IEEE SCT SB

IEEE

Code in 10 days

Day 4

dotSlash
Workshop series

Topics for Today

- Array
- Array Operations
- 2D Array

Arrays

- An array is a collection of elements of the same

type placed in contiguous memory locations.

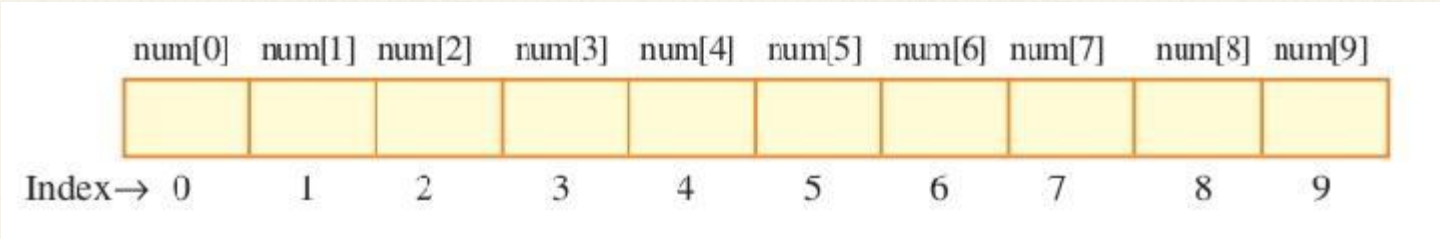
- Arrays are used to store a set of values of the

same type under a single variable name.

Declaring Arrays

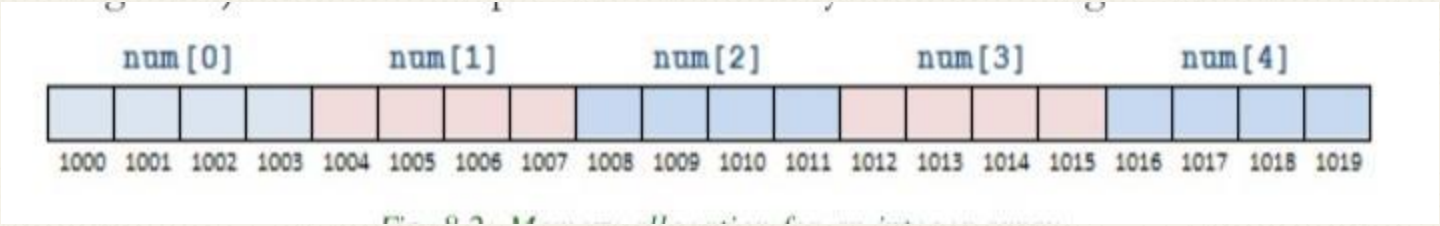
data_type array_name[size];
int num[10];

The above statement declares an array named num that can store 10 integer numbers.
Each item in an array is called an element of the array.



Memory allocation of Arrays

total_bytes = sizeof(array_type) × size_of_array
For example, total bytes allocated for the array declared as float num[10]; will be
4 × 10 = 40 bytes.



Array initialisation

```
int score[5] = {98, 87, 92, 79, 85};

char code[6] = {‘s’, ‘a’, ‘m’, ‘p’, ‘l’, ‘e’};

float wgpa[7] = {9.60, 6.43, 8.50, 8.65,
5.89, 7.56, 8.22};


int num[] = {16, 12, 10, 14, 11};
```

Array example

```
//To input the scores of 5 students and display them in
reverse order
#include <iostream>
using namespace std;
int main()
{
int i, score[5];
for(i=0; i<5; i++) // Reads the scores
{
cout<<"Enter a score: ";
cin>>score[i];
}
for(i=4; i>=0; i--) // Prints the scores
cout<<"score[" << i << "] is " << score[i]<<endl;
return 0;
}
```

Array operations

The operations performed on arrays include

- Traversal
- Searching
- Insertion
- Deletion
- Sorting
- Merging

Traversal of an Array

Traversal means accessing each element of the array at least once.

```
// Traversal of an array
#include <iostream>
using namespace std;
int main()
{
int a[10], i;
cout<<"Enter the elements of the array
:";
for(i=0; i<10; i++)
 cin >> a[i];
for(i=0; i<10; i++)
 a[i] = a[i] + 1;
cout<<"\nEntered
elements of the array
are...\n";
for(i=0; i<10; i++)
 cout<< a[i]<< "\t";
return 0;
}
```

