Unit - 3

Chapter 14

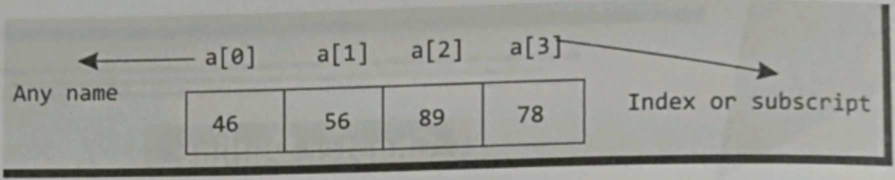
Array Techniques

Chapter Outline

14.1 Array Techniques

An array is an indexed collection of data elements of the same type. Indexed means that the array elements are numbered (starting at 0). Arrays are stored in consecutive memory cells. Every cell must be the same type and same size.

All the elements will have the same name, but each element is accessed or identified using the index or subscript or position value. The index will always begin with 0, hence if there are 'n' elements in an array the index of last element will be n – 1. The index is used to access the individual element.



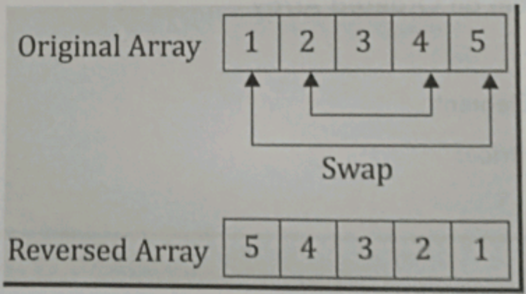
One Dimensional Array Example

Arrays plays an important role in computer algorithms as using arrays simplifies the operations performed on collection of data. Let us discuss some problems related to arrays.

14.2 Array Order Reversal

Problem

Rearrange the elements in an array so that they appear in reverse order



Consider an array a[5] .The elements before reversal and after reversal are given below.

Array element	a[0]	a[1]	a[2]	a[3]	a[4]
Input	14	21	67	32	45
Output	45	32	67	21	14

To reverse an array, the concept of swapping is used. Swapping is a term used for interchanging the values at two different locations with each other.

The set of exchanges in the reversal process is given below:

Step 1: Exchange a[0] ⇔ a[4]

Step 2: Exchange a[1] ⇔ a[3]

Step 3: a[2] ⇔ a[2] - (No Exchange)

The logic of reversing an array is to swap elements of the first half with elements of the second half i.e. the first element with the last element, then the second element with the second last element, and so on, till the middle element of the array. In each step, the index on the left is increasing by one and the index on the right is decreasing by 1. The index from left to right is denoted by i and the index from right to left is denoted by j.

In the example, it has five elements. so n = 5, i = 0, j = n-1 =5-1 = 4

Increasing index i	Decreasing index j.
0	4
1	3
2	2

We need to exchange the elements till the middle element of the array. We should exchange the elements until i < j . Therefore, two exchanges are required to reverse an array of 5 elements.

In general the, exchanges can be formulated as

```
temp  = a[i]
a[i] = a[j]
a[j]  = temp
```

Where temp is a temporary variable used for exchange.

Algorithm **To Reverse Elements of an Array.**

```
Let a[0…n-1] be an array of n elements to be reversed.
Step 1: Start
Step 2: Declare variables a[] , n, i , j, and temp
Step 3: Read n elements and store in array A
Step 4: Initialze i = 0, j = n-1
Step 5: While i < j , do the following steps
        (a) exchange i^th element with j^th element.
        (b) increment i by 1 and decrement j by 1
Step 6: Print elements of an array a.
Step 7: End
```

Example

Let us consider the below array of 5 elements.

a[0]	a[1]	a[2]	a[3]	a[4]
14	21	67	32	45

Initially, i = 0, j = n-1 = 4

Iteration	while (i < j)	Swap i^th and j^th element	i = i+1	j = j- 1

1	while (0 < 4) is True	Swap a[0] and a[4]
		After swapping
		a[0]= 45
		a[4] = 14
			i = 0 +1
			i = 1
				j = 4-1
				j =3

2	while (1 < 3) is True	Swap a[1] and a[3]
		After swapping
		a[1]= 32
		a[3] = 21
			i = 1 +1
			i = 2
				j = 3-1
				j =2

3	while (2 < 2) is False			

An array after reversal is shown below.

a[0]	a[1]	a[2]	a[3]	a[4]
45	32	67	21	14

Program 1 C Program to Reverse Elements of an Array.

```
#include <stdio.h>

void main()
{
    int n,i,j,temp;
    printf("Enter the size of an array: ");
    scanf("%d", &n);
    int a[n];
    printf("\n Enter %d elements: ");
    for(i=0; i<n; i++)
      scanf("%d", &a[i]);
        i=0;
    j=n-1;
    while (i<j)
    {
       temp = a[i];
       a[i] = a[j];
       a[j] = temp;
       i++;
       j--;
    }
   printf("\n After reversing an array :");
   for(i=0; i<n; i++)
      printf(" %d ",a[i]);
}
```

Output :

```
Enter the size of an array: 5
Enter 5 elements: 50 40 30 20 10
After reversing an array : 10  20  30  40  50
```

14.3 Array Counting or Histogramming

Problem

Given a set of n students' examination marks in the range 0 to 100. Find out the frequency count of the number of students that obtained each possible mark.

? What is Histogram?

A histogram is a graphical representation that classifies a set of data points into ranges defined by the user. The histogram, which resembles a bar graph in appearance, consolidates a data series. A histogram is used to count or visualize the frequency of data (the number of occurrences) over discrete intervals.

