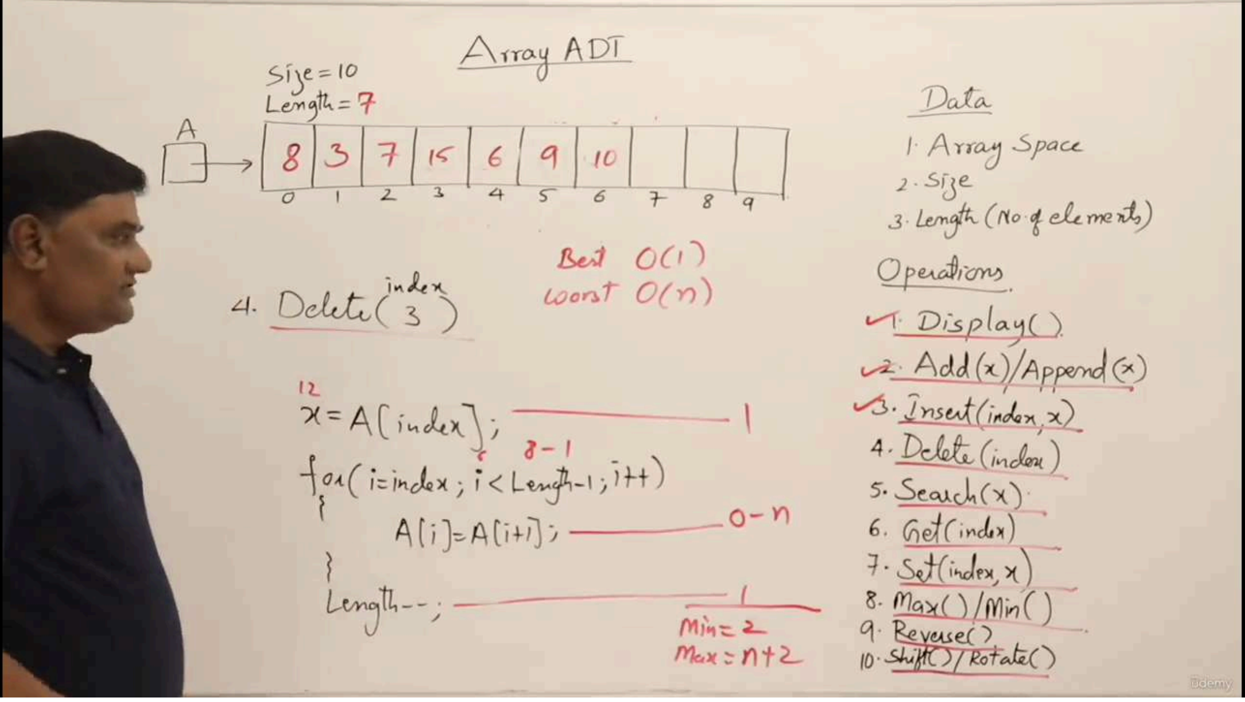


Array ADT

Size=10
Length=7

A

8	3	7	15	6	9	10			
0	1	2	3	4	5	6	7	8	9

Best O(1)
worst O(n)

4. Delete($\overset{index}{3}$)

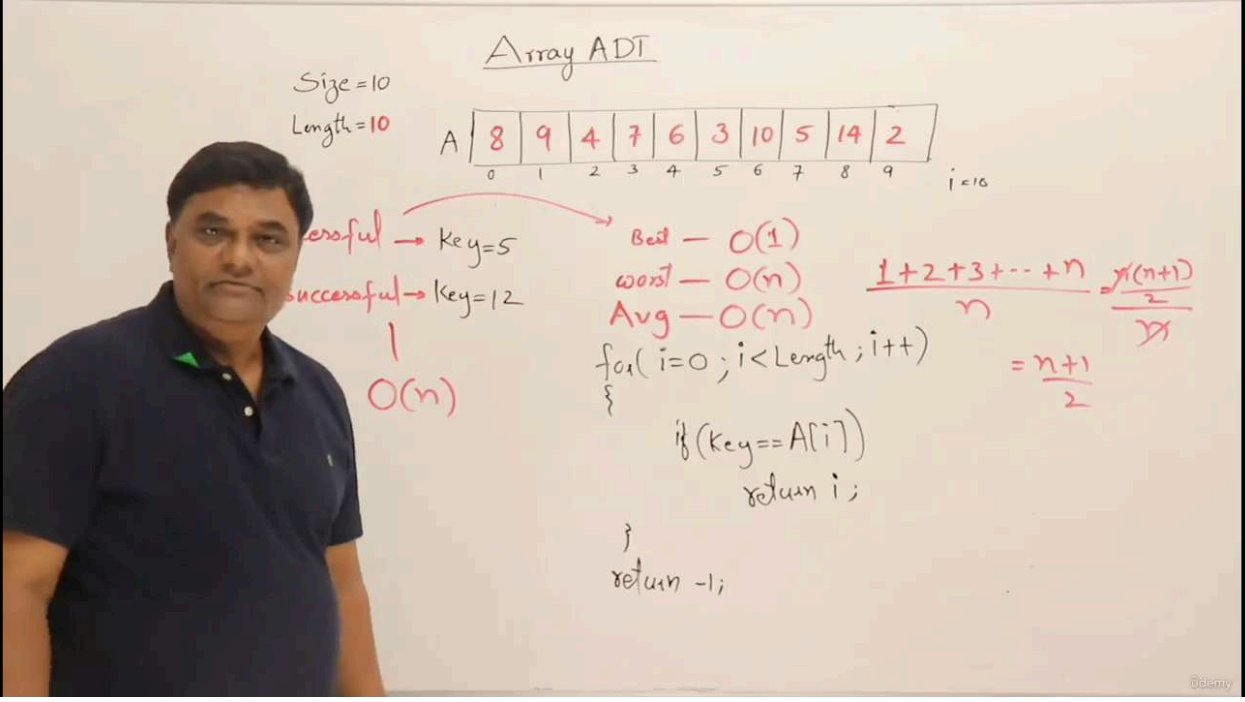
```
12
x=A[index];                       ——— 1
          6    8-1
for(i=index; i<Length-1; i++)
{
      A[i]=A[i+1];                ——— 0-n
}
Length--;                         ——— 1
                                  Min=2
                                  Max=n+2
```

