ii. Bubble Sort

- Bubble sort is a sorting algorithm that compares two adjacent elements and swaps them until they are in the intended order.

In this algorithm,

- Traverse from left and compare adjacent elements and the higher one is placed at right side.
- In this way, the largest element is moved to the rightmost end at first.
- This process is then continued to find the second largest and place it and so on until the data is sorted.

Algorithm:

```
for all elements of list
  if list[i] > list[i+1]
    swap(list[i], list[i+1])
  end if
end for
return list
```

iii. Insertion Sort

- Insertion sort is a sorting algorithm that places an unsorted element at its suitable place in each iteration.

Algorithm:

Step 1 - If the element is the first element, assume that it is already sorted. Return 1.

Step 2 - Pick the next element, and store it separately in a key.

Step 3 - Now, compare the key with all elements in the sorted array.

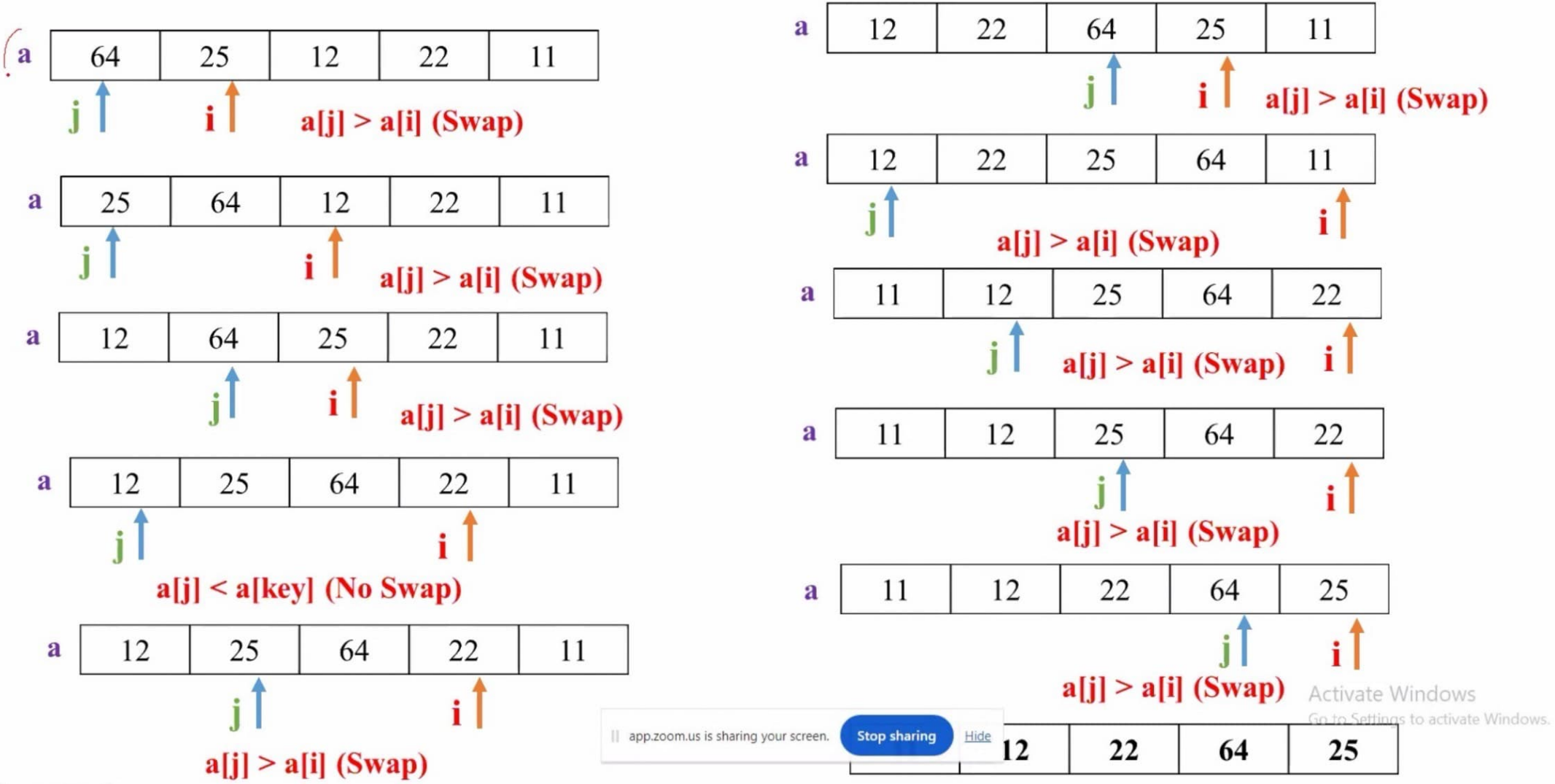
Step 4 - If the element in the sorted array is smaller than the current element, then move to the next element.