Sorting

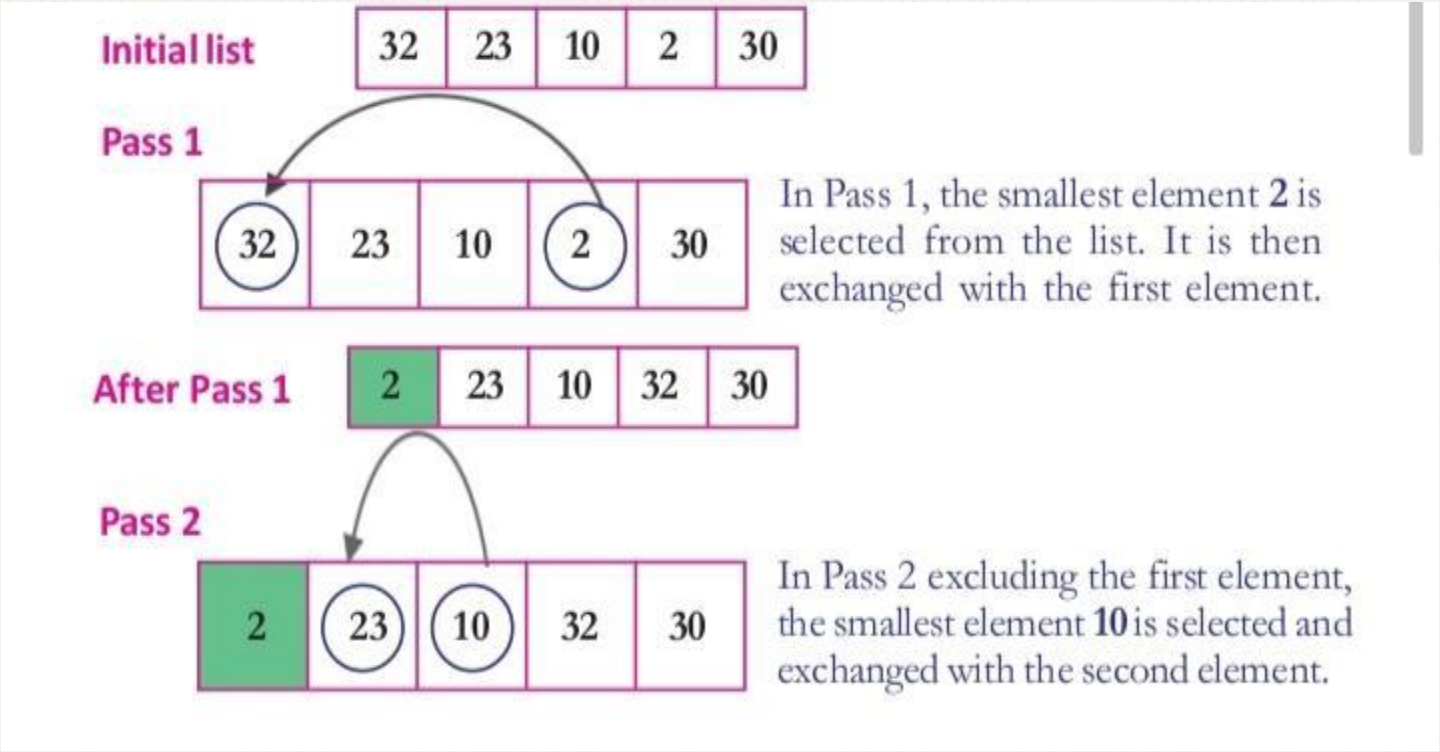
- Sorting is the process of arranging the elements of the array in some logical order.
- This logical order may be ascending or descending in case of numeric values or

dictionary order in case of strings.