Data

1. Array Space
2. Size
3. Length (No of elements)

Operations

✓1. Display()
✓2. Add(x)/Append(x)
✓3. Insert(index,x)
4. Delete(index)
5. Search(x)
6. Get(index)
7. Set(index,x)
8. Max()/Min()
9. Reverse()
10. Shift()/Rotate()



Array ADT

Size = 10

Length = 10

A									
8	9	4	7	6	3	10	5	14	2
0	1	2	3	4	5	6	7	8	9

i = 10

cessful → Key = 5

successful → Key = 12

O(n)

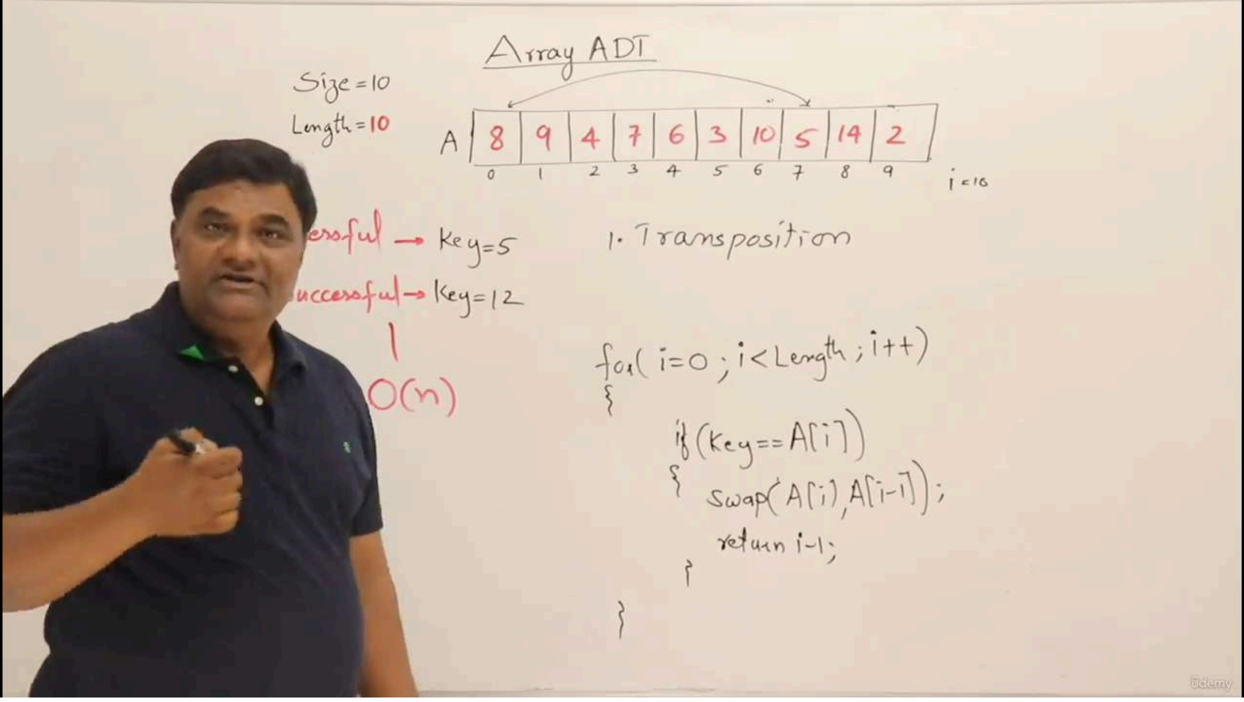
Best — O(1)

worst — O(n)

Avg — O(n)

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

```
for( i=0 ; i<Length ; i++)
{
        if (Key==A[i])
              return i;
  }
return -1;
```



Array ADT

Size=10

Length=10

A	8	9	4	7	6	3	10	5	14	2
	0	1	2	3	4	5	6	7	8	9

i=10

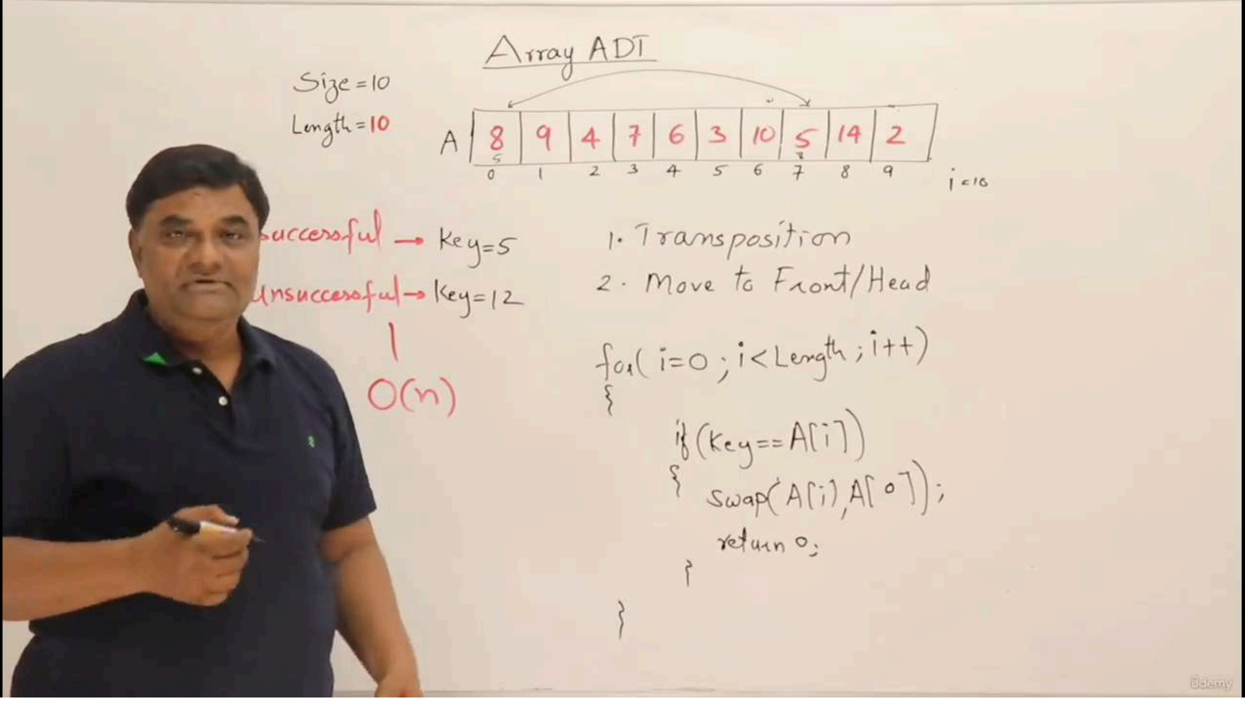
essful → Key=5

uccessful → Key=12

O(n)