Else, shift greater elements in the array towards the right.

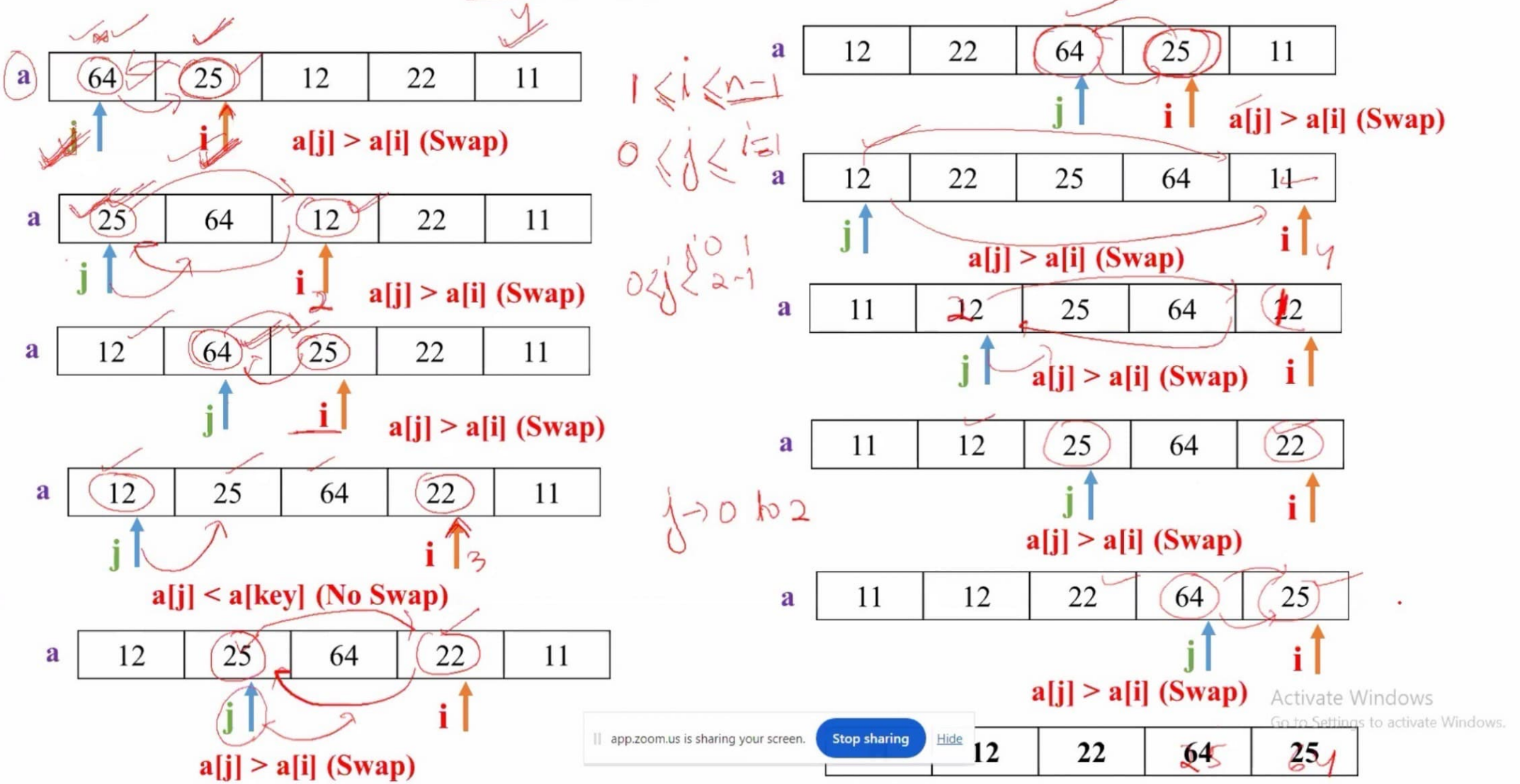
Step 5 - Insert the value.

Step 6 - Repeat until the array is sorted.

