Searching

Linear Search

10	15	45	20	25	6	1	100	65	99
0	1	2	3	4	5	6	7	8	9

item	25

position	-1

Input (*Declarations and Initializations***):** int arr[10], int item, int position = -1.

Process:

1. Compare the value of ***item*** with the ***element*** in the ***index-value*** 0 of the array.
2. If, they are equal, the value of ***position*** will be the value of the ***index*** and exit. Else, go to next index.
3. Repeat (1) and (2) for all the indexes.

Output:

1. Check the value of position.
   If, it is -1, Print ***item*** not found in the array.
   Else, Print ***item*** found at ***position***.

Binary Search

10	15	45	20	25	6	1	100	65	99
0	1	2	3	4	5	6	7	8	9

item	25

position	-1

Input (*Declarations and Initializations***):** int arr[10], int item, int position = -1.

Process:

1. Start with f_index = 0 and l_index = size-1
2. The value of m_index will be (f_index+l_index)/2.
3. Compare the value of item with arr[m_index].
   (a) If item < arr[m_index], l_index will be m_index-1.
   (b) Else if item > arr[m_index], f_index will be m_index+1.
   (c) Else, position will be m_index. Exit.
4. Repeat (2), (3) till f_index<= m_index.

Output:

1. Check the value of position.
   If, it is -1, Print ***item*** not found in the array.
   Else, Print ***item*** found at ***position***.

Sorting

Selection Sort:

12	56	30	21	71	25	9
0	1	2	3	4	5	6

9	56	30	21	71	25	12
0	1	2	3	4	5	6

9	12	30	21	71	25	56
0	1	2	3	4	5	6

9	12	21	30	71	25	56
0	1	2	3	4	5	6

9	12	21	25	71	30	56
0	1	2	3	4	5	6

9	12	21	25	30	71	56
0	1	2	3	4	5	6

9	12	21	25	30	56	71
0	1	2	3	4	5	6

Initializations and Inputs: int soa, int arr[ ].
Process:

1. Value of Starting index (***starting_index***) will be 0.
2. We will consider the starting index as the index (***mini_index***) containing the minimum element.
3. Value of Current index (***current_index***) will be starting_index + 1.
4. If, the element in current_index is less than the element in mini_index, the value of mini_index will be current_index.
5. Increase the value of current_index and repeat (4) for all the indexes.
6. Swap the elements in start_index and mini_index.
7. Increase the value of start_index and repeat (2) (3) (4) (5) (6) till start_index < size-1.

Output: The arr[ ] array.

starting_index	0						
mini_index	0	0	0	0	0	0	6
current_index	1	2	3	4	5	6	

mini_element	12	12	12	12	12	12
current_element	56	30	21	71	25	9

starting_index	1					
mini_index	1	2	3	3	3	6
current_index	2	3	4	5	6	

mini_element	56	30	21	21	21
current_element	30	21	71	25	12

starting_index	2				
mini_index	2	3	3	3	3
current_index	3	4	5	6	

mini_element	30	21	21	21
current_element	21	71	25	56

starting_index	3			
mini_index	3	3	5	5
current_index	4	5	6	

mini_element	30	30	25
current_element	71	25	56

starting_index	4		
mini_index	4	5	5
current_index	5	6	

mini_element	71	30
current_element	30	56

starting_index	5	
mini_index	5	6
current_index	6	

mini_element	71
current_element	56

Insertion Sort:

12	56	30	21	71	25	9
0	1	2	3	4	5	6

12	56	30	21	71	25	9
0	1	2	3	4	5	6

12	30	56	21	71	25	9
0	1	2	3	4	5	6

12	21	30	56	71	25	9
0	1	2	3	4	5	6

12	21	30	56	71	25	9
0	1	2	3	4	5	6

12	21	25	30	56	71	9
0	1	2	3	4	5	6

9	12	21	25	30	56	71
0	1	2	3	4	5	6

Initializations and Inputs: int soa, int arr[ ].
Process:

1. Value of Staring index (***starting_index***) will be 1.
2. We will consider the element in starting_index as the ***element_on_hand***.
3. Value of Current index (***current_index***) will be starting_index – 1.
4. If, current_index >= 0 and the element in current_index is greater than element_on_hand, do (a) (b) (5), else go to (6).
    a. The element of current_index+1 index will be the element in current_index.
    b. Decrease the value of current_index by 1.
5. Repeat (4).
6. The element in current_index+1 index will be the element_on_hand.
7. Increase the value of starting_index and repeat (2) (3) (4) (5) (6) till starting_index<size.