1. Transposition

```
for( i=0 ; i<Length ; i++)
{
        if(Key==A[i])
        {
          swap(A[i],A[i-1]);
          return i-1;
        }
}
```



Array ADT

Size = 10

Length = 10

A									
8	9	4	7	6	3	10	5	14	2
0	1	2	3	4	5	6	7	8	9

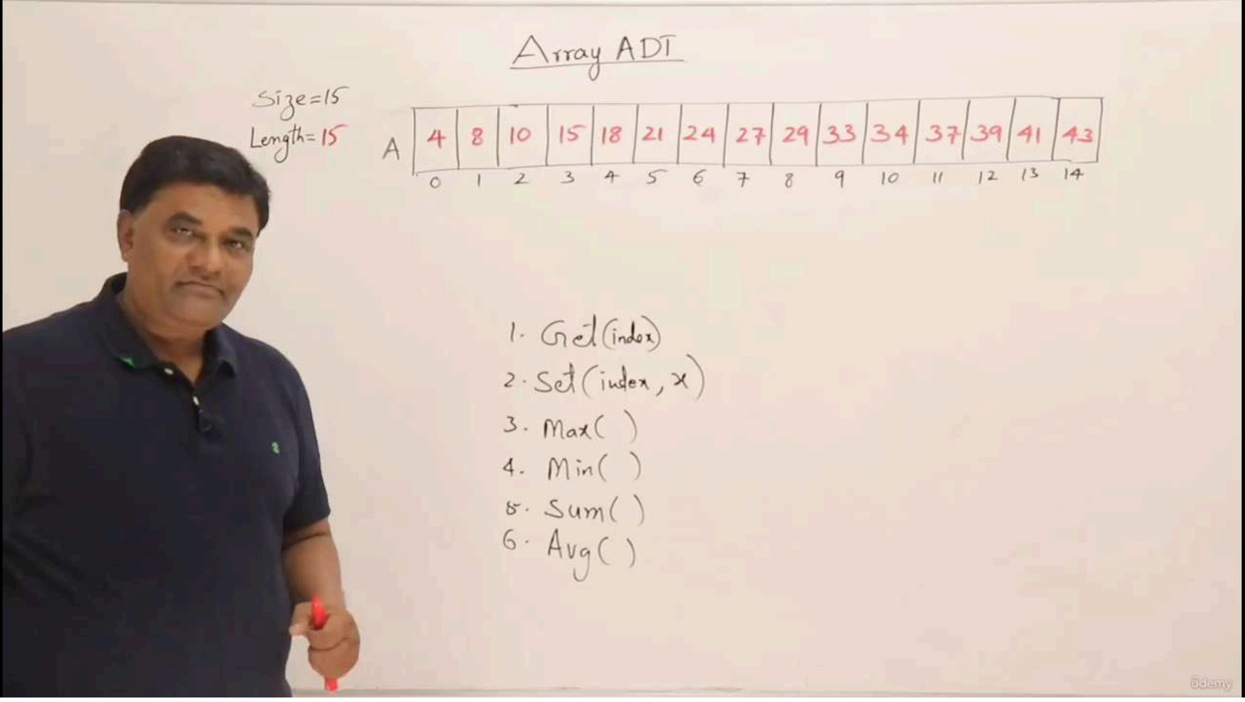
successful → Key=5

Unsuccessful → Key=12

O(n)

1. Transposition
2. Move to Front/Head

```
for( i=0 ; i<Length ; i++)
{
        if(Key==A[i])
        {  swap(A[i],A[0]);
           return 0;
        }
}
```



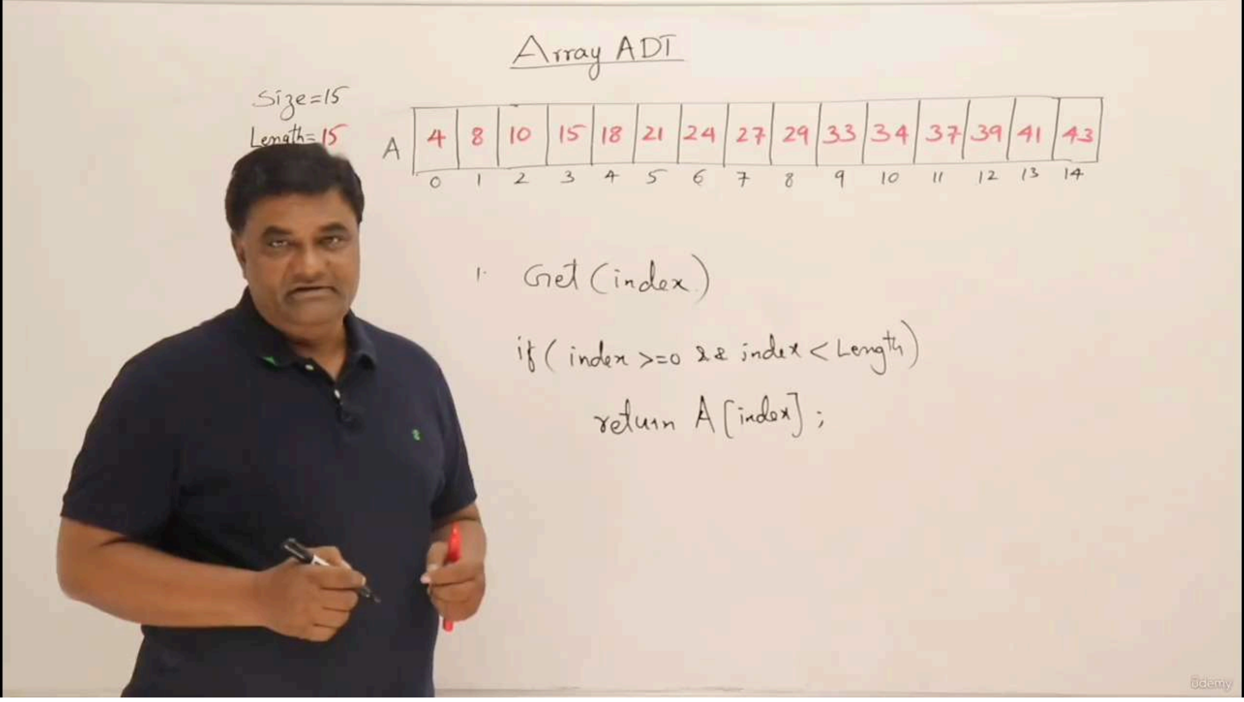
Array ADT

Size=15

Length=15

A															
	4	8	10	15	18	21	24	27	29	33	34	37	39	41	43
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