Example: Sorting of an array in ascending order using insertion sort



Example: Sorting of an array in ascending order using insertion sort

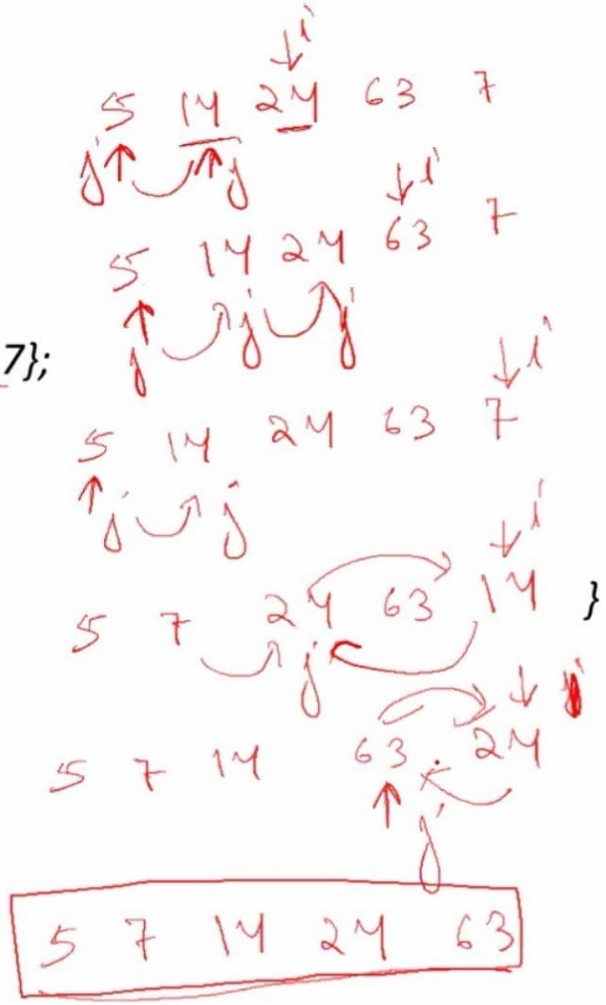


```
#include<iostream>
using namespace std;
int main()
{
   int arr[5]={5,14,24,63,7};
   int i,j,temp;
   for(i=1;i<5;i++)
   {
      for(j=0;j<=i;j++)
      {
         if(arr[j]>arr[i])
         {
            temp=arr[j];
            arr[j]=arr[i];
            arr[i]=temp;
         }
      }
   }
   for (i=0;i<5;i++)
   {
      cout<<arr[i]<<'\t';
   }
   return(0);
}
```

Output:

5 7 14 24 63

```
#include<iostream>
using namespace std;
int main()
{
   int arr[5]={5,14,24,63,7};
   int i,j,temp;
   for(i=1;i<5;i++)
   {
      for(j=0;j<i;j++)
      {
         if(arr[j]>arr[i])
         {
            temp=arr[j];
            arr[j]=arr[i];
            arr[i]=temp;
         }
      }
   }
```




```
   for (i=0;i<5;i++)
   {
      cout<<arr[i]<<'\t';
   }
   return(0);
}
```

Output:

5 7 14 24 63

3. Searching

- Array searching can be defined as the operation of finding a particular element or a group of elements in the array.
- There are several searching algorithms. The most commonly used among them are:
    i. Linear Search
    ii. Binary Search

i. Linear Search is defined as a sequential search algorithm that starts at one end and goes through each element of a list until the desired element is found, otherwise the search continues till the end of the data set.

3. Searching

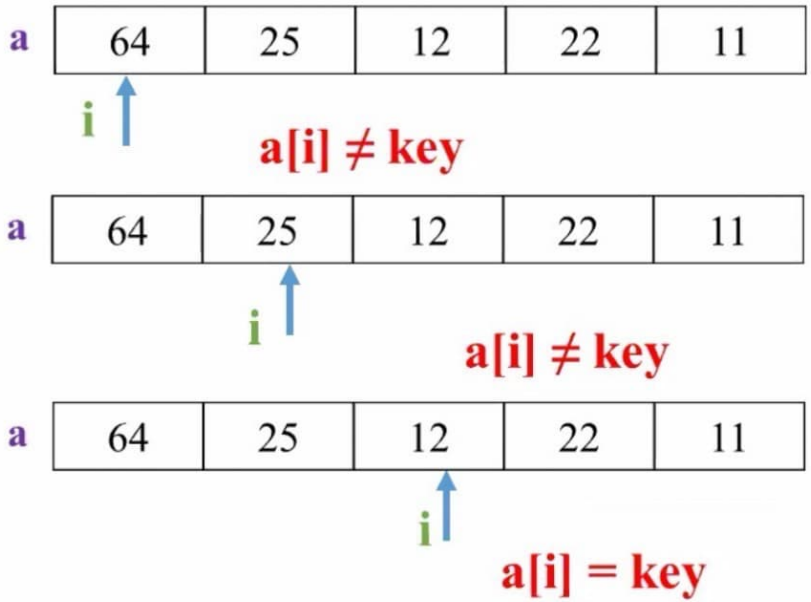
- Array searching can be defined as the operation of finding a particular element or a group of elements in the array.
- There are several searching algorithms. The most commonly used among them are:

i. Linear Search (both sorted & unsorted array)
(a.k.a Sequential Search)

ii. Binary Search (sorted array) (Bracketing Search)

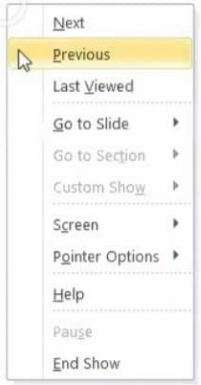
i. Linear Search is defined as a sequential search algorithm that starts at one end and goes through each element of a list until the desired element is found, otherwise the search continues till the end of the data set.