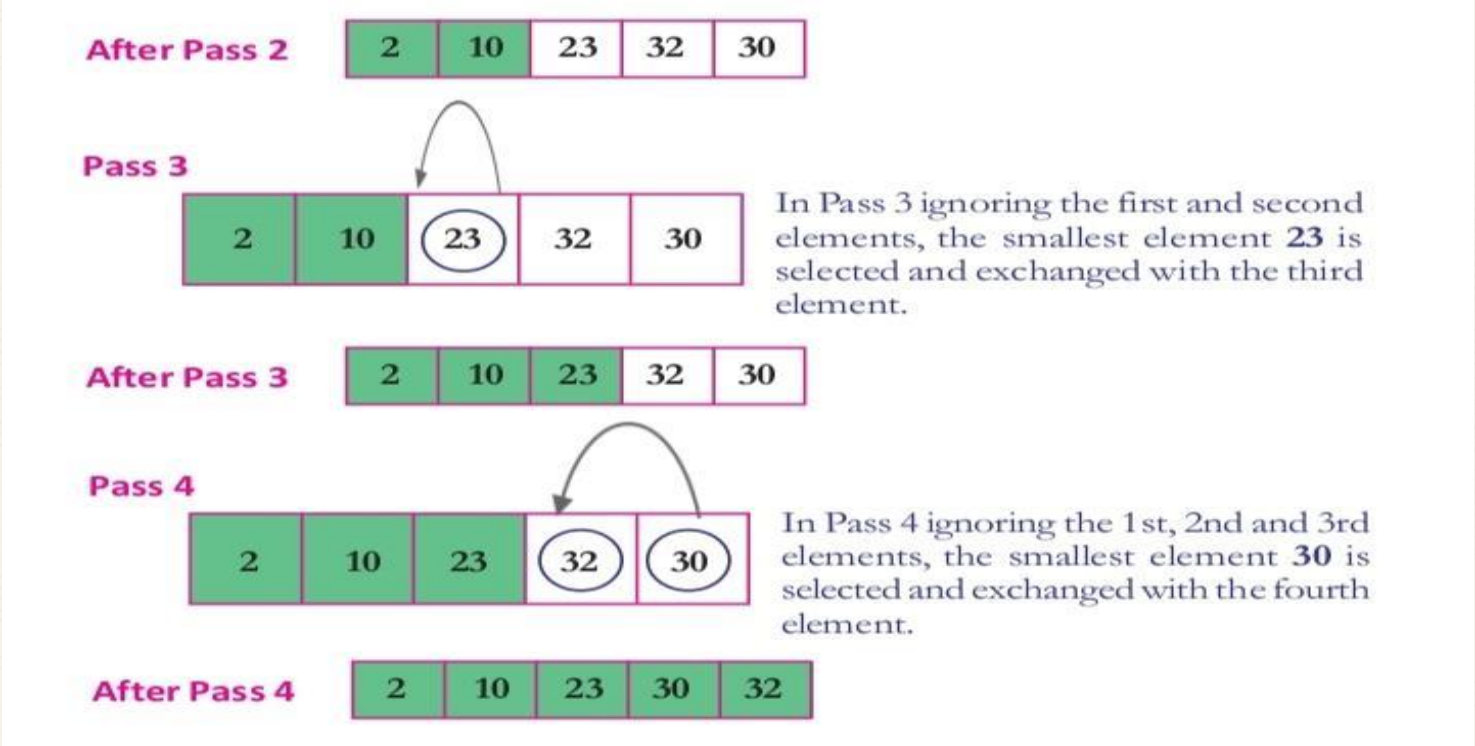
- Two types of sorts are:

1. Selection sort
2. Bubble sort

Selection sort



Selection sort



Selection sort algorithm

Step 1. Start
Step 2. Accept a value in N as the number of elements of the array
Step 3. Accept N elements into the array AR
Step 4. Repeat Steps 5 to 9, (N – 1) times
Step 5. Assume the first element in the list as the smallest and store it in MIN
and its position in POS
Step 6. Repeat Step 7 until the last element of the list
Step 7. Compare the next element in the list with the value of MIN. If it is found
smaller, store it in MIN and its position in POS
Step 8. If the first element in the list and the value in MIN are not the same, then
swap the first element with the element at position POS
Step 9. Revise the list by excluding the first element in the current list