1. Get(index)
2. Set(index, x)
3. Max( )
4. Min( )
5. Sum( )
6. Avg( )



Array ADT

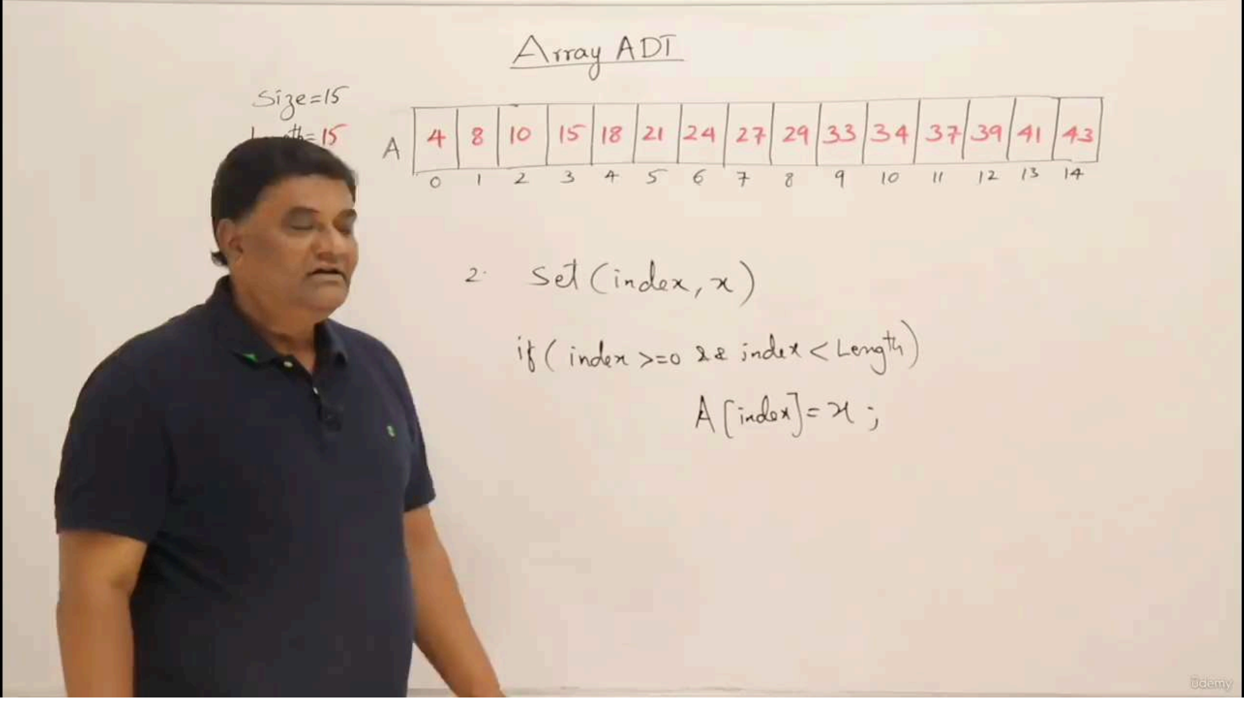
Size=15

Length=15

A	4	8	10	15	18	21	24	27	29	33	34	37	39	41	43
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

1. Get(index)

```
if ( index >=0 && index < Length)
        return A[index] ;
```



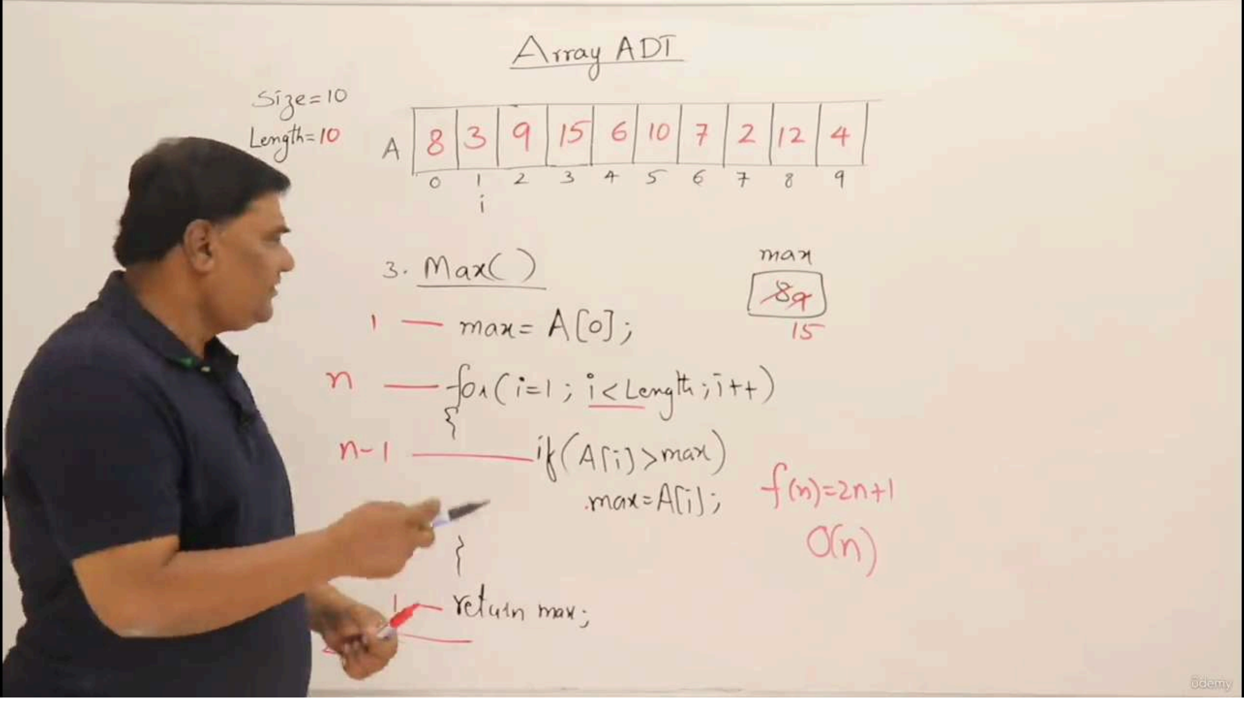
Array ADT

Size=15
Length=15

A

4	8	10	15	18	21	24	27	29	33	34	37	39	41	43
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

```
2. Set(index, x)
     if (index >=0 && index < Length)
          A[index]=x;
```