Linear Search algorithm to search 12 (key=12)



Now, the element to be searched is found. So algorithm will return the index of the element matched.

```
#include<iostream>
using namespace std;
int main()
{
   int arr[5]={5,14,24,63,7};
   int i,j,key; key=24;j=0;
   for(i=0;i<5;i++)
   {
      if(arr[i]==key)
      {
         cout<<"The index is "<<i;
         j=1;
      }
   }
   if (j==0)
   {
      cout<<"The index is not found";
   }
   return(0);
}
```



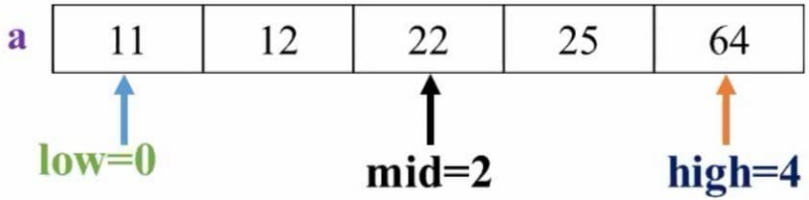
Output: *The index is 2*

```
#include<iostream>
using namespace std;
int main()
{
   int arr[5]={5,14,24,63,7};
   int i,j,key; key=24;j=0;
   for(i=0;i<5;i++)
   {
      if(arr[i]==key)
      {
         cout<<"The index is "<<i;
         j=1;
      }
   }
   if (j==0)
   {
      cout<<"The index is not found";
   }
   return(0);
}
```

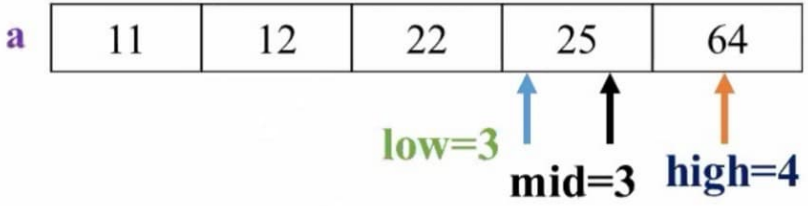
key = 10

Output: The index is 2

Binary Search algorithm to search 25 (key=25)

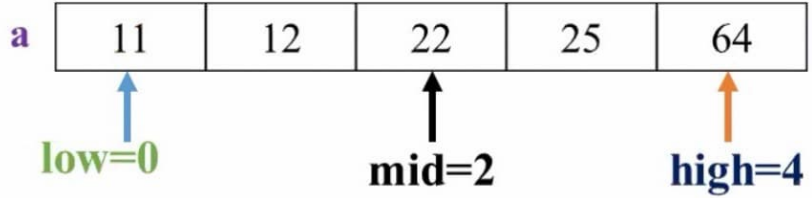


key > a[mid]
So low = mid+1;
mid = (low + high)/2

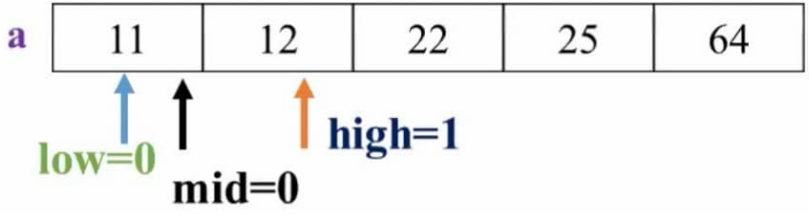


key = a[mid] = 3

Binary Search algorithm to search 11 (key = 11)



key < a[mid]
So high = mid-1;
mid=(low + high)/2



key = a[mid] = 0

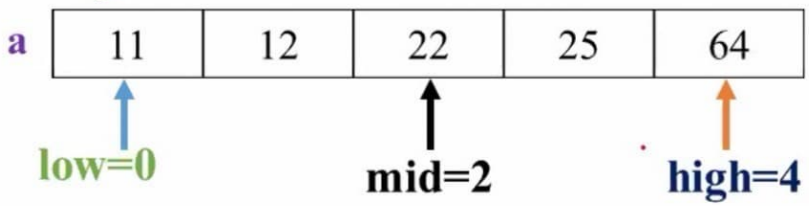
a = { 11, 22, 13, 16, 7 }

ii. Binary Search Binary Search is defined as a searching algorithm used in a **sorted array** by repeatedly dividing the search interval in half.