Step 10. Print the sorted array AR
Step 11. Stop

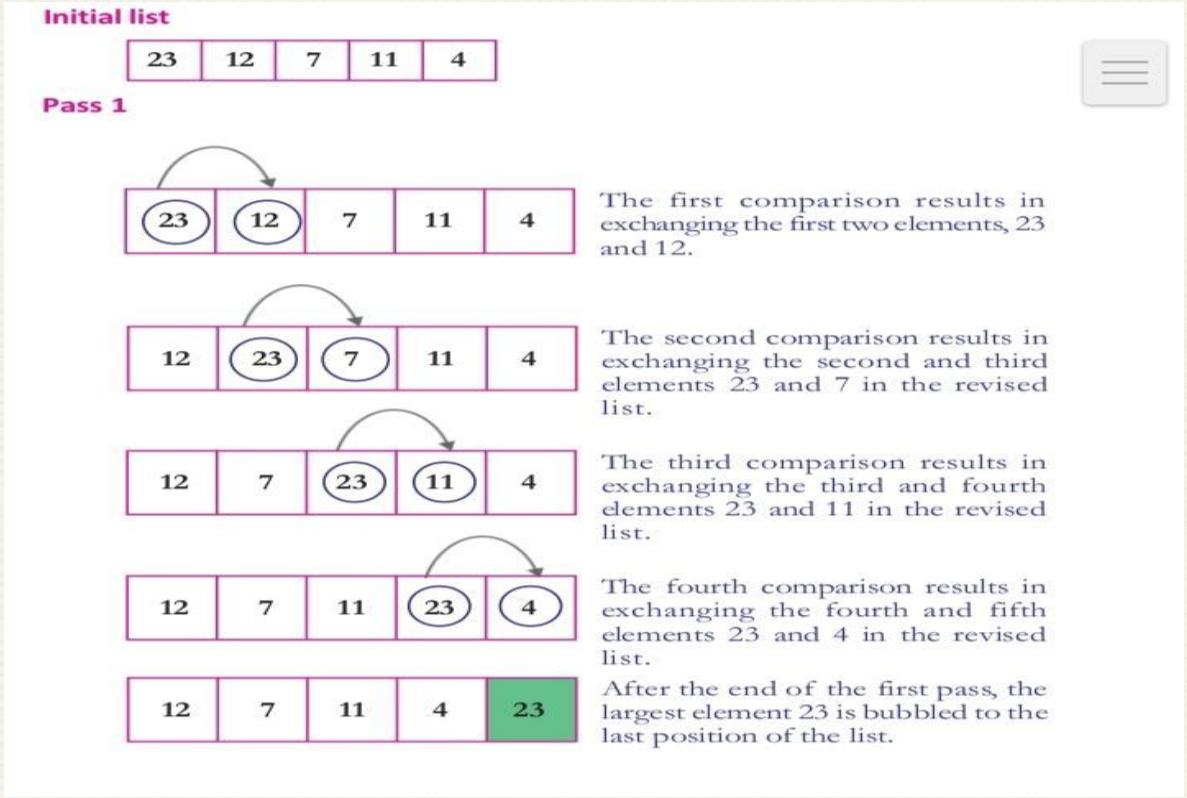
Selection sort program

```
// Selection sort for arranging elements in ascending
order
#include <iostream>
using namespace std;
int main()
{ int AR[25], N, I, J, MIN, POS;
cout<<"How many elements? ";
cin>>N;
cout<<"Enter the array elements: ";
for(I=0; I<N; I++)
cin>>AR[I];
for(I=0; I < N-1; I++)
{
MIN=AR[I];
POS=I;
for(J = I+1; J < N; J++)
if(AR[J]<MIN)
{
MIN=AR[J];
POS=J;
}
if(POS != I)
{
AR[POS]=AR[I];
AR[I]=MIN;
}
}
cout<<"Sorted array is: ";
for(I=0; I<N; I++)
cout<<AR[I]<<"\t";
return 0;
}
```