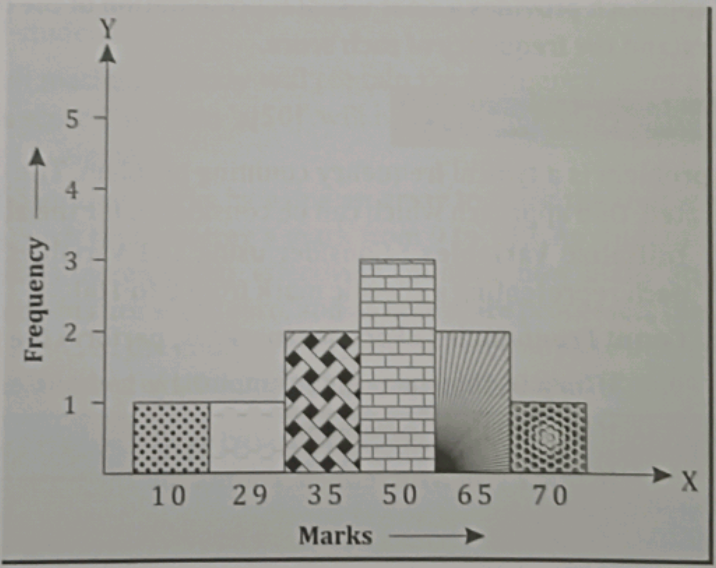
? What is Frequency Histogram?

A frequency histogram is a histogram that shows the frequencies (the number of occurrences) of the given data items.

Example Marks of 10 Students : 50, 70, 29, 50, 50, 65, 10, 35, 65, 35

Frequency of 50 = 3
Frequency of 70 = 1
Frequency of 29 = 1
Frequency of 65 = 2
Frequency of 10 = 1
Frequency of 35 = 2

This data can be now shown using a frequency histogram.



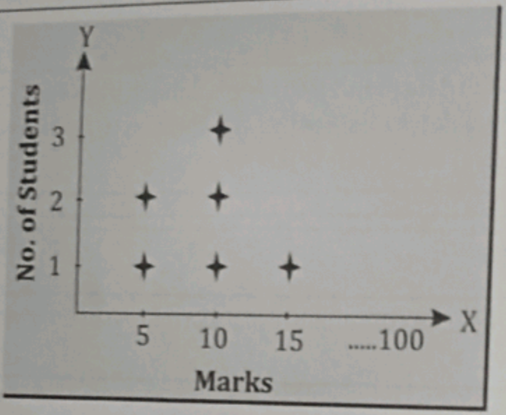
Visualizing Mark Distribution Using Frequency Histograms

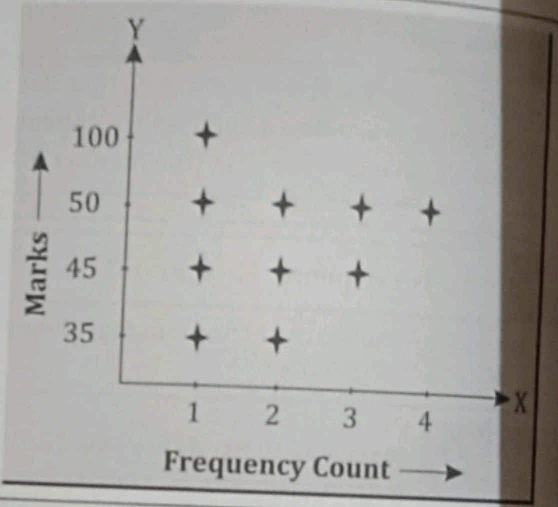
Instead of checking each mark individually with multiple conditions, a more visual approach can be taken:

1. **Vertical Frequency Histogram:** For each mark, place a star in the corresponding slot on the graph. The number of stars in each slot will represent how many times that particular mark has been achieved. This is shown in the vertical frequency histogram on the left. The Y-axis

represents the number of students, and the X-axis represents the marks. Each star corresponds to the frequency of students scoring a particular mark.

2. **Horizontal Frequency Histogram:** The horizontal frequency histogram on the right works similarly, but it displays the data with horizontal bars. The Y-axis represents the marks, and the X-axis represents the frequency count. The width of each bar corresponds to the number of students who achieved that mark.





This approach provides a clear visual representation of the distribution of marks, making it easier to understand the frequency of each score.

Algorithms Development

This problem is a typical frequency counting problem. The distribution of a set of marks need to be generated. One approach which can be considered for the above problem is as follows:

1. **Initialize Variables :** Consider using 101 variables: 'count0, count1, count2, ..., count100', each representing a specific mark from 0 to 100.
2. **Count Frequencies :** For each mark 'm', perform the following:
   - If 'm = 0', then increment 'count0' by 1.
   - If 'm = 1', then increment 'count1' by 1.
   - If 'm = 2', then increment 'count2' by 1.

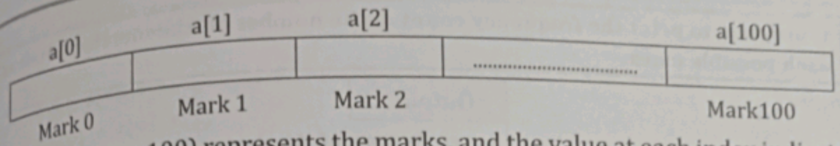
     ...
   - If 'm = 100', then increment 'count100' by 1.

However, this method requires 101 separate checks for each mark, leading to an unnecessarily lengthy and inefficient program.

A More Efficient Approach

A more streamlined solution involves using an array to track the frequency of each mark.

1. **Initialize an Array :** Create an array with 101 elements, where each index corresponds to a mark value from 0 to 100.



The index (0 to 100) represents the marks, and the value at each index indicates the frequency count of that mark.

2. **Count Frequencies :** For each mark, increment the corresponding array element. For instance:
   - If a student scores 50, increment 'a[50]' by 1.
   - If another student scores 75, increment 'a[75]' by 1.
   - If another student scores 50 again, increment 'a[50]' by 1.

   This can be generalized as:

   $a[m] = a[m] + 1$

   where 'm' is the mark obtained by a student.
3. **Update Counts :** Initially, all elements of the array are set to 0. As we read each student's mark, update the corresponding array element:
   - If the first student's mark is 50, then 'a[50]' becomes '0 + 1 = 1'.
   - If the second student's mark is 75, then 'a[75]' becomes '0 + 1 = 1'.
   - If the third student's mark is 50, then 'a[50]' becomes '1 + 1 = 2'.
   - Continue this process for all students.
4. **Final Output :** After processing all marks, the array will contain the frequency count of each mark. For example, if 25 students scored 50, then 'a[50]' will hold the value 25. Finally, array contains the frequency count of each element.

This approach simplifies the process of counting marks by using an array to track the frequency of each score. The array has 101 elements, each representing a mark from 0 to 100. As student marks are read, the corresponding array index is incremented, effectively counting how many students received each mark. Initially, all array elements are set to zero, and as marks are processed, the array updates to reflect the frequency of each score. In the end, the array provides a complete distribution of the marks, with each index holding the number of students who achieved that specific mark.