Array ADT

Size = 10

Length = 10

A	8	3	9	15	6	10	7	2	12	4
	0	1	2	3	4	5	6	7	8	9
		i								

3. Max( )

```
1   — max = A[0];
n   — for(i=1; i<Length; i++)
      {
n-1 —     if(A[i]>max)
              max=A[i];
      }
1   — return max;
```

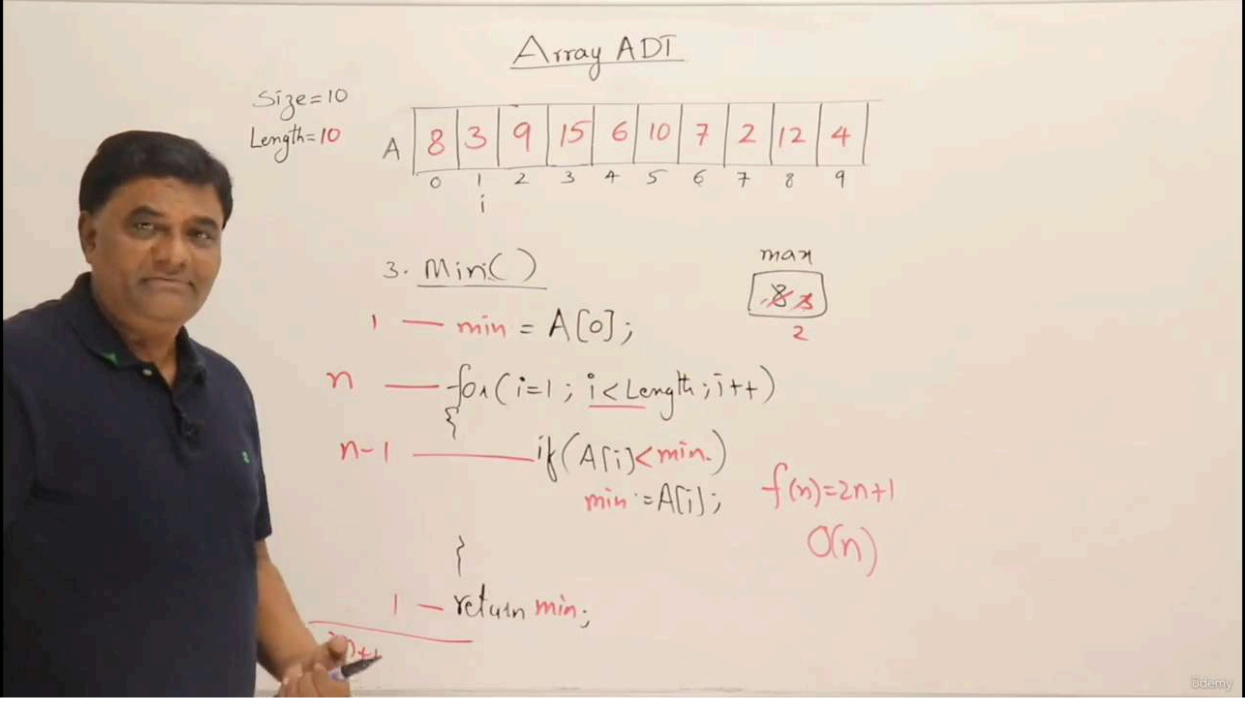
max

$\boxed{\cancel{8}\cancel{9}}$

15

$f(n)=2n+1$

$O(n)$



Array ADT

Size=10
Length=10

A	8	3	9	15	6	10	7	2	12	4
	0	1	2	3	4	5	6	7	8	9
		i								

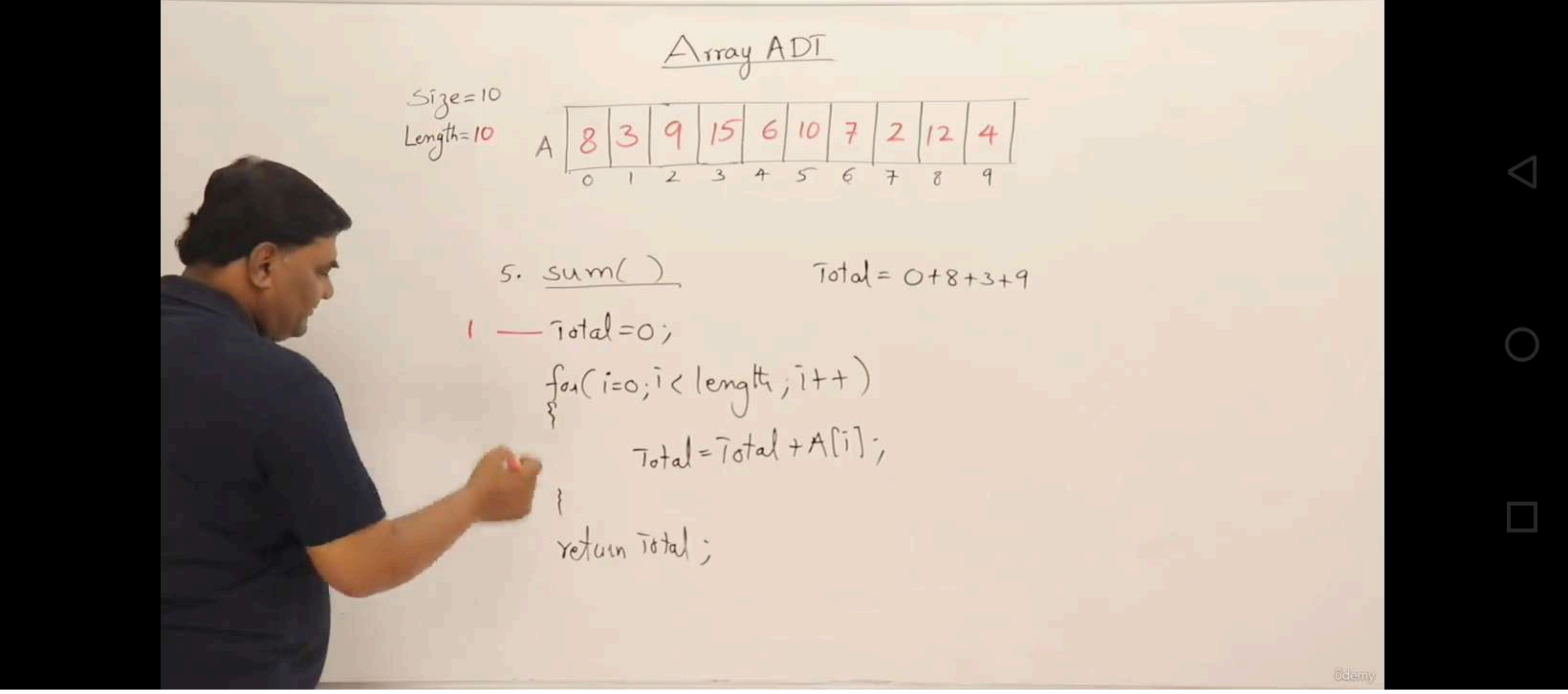
3. Min( )

```
1   — min = A[0];
n   — for(i=1; i<Length; i++)
      {
n-1 —     if(A[i]<min)
              min = A[i];
      }
1   — return min;
n+1
```