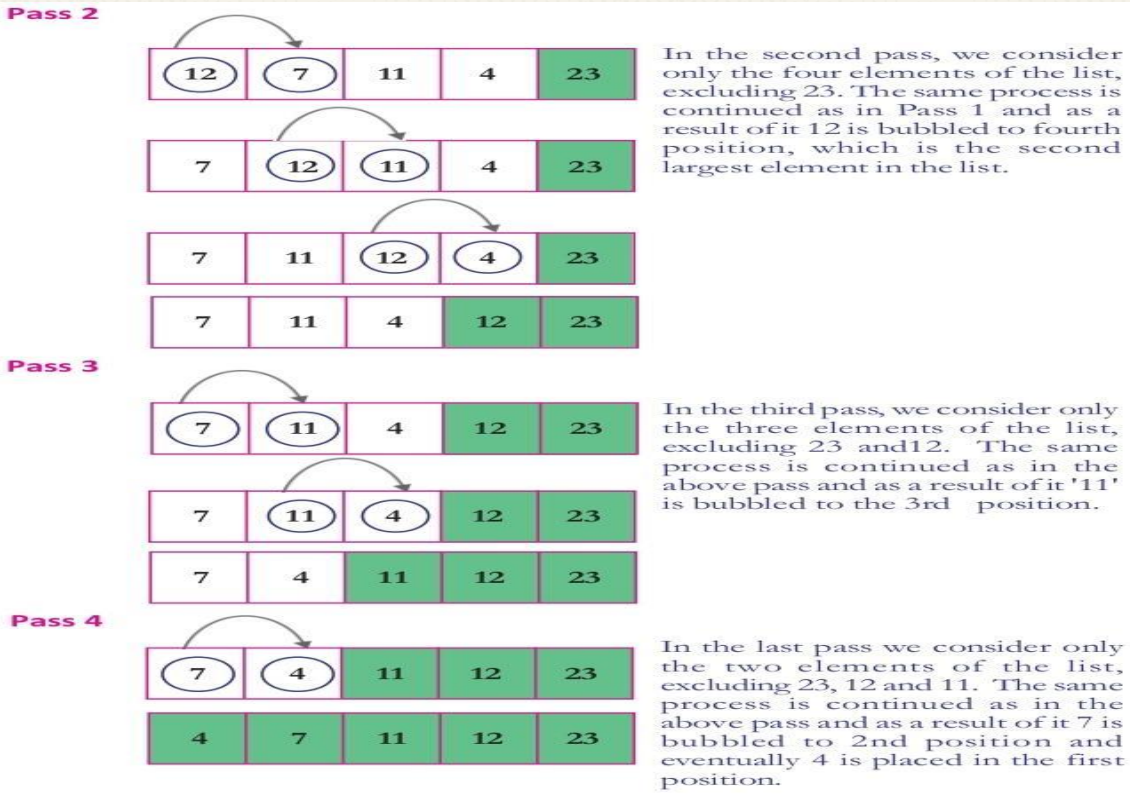
Bubble sort

- Bubble sort is a sorting algorithm that works by repeatedly stepping through lists that need to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order.
- This passing procedure is repeated until no swaps are required, indicating that the list is sorted.

Bubble sort


Initial list
23 12 7 11 4
Pass 1
23 12 7 11 4
The first comparison results in exchanging the first two elements, 23 and 12.
12 23 7 11 4
The second comparison results in exchanging the second and third elements 23 and 7 in the revised list.
12 7 23 11 4
The third comparison results in exchanging the third and fourth elements 23 and 11 in the revised list.
12 7 11 23 4
The fourth comparison results in exchanging the fourth and fifth elements 23 and 4 in the revised list.
12 7 11 4 23
After the end of the first pass, the largest element 23 is bubbled to the last position of the list.

Bubble sort



In the second pass, we consider only the four elements of the list, excluding 23. The same process is continued as in Pass 1 and as a result of it 12 is bubbled to fourth position, which is the second largest element in the list.

In the third pass, we consider only the three elements of the list, excluding 23 and12. The same process is continued as in the above pass and as a result of it '11' is bubbled to the 3rd position.

In the last pass we consider only the two elements of the list, excluding 23, 12 and 11. The same process is continued as in the above pass and as a result of it 7 is bubbled to 2nd position and eventually 4 is placed in the first position.