Algorithm **To Display the Frequency Count of each element in an array**

```
Step 1: Start
Step 2: Input n, number of marks to be processed.
Step 3: Initialize counting array elements a[0, ….100] to zero.
Step 4: While n > 0, do
        (i) Read next mark m
        (ii) Increase the count by one in location m in counting array
             i.e., a[m] = a[m] + 1
Step 5: Output frequency count distribution or histogram for marks.
Step 6: End
```

Program 2 C program to print the frequency count of the number of students that obtained each possible mark.

```
#include <stdio.h>

int main()
{
      int n,m,i,j,a[101];

     for(i=0; i<=100; i++)
        a[i] = 0;

     printf("Enter Number of Students: ");
     scanf("%d", &n);
     printf("\n Enter %d marks: ",n);
     for(i=1; i<=n; i++)
     {
        scanf("%d",&m);
        a[m] = a[m] + 1;
     }

     printf("Marks\tFrequency\n");
     for(i=0; i<=100; i++)
        if(a[i] != 0)
           printf("\n%3d :\t%3d",i,a[i]);
}
```

Output :

```
Enter Number of Students: 10
Enter 10 marks: 37 45 50 45 100 0 45 50 50 100
Marks   Frequency
  0 :      1
 37 :      1
 45 :      3
 50 :      3
100 :      2
```

Let us display the same output in Horizontal Frequency Histogram graph **in the below program.** The widths of the bars represent the frequency count.

Program 3 C program to print the frequency count of the number of students that obtained each possible mark in Horizontal Frequency Histogram graph.

```
#include <stdio.h>
int main()
{
      int n,m,i,j,a[101];

      for(i=0; i<=100; i++)
         a[i] = 0;

      printf("Enter Number of Students: ");
      scanf("%d", &n);
      printf("\n Enter %d marks: ",n);
for(i=1; i<=n; i++)
{
  scanf("%d",&m);
  a[m] = a[m] + 1;
}
printf("\nFrequency Histogram in Horizontal Format");
printf("\n****************************************\n");
printf("Marks\tFrequency\n");
for(i=0; i<=100; i++)
{
    if(a[i] != 0)
    {
    printf("\n%3d |\t",i);
      for (j=1; j<=a[i]; j++)
      printf(" * ");
    }
}
}
```

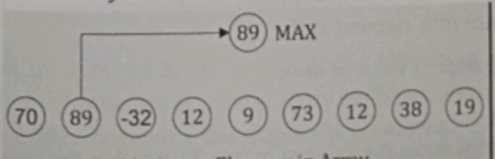
Output:

```
Enter Number of Students: 10
Enter 10 marks: 45 35 50 45 65 50 50 50 100 35

Frequency Histogram in Horizontal Format
****************************************
Marks  Frequency


 35 |   *  *
 45 |   *  *
 50 |   *  *  *  *
 65 |   *
100 |   *
```

14.4 Finding the Maximum Number in a Set

Problem

To find maximum number in an array of *n* elements.



Maximum Elemeny in Array

The maximum number is a number which is greater than or equal to all other numbers in an array of 'n' elements. An array should have one or more elements to find the maximum number. All numbers need to be examined to find the maximum number. The solution is to initialize the first element as max, then traverse the given array from second element till end. For every traversed element, compare it with max, if it is greater than max, then update max.

The approach for finding the maximum element in an array is as follows:

Step 1: Start by examining every element in the array and compare them with each other.

Step 2: Set the first element of the array as a temporary candidate for the maximum value, referred to as 'max'.

Step 3: As we examine the second number (i.e., the second element of the array), one of three scenarios can occur:

1. The second number is less than 'max'.
2. The second number is equal to 'max'.
3. The second number is greater than 'max'.

Step 4: If the second number is less than or equal to 'max', move on to the next number and compare it. However, if the second number is greater than 'max', update 'max' with the value of the second number.

This process continues until all elements in the array have been examined, and by the end, 'max' will hold the highest value in the array.

Example

Consider an array with five elements a[5] = {11, 24, 90, 35, 75};

Let temporary maximum variable, ***max*** = a[0] = 11

Step 1: Compare max with second element a[1]

11 < 24, so update the value of ***max*** *with value of second number (24)*

max = 24

Step 2: Compare max with third element a[2]

24 < 90, so update the value of ***max 90***

max = 90

Step 3: Compare max with fourth element a[3]

90 > 35, so no change in value of ***max***.

max = 90

Step 4: Compare max with fifth element a[4]

90 > 75, so no change in value of ***max***.

max = 90

After iterating through the entire array, the value of max is 90, which is the largest element in the array.

Algorithm Finding the Maximum Number in an Array of 'n' Elements

```
Let a[0...n-1] be the array, where n ≥ 1.
Step 1: Start
Step 2: Declare variables a[], n, and i.
Step 3: Read n elements of an array and store the elements in array a.
Step 4: Set first array element as temporary maximum max and initialize i to 1
        max= a[0]
        i = 1
Step 5: while i < n  do
            (a) If next array element a[i] > current maximum max then
                max = a[i]
            (b) i = i + 1
Step 6: Print largest element - max.
```

Program 4 C Program to Find the Maximum Number in an Array of n elements.

```c
#include <stdio.h>

void main()
{
 int i, max, n;

 printf(" Enter the number of elements : ");
 scanf("%d", &n);
 int a[n];
 printf("\n Enter %d elements: ", n);
 for(i=0; i<n; i++)
    scanf("%d", &a[i]);

 max = a[0];
 for(i=1; i<n; i++)
   if(max < a[i])
       max = a[i];

 printf(" Maximum number in an array is: %d ", max);
}
```