max
$\boxed{\cancel{8}\ \cancel{3}}$
2

$f(n)=2n+1$

$O(n)$



Array ADT

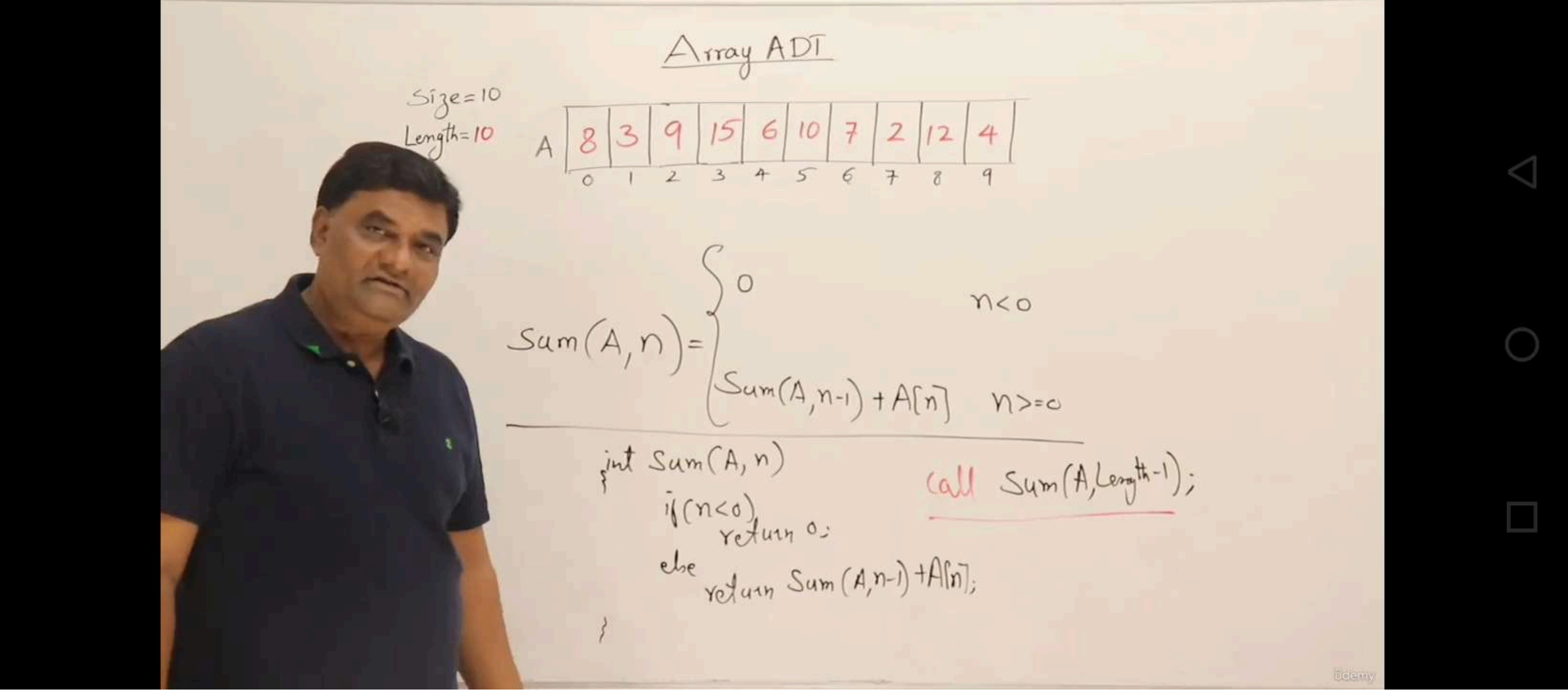
Size=10
Length=10

A	8	3	9	15	6	10	7	2	12	4
	0	1	2	3	4	5	6	7	8	9

5. sum( )

Total = 0+8+3+9

```
1 —— Total=0;
     for(i=0;i<length;i++)
     {
          Total=Total+A[i];
     }
     return Total;
```