Bubble sort algorithm

Step 1. Start
Step 2. Accept a value in N as the number of elements of the array
Step 3. Accept N elements into the array AR
Step 4. Repeat Steps 5 to 7, (N - 1) times
Step 5. Repeat Step 6 until the second last element of the list
Step 6. Starting from the first position, compare two adjacent elements in the
list. If they are not in proper order, swap the elements.
Step 7. Revise the list by excluding the last element in the current list.
Step 8. Print the sorted array AR
Step 9. Stop

Bubble sort program

```
#include<iostream>
using namespace std;
int main()
{
   int n, i, arr[50], j, temp;
   cout<<"Enter the Size (max. 50): ";
   cin>>n;
   cout<<"Enter "<<n<<" Numbers: ";
   for(i=0; i<n; i++)
      cin>>arr[i];
   cout<<"\nSorting the Array using Bubble Sort Technique..\n";
   for(i=0; i<(n-1); i++)
   {
      for(j=0; j<(n-i-1); j++)
      {
         if(arr[j]>arr[j+1])
         {
            temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
         }
      }
   }
   cout<<"\nArray Sorted Successfully!\n";
   cout<<"\nThe New Array is:\n";
   for(i=0; i<n; i++)
      cout<<arr[i]<<" ";
   cout<<endl;
   return 0;
}
```

Searching

- Searching is the process of finding the location of the given element in the array.
- The search is said to be successful if the given element is found, that is the element
- exists in the array; otherwise unsuccessful.
- Two types are
    1. linear search
    2. Binary Search

Linear search

- Linear search or sequential search is a method for finding a particular value in a list.
- Assume that the element ‘45’ is to be searched from a sequence of elements 50, 18, 48, 35, 45, 26, 12.
- Linear search starts from the first element 50, comparing each element until it reaches the 5th Position where if finds 45

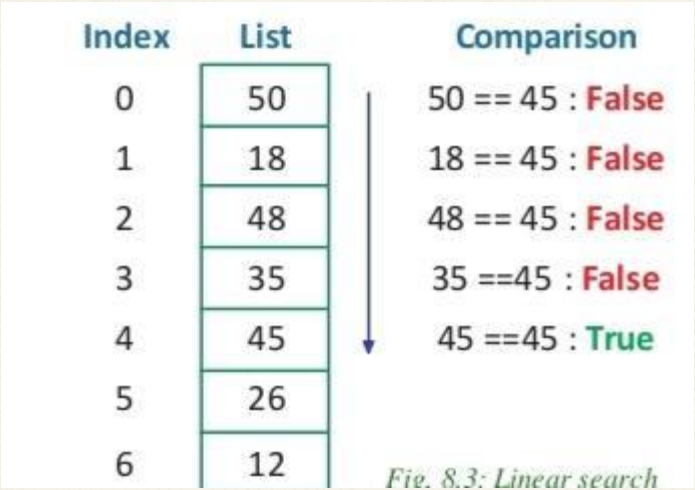


Fig. 8.3: Linear search

Linear search algorithm

Algorithm for Linear Search
Step 1. Start
Step 2. Accept a value in N as the number of elements of the array
Step 3. Accept N elements into the array AR
Step 4. Accept the value to be searched in the variable ITEM
Step 5. Set LOC = -1
Step 6. Starting from the first position, repeat Step 7 until the last element
Step 7. Check whether the value in ITEM is found in the current position. If found then store the position in LOC and Go to Step 8, else move to the next position.
Step 8. If the value of LOC is less than 0 then display "Not Found", else display the value of LOC + 1 as the position of the search value.
Step 9. Stop

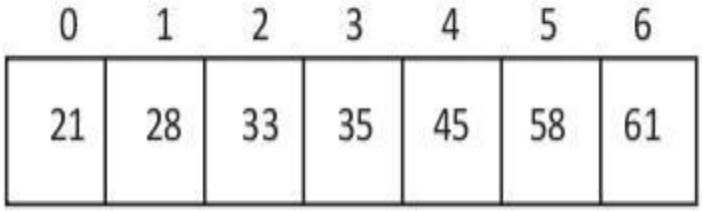
Linear search program

```
//Linear search to find an item in the array
#include <iostream>
using namespace std;
int main()
{
int AR[25], N;
int I, ITEM, LOC=-1;
cout<<"How many elements? ";
cin>>N;
cout<<"Enter the array elements: ";
for(I=0; I<N; I++)
 cin>>AR[I];
 cout<<"Enter the item you are searching for:
";
cin>>ITEM;
for(I=0; I<N; I++)
if(AR[I] == ITEM)
{
LOC=I;
break;
}
if(LOC!=-1)
cout<<"The item is found at
position "<<LOC+1;
else
cout<<"The item is not found
in the array";
return 0;
```

Binary search

- Binary search is an algorithm which uses minimum number of searches for locating the position of an element in a sorted list, by checking the middle, eliminating half of the list from consideration, and then performing the search on the remaining half.
- If the middle element is equal to the searched value, then the position has been found; otherwise the upper half or lower half is chosen for search, based on whether the element is greater than or less than the middle element.