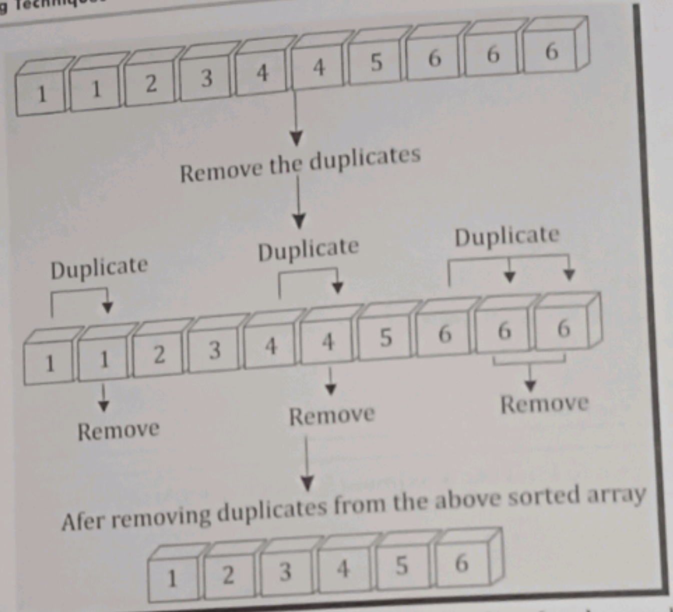
Output :

```
Enter the number of elements : 5
Enter 5 elements: 20 50 90 70 60
Maximum number in an array is: 90
```

14.5 To Remove Duplicates from an Ordered Array

Problem

Remove all duplicates from an ordered array and contract the array accordingly. Given a sorted array, we should remove all the duplicates from the array such that each element appears only once and array should be ordered after removing the duplicate elements.



The elements of a sorted or ordered array are arranged in ascending (or descending) order. Duplicates are always adjacent in a sorted array. For example, in the array {1,3,5,5,7,7,9}, 5 and 7 are duplicate elements. Let us understand the different methods of removal of duplicates in sorted array.

Method 1: Using Temporary Array

Approach:

1. **Create a Temporary Array** : Start by creating a temporary array temp[] to store unique elements from the original array arr[].
2. **Traverse the Input Array** : Go through each element in arr[] and, one by one, copy the unique elements to temp[]. Simultaneously, keep track of the count of unique elements using a variable j.
3. **Copy Back and Print:** After the traversal, copy the first j elements from temp[] back to arr[], and then print the array.

Detailed Steps:

1. **Initialize Index Variables:**
   - Use an index variable 'i' to iterate through each element in 'arr[]'.
   - Use another index variable 'j' to track the position in 'temp[]' where the next unique element will be stored. Initialize 'j = 0' since 'temp[]' is initially empty.
2. **Check for Uniqueness:**
   - Compare each element 'arr[i]' with the next element 'arr[i+1]'.
   - If 'arr[i]' is not equal to 'arr[i+1]', it means 'arr[i]' is unique, so store 'arr[i]' in 'temp[j]' and increment both 'i' and 'j' by 1.
   - If 'arr[i]' is equal to 'arr[i+1]', increment 'i' by 1 to skip the duplicate element.

3. **Handle the Last Element:**
   - The loop runs from the first element to the second-last element of 'arr[]'. The last element is not included in the loop because there is no 'arr[i+1]' for the last element. After the loop, the last element (if unique) should be handled separately by copying it directly to 'temp[j]'.
4. **Copy Back to Original Array:**
   - After processing, copy the first 'j' elements from 'temp[]' back to 'arr[]'.
5. **Output the Result:**
   - Finally, print the array 'arr[]', which now contains only unique elements.

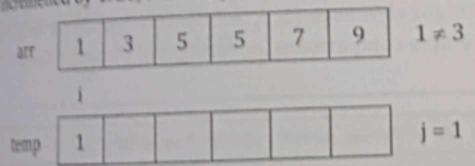
Example:

Original Array : arr[] ={1,3,5,5,7,9}.

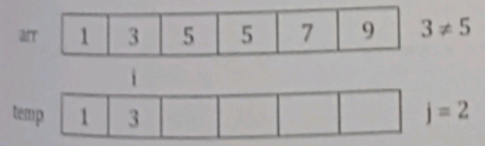
Temporary Array: temp[] (Initially empty)

Index Variables: i (to traverse arr[]) and j (to track the position in temp[] where the next unique element will be stored). Initially i=0 and j=0

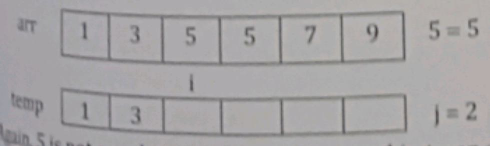
In the first iteration of the for loop, 'i' will point to the index of the first element (1). Then we will check if the ith element is equal to the (i+1)th element. 1 is not equal to 3, so the ith element (1) will be stored in another array temp and i will be incremented by 1 so that it points to the index of the second element (3). 'j' will be incremented by 1. So, i=1 and j=1



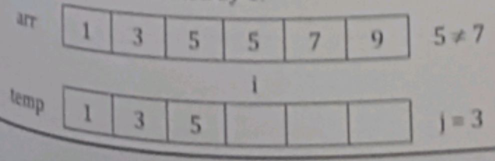
Again, we will check whether 3 is equal to 5. It is not equal, so 3 will be stored in temp and 'i' will be incremented by 1 so that it points to the index of the third element (5). 'j' will be incremented by 1.



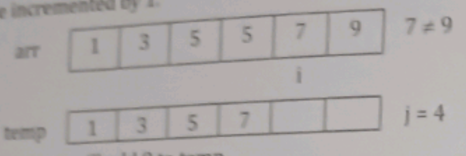
This time, the ith element (5) is equal to the (i+1)th element (5). So, the ith element will not be stored in temp and 'i' will be incremented so that it points to the index of the next 5.



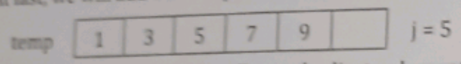
Again, 5 is not equal to 7. So, 5 will be stored in temp and 'i' will be incremented so that it points to the index of 7. 'j' will be incremented by 1.



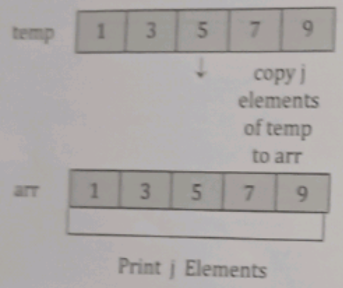
7 is not equal to 9. So, 7 will be stored in temp and i will be incremented by 1 to point to the index of 9. 'j' will be incremented by 1.



At last, we will add 9 to temp.



So, the array temp contains all non-duplicate elements of arr. Copy the 'j' elements of temp to arr and print them.