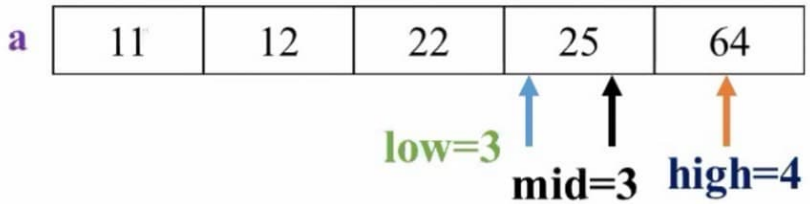
Algorithm:

- Initialize 'low' as minimum index and 'high' as maximum index number (n-1).
- Divide the search space into two halves by finding the middle index "mid".
- Compare the middle element of the search space with the "key".
- If the 'key' is found at middle element, the process is terminated.
- If the key is not found at middle element, choose which half will be used as the next search space.
  - If the key is smaller than the middle element, then the left side is used for next search i.e. high=mid-1.
  - If the key is larger than the middle element, then the right side is used for next search i.e. low=mid+1.
- This process is continued until the key is found or the total search space is exhausted.

Binary Search algorithm to search 25 (key=25)

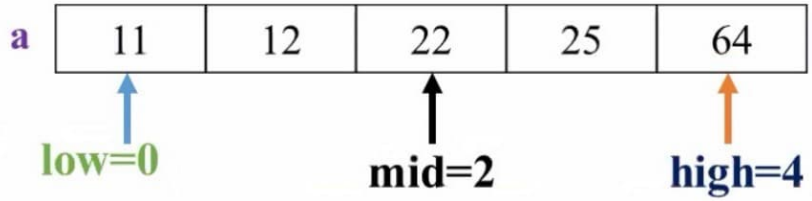


key > a[mid]
So low = mid+1;
mid = (low + high)/2

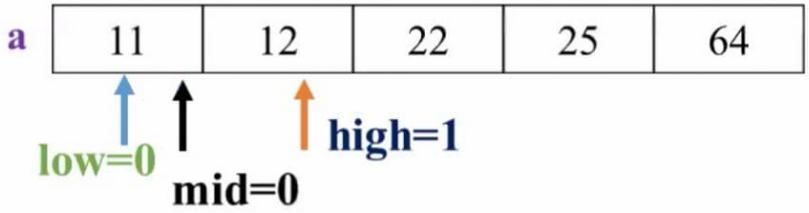


key = a[mid] = 3

Binary Search algorithm to search 11 (key = 11)

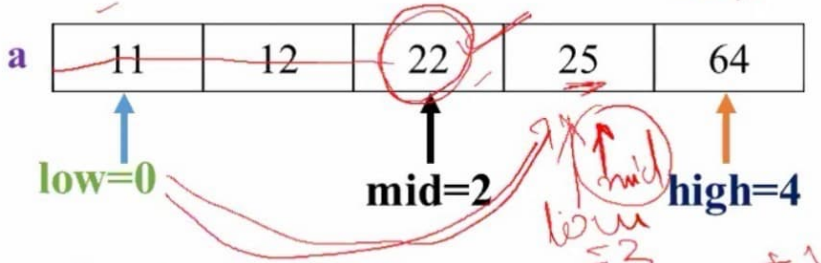


key < a[mid]
So high = mid-1;
mid=(low + high)/2



key = a[mid] = 0

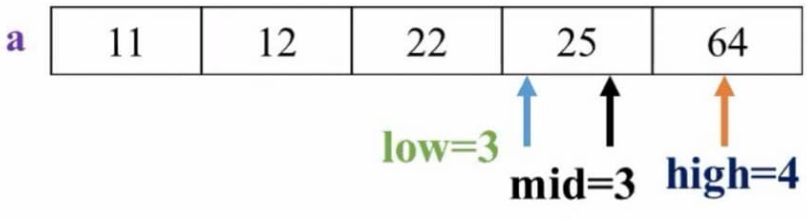
Binary Search algorithm to search 25 (key=25)



key > a[mid]

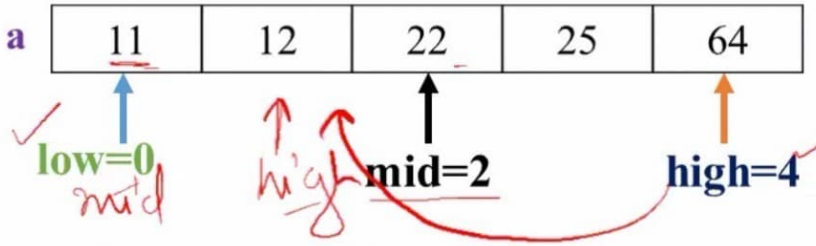
So low = mid+1;

mid = (low + high)/2



key = a[mid] = 3

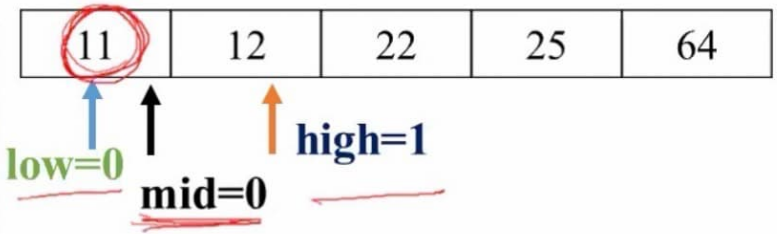
Binary Search algorithm to search 11 (key = 11)



key < a[mid]