Array ADT

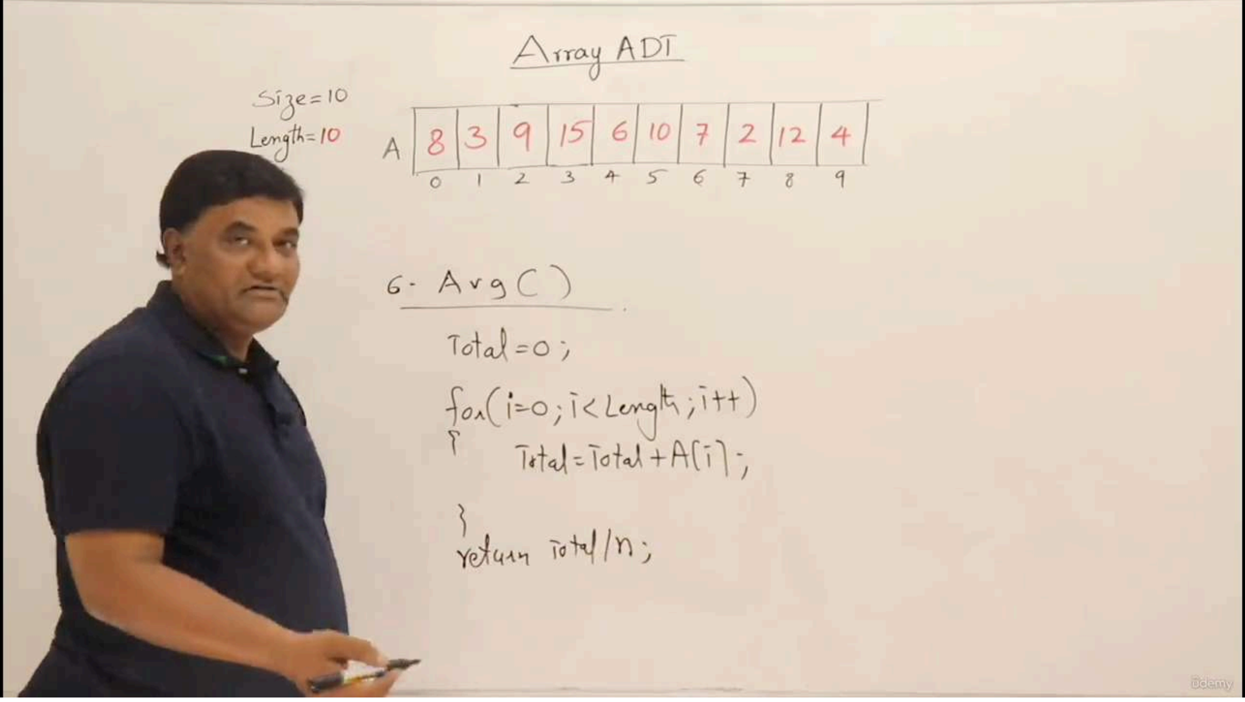
Size = 10
Length = 10

A	8	3	9	15	6	10	7	2	12	4
	0	1	2	3	4	5	6	7	8	9

$$Sum(A,n) = \begin{cases} 0 & n<0 \\ Sum(A,n-1) + A[n] & n>=0 \end{cases}$$

```
int Sum(A, n)
{
      if(n<0)
          return 0;
      else
          return Sum (A,n-1) +A[n];
}
```

call Sum(A,Length-1);



Array ADT

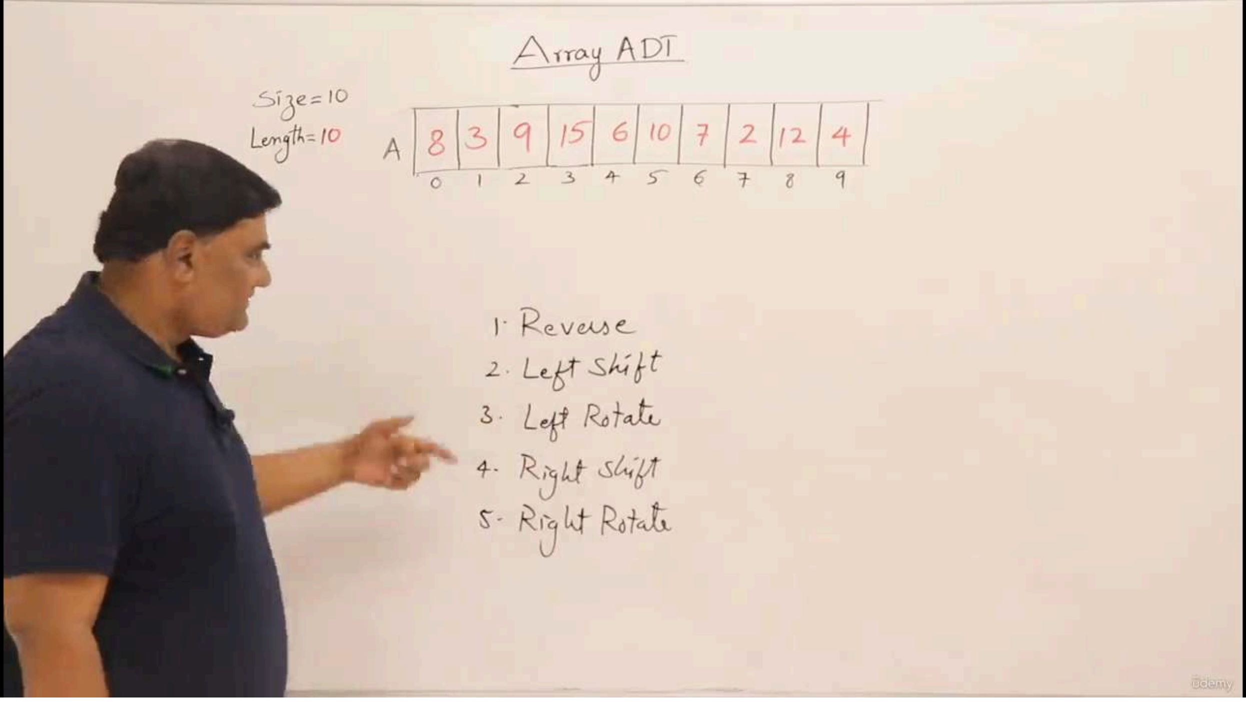
Size=10

Length=10

A	8	3	9	15	6	10	7	2	12	4
	0	1	2	3	4	5	6	7	8	9

6. Avg( )

```
Total=0;
for(i=0;i<Length;i++)
{
        Total=Total+A[i];
}
return Total/n;
```



Array ADT

Size=10
Length=10

A	8	3	9	15	6	10	7	2	12	4
	0	1	2	3	4	5	6	7	8	9

1. Reverse
2. Left Shift
3. Left Rotate
4. Right shift
5. Right Rotate

Deleting from Array

```c
#include<stdio.h>
struct Array
{
    int A[10];
    int size;
    int length;
};
    void Display(struct Array arr)
    {
        int i;
        printf("\nElements are\n");
        for(i=0;i<arr.length;i++)
            printf("%d ",arr.A[i]);
    }

int Delete(struct Array *arr,int index)
{
    int x=0;
    int i;
    if(index>=0 && index<arr->length)
    {
        x=arr->A[index];
        for(i=index;i<arr->length-1;i++)
            arr->A[i]=arr->A[i+1];
        arr->length--;
        return x;
    }
    return 0;
}

int main()
{

    struct Array arr1={{2,3,4,5,6},10,5};
    printf("%d",Delete(&arr1,0));
    Display(arr1);
    return 0;
}
```