After processing, the array arr[] now contains only the unique elements: {1, 3, 5, 7, 9}. This approach effectively removes duplicates by utilizing a temporary array and simple comparisons, ensuring that the original order is preserved while eliminating any repeated values.

Algorithm To Remove Duplicate Elements from an Ordered Array using Temporary Array

Let arr[0...n-1] be the sorted array, where n ≥ 1.

```
Step 1: Start
Step 2: Declare variables arr[], temp[], n, i and j.
Step 3: Read n elements of an array and store the elements in array arr.
Step 4: Initialize i=0 and j=0
Step 5: for i = 0 to n-2  do
            (a) if ith element is not equal to (i+1)th element of arr then
                temp[j] = arr[i];
            (b) j = j + 1
Step 6: Store last value of arr in temp.
        temp[j] = arr[n-1]
        j = j + 1
Step 7: Copy elements from temp to arr
        for i = 0 to j-1  do
            arr[i] = temp[i];
Step 8: Print final array arr from i = 0 to j-1
Step 9: End
```

C program to Remo

```
<stdio.h>
removeDuplicates(int arr[]
nt i, j = 0;
nt temp[n];
or (i = 0; i < n-1; i++)

  // if ith element is
  if (arr[i] != arr[i+1
  {
     temp[j] = arr[i]
     j++;
  }
}
// store last value of
temp[j++] = arr[n-1];

// make array arr equa
for (i = 0; i < j; i++
    arr[i] = temp[i]

// print final array a
printf("\n Array Elem
for (i = 0; i < j; i+
    printf("%d ", ar
}

void main()
{
  int n,i;

  printf(" Enter Num
  scanf("%d",&n);
  int arr[n];
  printf("\n Enter
  for(i=0; i<n; i++
     scanf("%d", &ar

  removeDuplicates(
}
```

Output

```
Enter Number of Ele
Enter 10 Elements i
Array Elements afte
```

Program 5 C program to Remove the Duplicates from an Ordered Array.

```
#include <stdio.h>

void removeDuplicates(int arr[], int n)
{
    int i, j = 0;
    int temp[n];

    for (i = 0; i < n-1; i++)
    {
        // if ith element is not equal to (i+1)th element of arr, then store ith value in temp
        if (arr[i] != arr[i+1])
        {
            temp[j] = arr[i];
            j++;
        }
    }
    // store last value of arr in temp
    temp[j++] = arr[n-1];

    // make array arr equal to temp
    for (i = 0; i < j; i++)
        arr[i] = temp[i];

    // print final array arr
    printf("\n Array Elements after Removal of Duplicates : ");
    for (i = 0; i < j; i++)
        printf("%d ", arr[i]);
}

void main()
{
    int n,i;

    printf(" Enter Number of Elements : ");
    scanf("%d",&n);
    int arr[n];
    printf("\n Enter %d Elements in Sorted Order: ", n);
    for(i=0; i<n; i++)
        scanf("%d", &arr[i]);

    removeDuplicates(arr, n);
}
```

Output

```
Enter Number of Elements : 10
Enter 10 Elements in Sorted Order: 10 20 20 30 40 50 50 50 80 80
Array Elements after Removal of Duplicates : 10 20 30 40 50 80
```

Note: The above program will work only for sorted array so while providing an input values make sure that the given array is in sorted order, otherwise it will give unexpected results.

Method 2: Without Using Temporary Array

This method is the same as the previous one, but we will not make a separate array (temp). Instead, we will store our final resultant elements in the same array (arr) instead of temp.

Example:

Let us consider the below sorted array **arr** of 6 elements. n = 6

Initially i =0 and j =0

0	1	2	3	4	5
11	21	21	34	34	50
i, j					

In the first iteration of the for loop, i will point to the index of the first element (1). Then we will check if the ith element is equal to the (i+1)th element. if ith element is not equal to (i+1)th element of arr, then store ith value in arr[j]

Step 1: The element at 1st position (a[i] =11) is compared with element at 2nd position (a[i+1] =21). Elements compared are different (11, 21). 11 is a unique element placed at 1st position.

arr[j]=arr[i]
arr[j] = arr[0]
arr[j] = 11
arr[0] = 11
increment i by1 and j by 1
So, i =1 and j =1

0	1	2	3	4	5
11	21	21	34	34	50
	i, j				

Step 2: The element at 2nd position (a[i] =21) is compared with element at 3rd position (a[i+1] =21). Elements compared are same (21, 21).

If it is same, then just increment i by 1.
i=2
No change in 'j' value. So, j=1

0	1	2	3	4	5
11	21	21	34	34	50
	j	i			

Step 3: The element at 3rd position (a[i] =21) is compared with element at 4th position (a[i+1] =34). Elements compared are different (21, 34). 21 is a unique element placed at 2nd position.

arr[j]=arr[i]
arr[j] = arr[2]
arr[j] = 21

arr[1]= 21

increment i by1 and j by 1

So, i =3 and j =2

0	1	2	3	4	5
11	**21**	21	34	34	50
		j	i		

Step 4: The element at 4th position (a[i] =34) is compared with element at 5th position (a[i+1] =34).

Elements compared are same (34, 34).

If it is same, then just increment i by 1.

i=4

No change in 'j' value. So, j=2

0	1	2	3	4	5
11	**21**	21	34	34	50
		j		i	

Step 5: The element at 5th position (a[i] =34) is compared with element at 6th position (a[i+1] =50). Elements compared are different (34, 50). 34 is a unique element placed at 3rd position.

arr[j]=arr[i]

arr[j] = arr[4]

arr[j] = 34

arr[2] = 34

increment i by1 and j by 1

So, i =5 and j =3

0	1	2	3	4	5
11	**21**	**34**	34	34	50
			j		i

Since i value is n-1, we can't compare with a[i+1]. So, we stop iterating the loop.

Finally store the last element of an array in arr[j]

arr[j]= arr[n-1] or arr[i]

arr[3] = arr[5]

arr[3]= 50

Increment j by 1, so j = 4.

0	1	2	3	4	5
11	**21**	**34**	**50**	34	50
				j	

Now, print first j elements of array arr.

arr[0] = 11 arr[1]= 21 arr[2] = 34 and arr[3]=50

Note that, length of an array should be considered as 'j'. If we iterate the elements from 0 to j-1, we get the unique elements.