So high = mid-1;

mid=(low + high)/2



key = a[mid] = 0

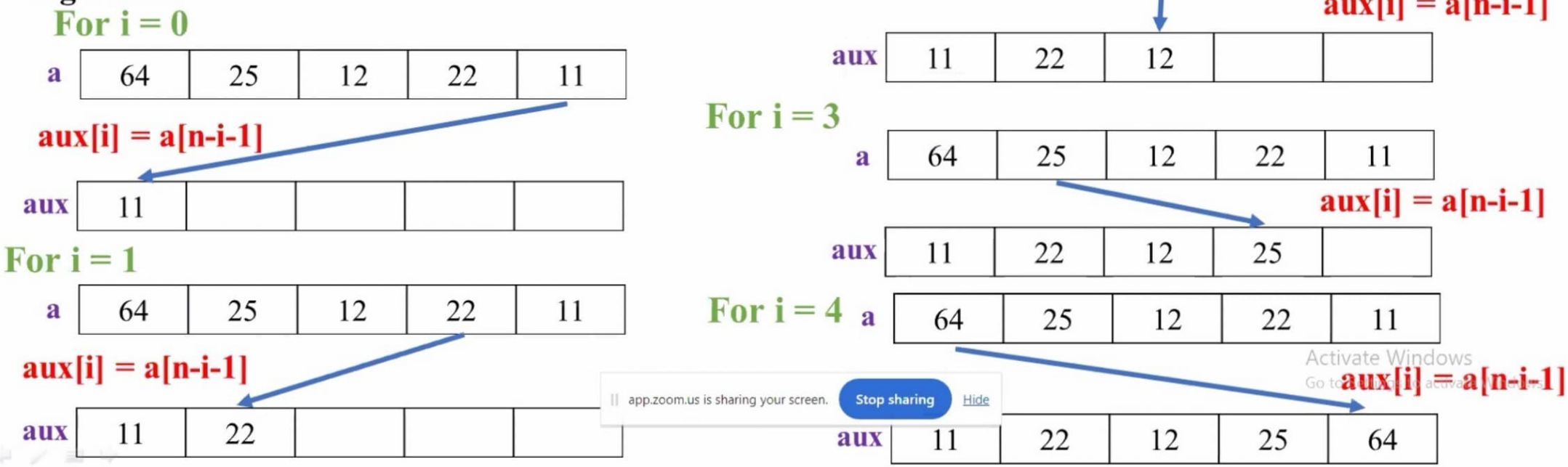
5. Reversing

- Reversing an array is one of the easiest problems, while there are several methods to reverse an array in C, two of them being the most efficient solutions; Using an **auxiliary array** and **In-place Swapping**.

Declaring auxiliary array

- We can declare another array, iterate the original array from backward and send them into the new array from the beginning.

Algorithm:



Write a program to reverse an array

$$n = \frac{\text{sizeof(arr)}}{\text{size of(int)}} = 20$$

```
#include<iostream>
using namespace std;
int main()
{
   int arr[5]={64,25,12,22,11};
   int i; int aux[5]; int n=5;
   for (i=0;i<5;i++)
   {
      aux[i]=arr[n-1-i];
   }
   for (i=0;i<5;i++)
   {
      cout<<aux[i]<<'\t';
   }
   return(0);
}
```

Output:

11 22 12 25 64

Write a program to reverse an array

$$n = \frac{\text{sizeof(arr)}}{\text{size of(int)}} = \frac{20}{4} = 5$$

```
#include<iostream>
using namespace std;
int main()
{
   int arr[5]={64,25,12,22,11};
   int i; int aux[5]; int n=5;
   for (i=0;i<5;i++)
   {
      aux[i]=arr[n-1-i];
   }
   for (i=0;i<5;i++)
   {
      cout<<aux[i]<<'\t';
   }
   return(0);
}
```