Output: The arr[ ] array.

element_on_hand
30

starting_index	1
current_index	0
current_element	12

starting_index	2	2
current_index	1	0
current_element	56	12

starting_index	3	3	3
current_index	2	1	0
current_element	56	30	12

starting_index	4
current_index	3
current_element	56

starting_index	5	5	5	5
current_index	4	3	2	1
current_element	71	56	30	21

starting_index	6	6	6	6	6	6	6
current_index	5	4	3	2	1	0	-1
current_element	71	56	30	25	21	12	

Bubble Sort:

12	56	30	21	71	25	9
0	1	2	3	4	5	6

1st Phase:

12	56	30	21	71	25	9
0	1	2	3	4	5	6

12	56	30	21	71	25	9
0	1	2	3	4	5	6

12	30	56	21	71	25	9
0	1	2	3	4	5	6

12	30	21	56	71	25	9
0	1	2	3	4	5	6

12	30	21	56	71	25	9
0	1	2	3	4	5	6

12	30	21	56	25	71	9
0	1	2	3	4	5	6

12	30	21	56	25	9	71
0	1	2	3	4	5	6

2nd Phase:

12	30	21	56	25	9	71
0	1	2	3	4	5	6

12	30	21	56	25	9	71
0	1	2	3	4	5	6

12	21	30	56	25	9	71
0	1	2	3	4	5	6

12	21	30	56	25	9	71
0	1	2	3	4	5	6

12	21	30	25	56	9	71
0	1	2	3	4	5	6

12	21	30	25	9	56	71
0	1	2	3	4	5	6

3rd Phase:

12	21	30	25	9	56	71
0	1	2	3	4	5	6

12	21	30	25	9	56	71
0	1	2	3	4	5	6

12	21	30	25	9	56	71
0	1	2	3	4	5	6

12	21	25	30	9	56	71
0	1	2	3	4	5	6

12	21	25	9	30	56	71
0	1	2	3	4	5	6

4th Phase:

12	21	25	9	30	56	71
0	1	2	3	4	5	6

12	21	25	9	30	56	71
0	1	2	3	4	5	6

12	21	25	9	30	56	71
0	1	2	3	4	5	6

12	21	9	25	30	56	71
0	1	2	3	4	5	6

<u>5th Phase:</u>

12	21	9	25	30	56	71
0	1	2	3	4	5	6

12	21	9	25	30	56	71
0	1	2	3	4	5	6

12	9	21	25	30	56	71
0	1	2	3	4	5	6

<u>6th Phase:</u>

12	9	21	25	30	56	71
0	1	2	3	4	5	6

9	12	21	25	30	56	71
0	1	2	3	4	5	6

Input and Initializations: int soa, int arr[ ]

Process:

1. Value of Staring index (***starting_index***) will be 0.
2. Value of Current index (***current_index***) will be 0.
3. If current_index < (size-1) – starting_index, go to (a) (b) (4), else go to (5).
    a. If, the element in current_index is greater than the element in current_index+1, swap the elements.
    b. Increase the value of current_index.
4. Repeat (3).
5. Increase the value of starting_index and repeat (2) (3) (4) till starting_index<size-1.

Ouput: The arr[ ] array.

starting_index	0						
current_index	0	1	2	3	4	5	6
condition (3)	0<6	1<6	2<6	3<6	4<6	5<6	6<6
current_element	12	56	56	56	71	71	
current_P1_element	56	30	21	71	25	9	
condition (3a)	12>56	56>30	56>21	56>71	71>25	71>9	

starting_index	1					
current_index	0	1	2	3	4	5
condition (3)	0<5	1<5	2<5	3<5	4<5	5<5
current_element	12	30	30	56	56	
current_P1_element	30	21	56	25	9	
condition (3a)	12>30	30>21	30>56	56>25	56>9	

starting_index	2				
current_index	0	1	2	3	4
condition (3)	0<4	1<4	2<4	3<4	4<4
current_element	12	21	30	30	
current_P1_element	21	30	25	9	
condition (3a)	12>21	21>30	30>25	30>9	

starting_index	3			
current_index	0	1	2	3
condition (3)	0<3	1<3	2<3	3<3
current_element	12	21	25	
current_P1_element	21	25	9	
condition (3a)	12>21	21>25	25>9	

starting_index	4		
current_index	0	1	2
condition (3)	0<2	1<2	2<2
current_element	12	21	
current_P1_element	21	9	
condition (3a)	12>21	21>9	

starting_index	5	
current_index	0	1
condition (3)	0<1	1<1
current_element	12	
current_P1_element	9	
condition (3a)	12>9	