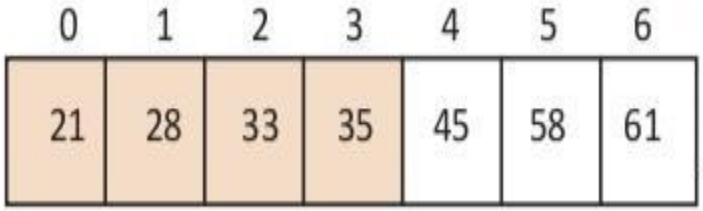
Binary search



FIRST = 0

LAST = 6

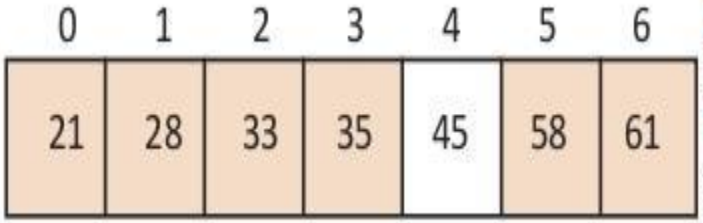
As FIRST<=LAST, let's start iteration



MIDDLE = (FIRST+LAST)/2 = (0+6)/2 = 3
Here LIST[3] is not equal to **45** and LIST[3] is less than search element therefore, we take
FIRST = MIDDLE + 1 = 3 + 1 = 4, LAST = 6

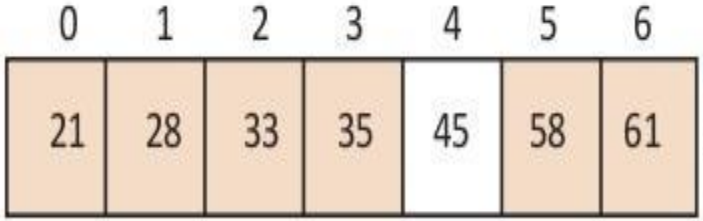
Binary search

As FIRST<=LAST, we start next iteration.



MIDDLE = (FIRST+LAST)/2 = (4+6)/2= 5
Here LIST[5] is not equal to **45** and LIST[5] is greater than the search element therefore, we take
FIRST = 4, LAST = MIDDLE - 1=5 -1 = 4,

As FIRST<=LAST, we start next iteration



MIDDLE = (FIRST+LAST)/2 = (4+4)/2 = 4
Here LIST[4] is equal to **45** and the **search terminates successfully**.

Binary search algorithm

Step 1. Start
Step 2. Accept a value in MAX as the number of elements of the array
Step 3. Accept MAX elements into the array LIST
Step 4. Accept the value to be searched in the variable ITEM
Step 5. Store the position of the first element of the list in FIRST and that of the last in LAST
Step 6. Repeat Steps 7 to 11 While (FIRST <= LAST)
Step 7. Find the middle position using the formula (FIRST + LAST)/2 and store it in MIDDLE
Step 8. Compare the search value in ITEM with the element at the MIDDLE of the list
Step 9. If the MIDDLE element contains the search value in ITEM then stop search, display the position and go to Step 12.
Step 10. If the search value is smaller than the MIDDLE element
Then set LAST = MIDDLE - 1
Step 11. If the search value is larger than the MIDDLE element
Then set FIRST = MIDDLE + 1
Step 12. Stop

Binary search program

```
//Binary search to find an item in the sorted
array
#include <iostream>
using namespace std;
int main()
{ int LIST[25],MAX;
int FIRST, LAST, MIDDLE, I, ITEM, LOC=-1;
cout<<"How many elements? ";
cin>>MAX;
cout<<"Enter array elements in ascending
order: ";
for(I=0; I<MAX; I++)
cin>>LIST[I];

cout<<"Enter the item to be searched: ";
cin>>ITEM;
FIRST=0;
LAST=MAX-1;
while(FIRST<=LAST)
{
MIDDLE=(FIRST+LAST)/2;
if(ITEM == LIST[MIDDLE])
{
LOC = MIDDLE;
break;
}


if(ITEM < LIST[MIDDLE])
LAST = MIDDLE-1;
else
FIRST = MIDDLE+1;
}
if(LOC != -1)
cout<<"The item is found at position "<<LOC+1;
else
cout<<"The item is not found in the array";
return 0;
 }
```