Algorithm **To Remove Duplicate Elements from an Ordered Array without using Extra Array**

```
Let arr[0...n-1] be the sorted array, where n ≥ 1.
Step 1:   Start
Step 2:   Declare variables arr[], n, i and j.
Step 3:   Read n elements of an array and store the elements in array arr.
Step 4:   Initialize i=0 and j=0
Step 5:   for i = 0 to n-2  do
            (a) if ith element is not equal to (i+1)th element of arr then
                arr[j] = arr[i];
            (b) j = j + 1
Step 6:   Store last value of arr in temp.
          arr[j] = arr[n-1]
          j = j + 1
Step 7:   Print final array arr from i = 0 to j-1
Step 8:   End
```

Program 6 **C Program to Remove Duplicates from an Ordered Array without using Extra Array.**

```c
#include <stdio.h>

void removeDuplicates(int arr[], int n)
{
    int i, j = 0;

    for (i = 0; i < n-1; i++)
    {
        // if ith element is not equal to (i+1)th element of arr, then store ith value in temp
        if (arr[i] != arr[i+1])
        {
            arr[j] = arr[i];
            j++;
        }
    }

    // store last value of arr in temp
    arr[j++] = arr[n-1];

    // print final array arr
    printf("\n Array Elements after removal of duplicates : ");
    for (i = 0; i < j; i++)
        printf("%d ", arr[i]);
}

void main()
{
```

```
int n,i;
printf(" Enter Number of Elements : ");
scanf("%d",&n);
int arr[n];
printf("\n Enter %d elements in Sorted Order: ", n);
for(i=0; i<n; i++)
  scanf("%d", &arr[i]);

removeDuplicates(arr, n);
```

Output :

```
Enter Number of Elements : 8
Enter 8 elements in Sorted Order: 10 20 20 30 40 40 50 60
Array Elements after removal of duplicates : 10 20 30 40 50 60
```

14.6 Partitioning an Array

Problem

Partition the elements of a given unordered array of 'n' elements into two subsets so that elements less than or equal to 'x' are in one subset and elements greater than 'x' are in other subset.

In an unordered array, the elements are randomly ordered. Elements are not sorted in ascending or, descending order.

Example: { 38, 35, 32, 12, 15, 10, 23, 28, 4, 10}

While this problem could be easily solved by sorting the array in ascending order, sorting is a time-consuming process. Therefore, a more efficient method is needed to partition the array without fully sorting it.

Simple Partitioning Approach

Let's break down the process of partitioning an array using a simpler method. We'll use an example array: '{1, 4, 32, 23, 15, 12, 10, 28, 35, 38}'. Our goal is to rearrange the array so that all elements less than or equal to 20 are on the left side, while all elements greater than 20 are on the right side.

Consider the unordered array of 10 elements.

```
     a[1]                                              a[10]
  {   1,    4,    32,    23,    15,    12,    10,    28,    35,    38     }
```

Let 20 be the value of partition key (x). The partitioned array with 20 as partition key is given as below.

```
                                         x=20
  {   1,    4,    10    12    15,    ↓   23,    32,    28,    35    38     }
     ←─────────── ≤ 20 ──────────→  ←─────────── > 20 ───────────→
```

By comparing the above two data sets, we can see, elements from left side which are greater than 20 must be moved to right side and elements from right side which are less than or equal to 20 must be moved to left side.

Steps to Achieve Partitioning an Array

We need to partition an array into two subsets to ensure that all elements less than or equal to a given partition value 'x' are on one side, while all elements greater than 'x' are on the other. The core idea of this approach is to use two index variables that move inward from opposite ends of the array, swapping elements that are out of place as they go.

Left partition growing to the right →		← Right partition growing to the left

Step 1: Initialize Index Variables

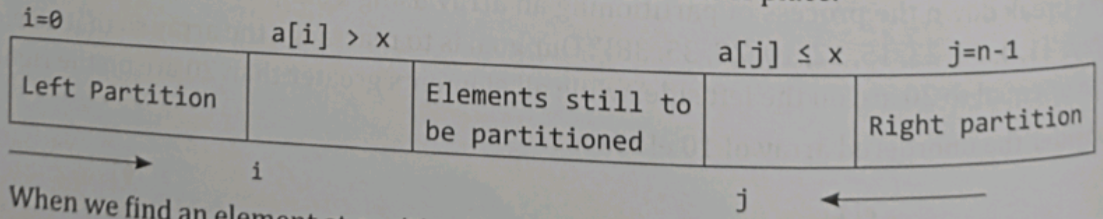
- **Left Index ('i') :** Start at the beginning of the array ('i = 0').
- **Right Index ('j') :** Start at the end of the array ('j = n-1', where 'n' is the total number of elements in the array).

Step 2: Inward Movement of Index Variables

- **Left Index Movement ('i'):**
  - The left index ('i') moves from left to right through the array.
  - It continues moving right as long as the current element is less than or equal to the partition value 'x'. When it encounters an element greater than 'x', it stops.
- **Right Index Movement ('j') :**
  - The right index ('j') moves from right to left.
  - It continues moving left as long as the current element is greater than the partition value 'x'. When it encounters an element less than or equal to 'x', it stops.

Step 3: Swap Elements

- **When Both Index Variables Stop:**
  - If the left index ('i') finds an element greater than 'x' and the right index ('j') finds an element less than or equal to 'x', these elements are out of place.



  - When we find an element at position 'i' where a[i] > x , that need to be moved to right and an element at 'j' where `a[j] ≤ x` that must be moved to left partition, i^{th} and j^{th} elements are exchanged at this point.
  - Exchange is done using swapping technique. Swap the elements at positions 'i' and 'j'.
  - After swapping, increment the left index ('i') and decrement the right index ('j') to continue the process.

Step 4: Repeat Until Index Variables Cross

- **Continue the Process:**
  - The index variables continue moving inward, swapping elements as needed, until they cross each other (i >= j).
  - Once the index variables cross, the partitioning is complete. All elements less than or equal to 'x' will be on the left side, and all elements greater than 'x' will be on the right side.