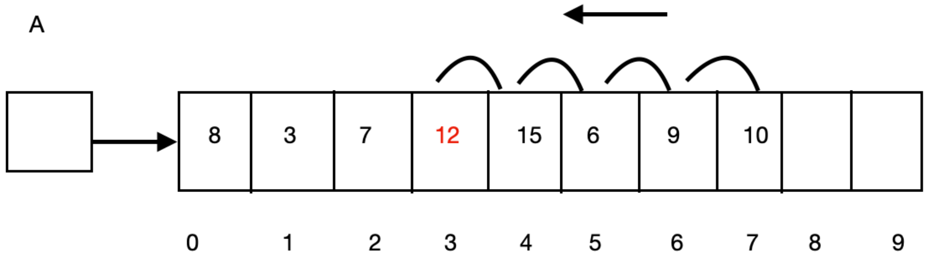
Deleting from Array

- Removing an element from an array is called deleting
- After deleting an element the space must not be empty in an array so shift the bits accordingly
- The index should not be beyond the array

Syntax : Delete(3)

```
x = A[ index ]
for(  i = index ; i < length - 1 ; i++ )
{
        A[ i ] = A[ i+1]  ;
}
```

Size = 10
Length = 8



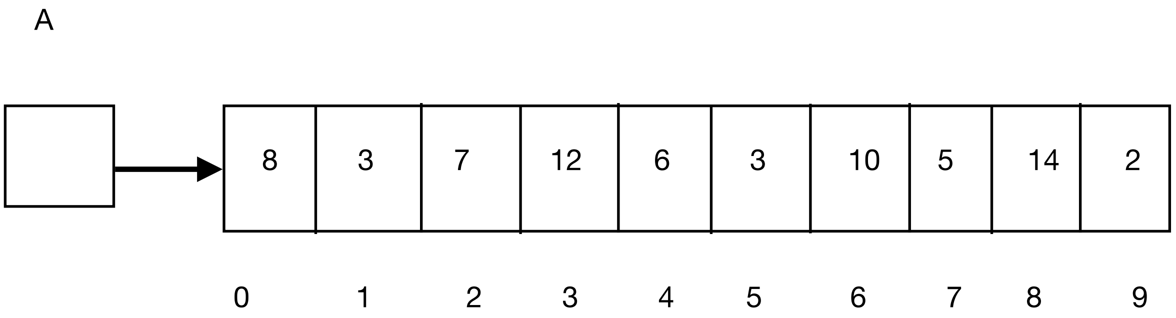
Linear Search

- They are 2 search method in an array

I. Linear search
II. Binary search

- Linear search :

Size = 10
Length = 10



8	3	7	12	6	3	10	5	14	2
0	1	2	3	4	5	6	7	8	9

Key = 5 (successful search)
Key = 12 (unsuccessful search)

- All the elements must be unique here

- The value you are searching is called key, In linear search we search the key element one by one linearly

- We search the element by comparing it with the key value

- The result of the search is the location of the element where its present (index number) , it is very useful in accessing the element in the list

- If the element is not found throughout the list that means it is not present in the list therefore search is unsuccessful

Syntax :

```
for( i = 0;  i < length ; i++ )
{
        if( key == A[ i ] )
        return i;                    //if search is successful it ends here
}

return -1;                           // if search unsuccessful returns -1
```

Searching in a Array

```c
#include<stdio.h>
struct Array
{
    int A[10];
    int size;
    int length;
};
    void Display(struct Array arr)
    {
        int i;
        printf("\nElements are\n");
        for(i=0;i<arr.length;i++)
            printf("%d ",arr.A[i]);
    }
 void swap(int *x,int *y)
 {
     int temp=*x;
     *x=*y;
     *y=temp;
 }

int LinearSearch(struct Array *arr,int key)
{
    int i;
    for(i=0;i<arr->length;i++)
    {
        if(key==arr->A[i])
        {
            swap(&arr->A[i],&arr->A[0]);
            return i;
        }
    }
    return -1;
}


int main()
{

    struct Array arr1={{2,23,14,5,6,9,8,12},10,8};
    printf("%d",LinearSearch(&arr1,14));
    Display(arr1);
    return 0;
}
```

Improving Linear Search

- When you are searching for a key element there is a possibility that you are searching the same element again

- To improve the speed of comparison , you can move a key element repeatedly search one step forward this method is called transposition

Syntax :

```
for( i = 0; i < length ; i++ )
        {
                if( key == A[ i ] )
                {
                        swap( A[i], A[i-1]);
                        return i-1;
                }
        }
```

- The second method is you can directly swap the key element to the first element this process is called move to head . The next search for the same element becomes faster.

```
for( i = 0; i < length ; i++ )
        {
                if( key == A[ i ] )
                {
                        swap( A[i], A[0]);
                        return 0;
                }
        }
```

Get Set Max Min on Array

```
#include<stdio.h>
struct Array
{
    int A[10];
    int size;
    int length;
};
    void Display(struct Array arr)
    {
        int i;
        printf("\nElements are\n");
        for(i=0;i<arr.length;i++)
            printf("%d ",arr.A[i]);
    }
 void swap(int *x,int *y)
 {
     int temp=*x;
     *x=*y;
     *y=temp;
 }
int Get(struct Array arr,int index)
{
    if(index>=0 && index<arr.length)
        return arr.A[index];
    return -1;
}

void Set(struct Array *arr,int index,int x)
{
    if(index>=0 && index<arr->length)
    arr->A[index]=x;
}

int Max(struct Array arr)
{
    int max=arr.A[0];
    int i;
    for(i=1;i<arr.length;i++)
    {
        if(arr.A[i]>max)
            max=arr.A[i];
    }
    return max;
}
```

```
int Min(struct Array arr)
{
    int min=arr.A[0];
    int i;
    for(i=1;i<arr.length;i++)
    {
        if(arr.A[i]<min)
            min=arr.A[i];
    }
    return min;
}

int Sum(struct Array arr)
{
    int s=0;
    int i;
    for(i=0;i<arr.length;i++)
        s+=arr.A[i];
    return s;
}
float Avg(struct Array arr)
{
    return (float)Sum(arr)/arr.length;
}
int main()
{

    struct Array arr1={{2,3,9,16,18,21,28,32,35},10,9};
    printf("%d",Sum(arr1));
    Display(arr1);
    return 0;
}
```