1	2

arr[] → 13 8 0

For example →

0 1 2
arr1[] = {6, 2, 4}
arr2[] = {7, 5, 6}
0 1 2

arr[] = {

1	3	8	0
3	2	1	0

}

i = 2
j = 2
K = max(2,2)+1-1 =
carry = 0

END OF MODULE