Algorithm **Partitioning an array**

```
Step 1:  Start
Step 2:  Set array a[0..n-1] and partitioning point value x.
Step 3:  initialize i=0 and j=n-1
Step 4:  While (i<j) do
           (a) Move left and right partition closer each other, until an incorrectly positioned
               pair is found.
               while (i < j and a[i] ≤ x) do i = i + 1
               while (i < j and a[j] > x) do j = j - 1
           (b) swap any pairs that are incorrectly positioned and extend the partitions
               inwards by one.
               temp=a[i]
               a[i]=a[j]
               a[j]=temp
           (c) i = i + 1
           (d) j = j - 1
Step 5:  Print all the elements of an array a.
```

Example

Consider an array of 10 elements which has to partitioned

```
      a[1]                                               a[10]
   {   1,    4,   32,   23,   15,   12,   10,   28,   35,   38     }
```

Step 1: Here n = 10 and initial value of i = 0 and j = n-1 = 9 and partitioning index x = 20

a[i] = a[0] = 1

a[j] = a[n-1] = a[9] = 38

a[i]									a[j]
0	1	2	3	4	5	6	7	8	9
1	4	32	23	15	12	10	28	35	38

Step 2: Since i < j is true (0 < 9 is true), lets proceed with moving inwards

While (i < j and a[i] ≤ x)do i = i + 1

0 < 9 and 1 ≤ 20 is true , so i = 0 + 1 = 1

1 < 9 and 4 ≤ 20 is true, so i = 1 + 1 = 2

2 < 9 and 32 ≤ 20 is false, so stop iterating loop.

i = 2

While (i < j and a[j] > x) do j = j - 1

2 < 9 and 38 > 20 is true, so j = 9 – 1 = 8

2 < 8 and 35 > 20 is true, so j = 8 - 1 = 7

2 < 7 and 28 > 20 is true, so j = 7 - 1 = 6

2 < 6 and 10 > 20 is false, so stop iterating loop

j = 6

Now, swap a[i] and a[j] i.e., a[2] and a[6] - swap 32 and 10

		i				j			
0	1	2	3	4	5	6	7	8	9
1	4	10	23	15	12	32	28	35	38

increment i by1 and decrement j by 1

i= 2 + 1 = 3

j = 6 - 1 = 5

Step 3: Since i<j is true (3 < 5 is true), lets proceed with moving inwards

While (i < j and a[i] ≤ x)do i = i + 1

3 < 5 and 23 ≤ 20 is false , so stop iterating loop

No change in i value, i = 3

While (i < j and a[j] > x) do j = j - 1

3 < 5 and 12 > 20 is false, so stop iterating loop

No change in j value, j = 5

Now, swap a[i] and a[j] i.e., a[3] and a[5] - swap 23 and 12

			i		j				
0	1	2	3	4	5	6	7	8	9
1	4	10	12	15	23	32	28	35	38

Increment i by1 and decrement j by 1

i= 3 + 1 = 4

j = 5 - 1 = 4

Step 4: Since i<j is false (4 < 4 is false), lets stop moving inwards.

Step 5: The final array after partition is shown below.

0	1	2	3	4	5	6	7	8	9
1	4	10	12	15	23	32	28	35	38

Program 7 **C program for partitioning an array**

```
#include <stdio.h>

void partition(int arr[], int n, int x)
{
    int j = n-1, i = 0, temp , k;

    while(i<j)
    {
    while(i < j && arr[i] <= x)
        i = i + 1;
      while(i < j && arr[j] > x)
        j = j - 1;

      temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;

    i = i + 1;
    j = j - 1;
    }
    printf("\n Complete Array After Partition : ");
    for(i = 0; i < n; i++)
      printf(" %d ",arr[i]);
    printf("\n Left Array Elements <= %d : ",x);
    for(i = 0; arr[i]<= x; i++)
      printf(" %d ",arr[i]);
    printf("\n Right Array Elements > %d: ",x);
    for(k = i; k < n; k++)
      printf(" %d ",arr[k]);
}

void main()
{
  int i, n, x;
  printf("\n Enter Number of Array Elements :");
  scanf("%d",&n);
  int arr[n];

  printf("Enter %d Elements:",n);
  for(i = 0; i < n; i++)
    scanf("%d",&arr[i]);
  printf("\n Enter the Partition Element x: ");
  scanf("%d",&x);

  partition(arr,n,x);
}
```

Output 1

```
Enter Number of Array Elements :10
Enter 10 Elements:28 26 25 11 16 12 24 29 6 10

Enter the Partition Element x: 17

Complete Array After Partition :  10  6  12  11  16  25  24  29  26  28
Left Array Elements <= 17 :  10  6  12  11  16
Right Array Elements > 17:  25  24  29  26  28
```

Output 2

```
Enter Number of Array Elements :10
Enter 10 Elements:100 90 80 70 60 50 40 30 20 10

Enter the Partition Element x: 50

Complete Array After Partition :  10  20  30  40  50  60  70  80  90  100
Left Array Elements <= 50 :  10  20  30  40  50
Right Array Elements > 50:  60  70  80  90  100
```

14.7 Finding the Kth Smallest Element

Problem

Find the kth smallest element in a randomly ordered array of n elements.

Example

K = 3

7	2	6	12	3	9	5	11

5 is the 3rd smallest element

Given an array A[] of n elements and a positive integer K, we need to find the Kth smallest element in the array. It is given that all array elements are distinct.

Example

Consider an array {8, 3, 7, 13, 4, 10, 6, 12}. Find 3rd Smallest Element

K=1 => 1st Smallest Element = 3

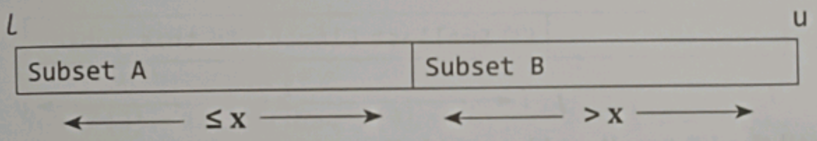
K=2 => 2nd Smallest Element = 4

K=3 => 3rd Smallest Element = 6

Approach: Partitioning-Based Method to Find the Kth Smallest Element

The task is to find the Kth smallest element in an unsorted array. This problem can be approached in several ways. One of the simplest methods is to sort the elements and then select the Kth element from the sorted array. However, a more efficient solution can be achieved using a partitioning strategy, similar to the approach discussed in the previous section. In the partitioning problem, the value of the pivot (x) is known. However, in this case, the value of the pivot (which corresponds to the Kth smallest element) is not known in advance. A highly efficient method to achieve this is by using a partitioning-based approach, similar to the Quickselect algorithm. This method is particularly effective because it focuses on finding the Kth smallest element directly, rather than sorting the entire array.