2D Array

2D Array Declaration
data_type array_name[rows][columns];

Program 1

```
// To create a matrix with m rows and n columns
#include <iostream>
using namespace std;
int main()
{ int m, n, row, col, mat[10][10];
cout<< "Enter the order of matrix: ";
cin>> m >> n;
cout<<"Enter the elements of matrix\n";
for (row=0; row<m; row++)
for (col=0; col<n; col++)
cin>>mat[row][col];
cout<<"The given matrix is:";
for (row=0; row<m; row++)
{
cout<<endl;
for (col=0; col<n; col++)
cout<<mat[row][col]<<"\t";
}
return 0;
}
```

Program 2

```
// To find the sum of two matrices if
conformable
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{ int m1, n1, m2, n2, row, col;
int A[10][10], B[10][10], C[10][10];
cout<<"Enter the order of first matrix: ";
cin>>m1>>n1;
cout<<"Enter the order of second matrix: ";
cin>>m2>>n2;
if(m1!=m2 || n1!=n2)
{
cout<<"Addition is not possible";
exit(0);
}
cout<<"Enter the elements of first matrix\n";
for (row=0; row<m1; row++)
for (col=0; col<n1; col++)
cin>>A[row][col];
cout<<"Enter the elements of second matrix\n";
for (row=0; row<m2; row++)
for (col=0; col<n2; col++)
cin>>B[row][col];
for (row=0; row<m1; row++)
for (col=0; col<n1; col++)
C[row][col] = A[row][col] + B[row][col];
cout<<"Sum of the matrices:\n";
for(row=0; row<m1; row++)
{
cout<<endl;
for (col=0; col<n1; col++)
cout<<C[row][col]<<"\t";
}
}
```

Program 3

```
//To find the sum of major diagonal elements of
a matrix
#include <iostream>
using namespace std;
int main()
{ int mat[10][10], n, i, j, s=0;
cout<<"Enter the rows/columns of square matrix:
";
cin>>n;
cout<<"Enter the elements\n";
for(i=0; i<n; i++)
for(j=0; j<n; j++)
cin>>mat[i][j];
cout<<"Major diagonal elements are\n";
for(i=0; i<n; i++)
{
cout<<mat[i][i]<<"\t";
s = s + mat[i][i];
}
cout<<"\nSum of major
diagonal elements is: ";
cout<<s;
return 0;
}
```

Program 4

```
// To find the transpose of a matrix
#include <iostream>
using namespace std;
int main()
{ int ar[10][10], m, n, row, col;
cout<<"Enter the order of matrix: ";
cin>>m>>n;
cout<<"Enter the elements\n";
for(row=0; row<m; row++)
for(col=0; col<n; col++)
cin>>ar[row][col];
cout<<"Original matrix is\n";
for(row=0; row<m; row++)
{
cout<<"\n";
for(col=0; col<n; col++)
cout<<ar[row][col]<<"\t";
}
cout<<"\nTranspose of the
entered matrix is\n";
for(row=0; row<n; row++)
{
cout<<"\n";
for(col=0; col<m; col++)
cout<<ar[col][row]<<"\t";
}
return 0;
}
```

Program 5

```
// To find the row sum and column sum of a
matrix
#include <iostream>
using namespace std;
int main()
{
int ar[10][10], rsum[10]={0}, csum[10]={0};
int m, n, row, col;
cout<<"Enter the number of rows & columns in
the array: ";
cin>>m>>n;
cout<<"Enter the elements\n";
for(row=0; row<m; row++)
for(col=0; col<n; col++)
cin>>ar[row][col];
for(row=0; row<m; row++)
for(col=0; col<n; col++)
{
rsum[row] += ar[row][col];
csum[col] += ar[row][col];
}
cout<<"Row sum of the 2D
array is\n";


for(row=0; row<m; row++)
cout<<rsum[row]<<"\t";
cout<<"\nColumn sum of the
2D array is\n";
for(col=0; col<n; col++)
cout<<csum[col]<<"\t";
return 0;
}
```

Thank You