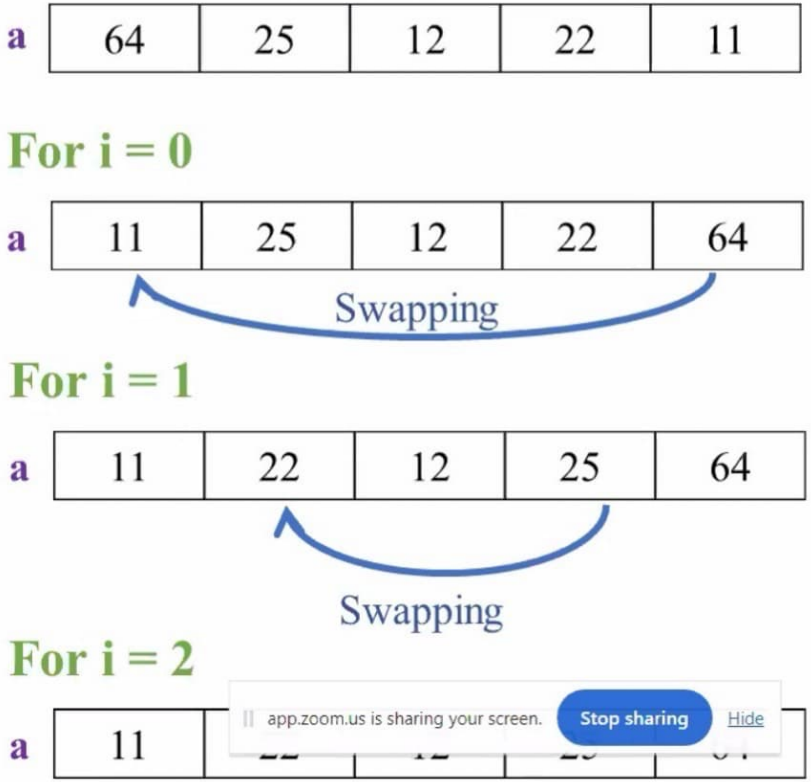
Output:

11 22 12 25 64

In-place Swapping

- We iterate the array till size/2, and for an element in index i, we swap it with the element at index (size - i - 1).
- We can keep iterating and swapping elements from both sides of the array till the middle element.

Algorithm: n is the number of elements of array



```
#include<iostream>
using namespace std;
int main()
{
   int a[5]={64,25,12,22,11};
   int i, temp; int n=5;
   for (i=0;i<n/2;i++)
   {
      temp=a[i];
      a[i]=a[n-1-i];
      a[n-1-i]=temp;
   }
   for (i=0;i<5;i++)
   {
      cout<<a[i]<<'\t';
   }
   return(0);
}
```

Output:

11 22 12 25 64

6. Insertion

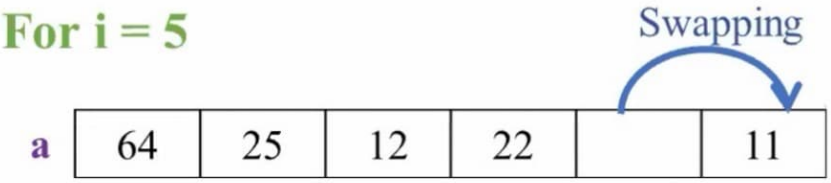
- We can insert the elements wherever we want, which means we can insert either at starting position or at the middle or at last or anywhere in the array.
- After inserting the element in the array, the positions or index location is increased but it does not mean the size of the array is increasing.

The logic used to insert element is −

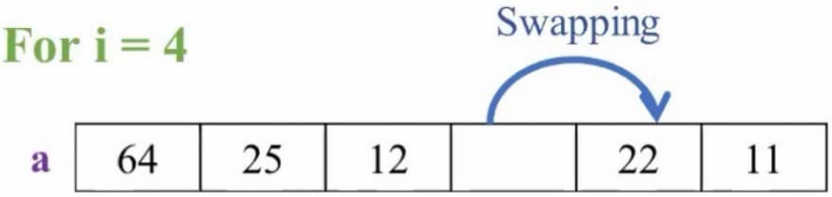
- ✓ Enter the size of the array
- ✓ Enter the position where you want to insert the element
- ✓ Next enter the number that you want to insert in that position

Insertion algorithm to insert 30 at position 2

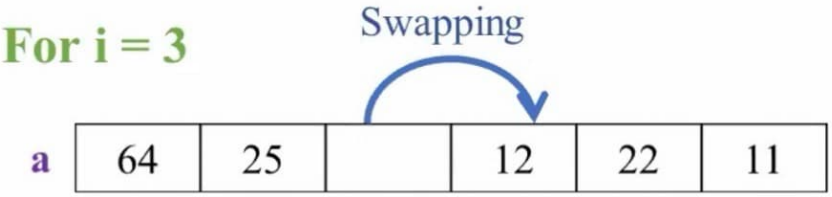
For i = 5



For i = 4

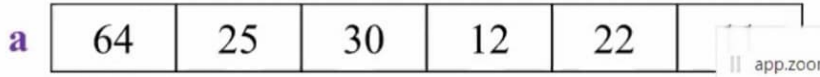


For i = 3

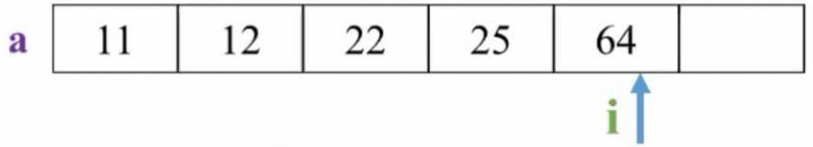


For i = 2

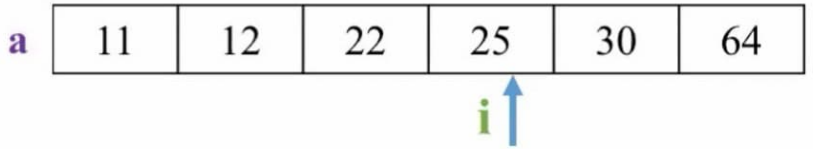
pos = i; So a[pos] = 30



Insertion algorithm to insert 30 (key) for shorted array



a[i] <= key, then a[i+1]=a[i]



a[i] > key, then a[i+1]=key;
break;

Insertion algorithm to insert 30 at position 2

```
#include<iostream>
using namespace std;
int main()
{
   int a[5]={64,25,12,22,11};
   int i, key; int n=5;
   key=30;
   for (i=5;i>2;i--)
   {
      a[i]=a[i-1];
   }
   a[2]=key;
   for (i=0;i<=5;i++)
   {
      cout<<a[i]<<'\t';
   }
   return(0);
}
```

Output:

64 25 30 12 22 11

Insertion algorithm to insert 18 (key) for shorted array

```
#include<iostream>
using namespace std;
int main()
{
   int a[5]={11,12,22,25,64};
   int i; int key=18;
   for (i=4;i>=0;i--)
   {
      if(a[i]>key)
      {
         a[i+1]=a[i];
      }
      else
      {
         a[i+1]=key;
         break;
      }
   }
   for (i=0;i<=5;i++)
   {
      cout<<a[i]<<'\t';
   }
}
```

Insertion algorithm to insert 18 (key) for shorted array

key = 18

a[1] > key

a[5] = a[4]

a[4] = a[3]

a[3] = a[2]

a[2] = 18

11 12 18 22 25 64

```
#include<iostream>
using namespace std;
int main()
{
   int a[5]={11,12,22,25,64};
   int i; int key=18;
   for (i=4;i>=0;i--)
   {
      if(a[i]>key)
      {
         a[i+1]=a[i];
      }
      else
      {
         a[i+1]=key;
         break;
      }
   }
   for (i=0;i<=5;i++)
   {
      cout<<a[i]<<'\t';
   }
}
```

7. Deletion

- To delete a specific element from an array, a user must define the position from which the array's element should be removed.

64 25 12 22 11

Steps

Step 1: Input the size of the array a[] using num, and then declare the pos variable to define the position, and i represent the counter value.

Step 2: Use a loop to insert the elements in an array until (i < num) is satisfied.

Step 3: Now, input the position of the particular element that the user or programmer wants to delete from an array.

Step 4: Compare the position of an element (pos) from the total no. of elements (num+1). If the pos is greater than the num+1, the deletion of the element is not possible and jump to step 7.

Step 5: Else removes the particular element and shift the rest elements' position to the left side in an array.

Step 6: Display the resultant array after deletion or removal of the element from an array.

```
#include<iostream>
using namespace std;
int main()
{
   int a[5]={64,25,12,22,11};
   int pos=2;
   for (int i=pos;i<5;i++)
   {
      a[i]=a[i+1];
   }
   for (int i=0;i<4;i++)
   {
      cout<<a[i]<<'\t';
   }
   return(0);
}
```

Output:

64 25 22 11

```
#include<iostream>
using namespace std;
int main()
{
   int a[5]={64,25,12,22,11};
   int pos=2;
   for (int i=pos;i<5;i++)
   {
      a[i]=a[i+1];
   }
   for (int i=0;i<4;i++)
   {
      cout<<a[i]<<'\t';
   }
   return(0);
}
```

i = 2
a[2] = a[3]
i = 3
a[3] = a[4]
i = 4
a[4] = a[5]

64 25 22 22 11
64 25 22 11 11
64 25 22 11 Garb

Output:

64 25 22 11

8. Merging

- Combining two arrays into a single array is known as merging two arrays. For example, if the first array has 5 elements and the second array has 4, the resultant array will have 9 elements.
- To merge any two arrays in C programming, start adding each and every element of the first array to the third array (the target array). Then start appending each and every element of the second array to the third array (the target array), as shown in the program given below.