The approach involves repeatedly partitioning the array around a chosen pivot element. The partitioning process divides the array into two parts: one part contains elements less than or equal to the pivot, and the other part contains elements greater than the pivot. The search for the Kth smallest element is then narrowed down to the relevant part of the array, based on the position of the Kth element relative to the pivot. This process continues until the Kth smallest element is identified.



Step 1: Initialize Boundaries

- **Lower Bound ('l') :** Set 'l = 0', starting at the beginning of the array.
- **Upper Bound ('u') :** Set 'u = n-1', starting at the end of the array, where 'n' is the length of the array.

Step 2: Choose a Pivot Element

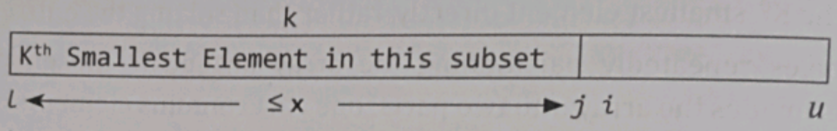
- Select the pivot element 'x' from the array. A common method is to choose the element at the position 'k' ('k-1' for zero-based indexing). We can also use pivot as middle element or any other element. So, x = a[k-1]

Step 3: Partition the Array Around the Pivot

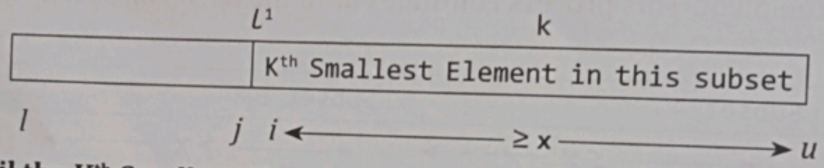
- The array is partitioned into two parts:
  - **Left Partition :** Contains all elements less than or equal to the pivot (x)
  - **Right Partition :** Contains all elements greater than the pivot (x)
- **Partitioning Process:**
  - Two index variables, 'i' and 'j', are used.
  - 'i' starts from the left bound 'l'.
  - 'j' starts from the right bound 'u'.
  - Increment 'i' until an element greater than the pivot is found.
  - Decrement 'j' until an element less than or equal to the pivot is found.
  - If 'i' is still less than 'j', swap the elements at 'i' and 'j'. Continue moving 'i' right and 'j' left until they cross each other.

Step 4: Adjust the Bounds Based on K^th^ Element's Position

- **After partitioning** , the kth smallest value could be in left partition (or) right partition.For example, If kth smallest value is in right parition then left partition is discarded and search for kth smallest value in right parition. Replace lower bound value and start searching for kth smallest value again in smaller subset of right partition. Repeat the partitioning process to smaller subsets to find kth smallest value.
  - If the Kth smallest element is in the left partition (where elements are ≤ pivot), update the lower bound 'l' to 'i'.



  - If the Kth smallest element is in the right partition (where elements are > pivot), update the upper bound 'u' to 'j'.



Step 5: Repeat Until the Kth Smallest Element is Found

- Continue the partitioning process within the updated bounds ('l' and 'u') until the left and right bounds overlap, isolating the Kth smallest element at position 'k'. Once isolated, return the Kth smallest element.

This optimized approach efficiently narrows down the search space to find the Kth smallest element without requiring the entire array to be sorted, making it a fast and effective method.

Example **Find the 3rd Smallest Element in the Array**

Let's go through a detailed, step-by-step process for finding the 3rd smallest element using the partitioning approach with the array 'A = [5, 2, 9, 1, 6]' and 'K = 3'.

Initial Setup

Array: A = [5, 2, 9, 1, 6]

K = 3 (we want to find the 3rd smallest element)

Convert 'K' to zero-indexed: K = 3 - 1 = 2

Step 1: Initialize Boundaries

- **Lower Bound ('l'):** Start at the beginning of the array (l = 0).
- **Upper Bound ('u'):** Start at the end of the array (u = 4), since the array has 5 elements.

Step 2: Choose a Pivot Element

- Select the pivot element from the array. We'll initially choose the element at position 'K', so:
- **Pivot = A[K] = A[2] = 9**

Step 3: Partition the Array Around the Pivot

- The goal is to rearrange the array such that, the elements less than or equal to 'pivot (9)' will move to the left. The elements greater than 'pivot (9)' will move to the right.

Initial array: '[5, 2, 9, 1, 6]'

Partitioning Process:

1. **Index Variables:**

   'i = l = 0'

   'j = u = 4'

2. **Increment 'i'** until an element greater than the 'pivot' is found:
   - 'i = 0': 'A[i] = 5' (less than pivot, so continue incrementing).
   - 'i = 1': 'A[i] = 2' (less than pivot, so continue incrementing).
   - 'i = 2': 'A[i] = 9' (equal to pivot, stop here).
3. **Decrement 'j'** until an element less than or equal to 'pivot' is found:
   - 'j = 4': 'A[j] = 6' (less than pivot, so stop here).
4. **Swap 'A[i]' and 'A[j]'** since 'i < j':
   - Swap 'A[2]' and 'A[4]': The array becomes '[5, 2, 6, 1, 9]'.
   - Update the index variables:
     - Increment 'i = 3'
     - Decrement 'j = 3'
5. **Now 'i' (3) is not less than 'j' (3),** so the partitioning stops here.

Step 4: Post-Partition Analysis

- After the partitioning, the array is '[5, 2, 6, 1, 9]'.
- The pivot element '9' is now at the end of the array, and we need to look for the 3rd smallest element, which is within the left partition '[5, 2, 6, 1]'.
- Since j < K (where j = 3 and K = 2), the 3rd smallest element must be in the left partition [5, 2, 6, 1].

Step 5: Focus on the Left Partition '[5, 2, 6, 1]'

Now, focus on the left partition with the new bounds:

Lower Bound ('l'): 'l = 0'

Upper Bound ('u'): 'u = 3'

1. **Choose a New Pivot:**
   - Select the element at position 'K' as the pivot again. Since the subarray is '[5, 2, 6, 1]':
   - Pivot = A[K] = A[2] = 6
2. **Partition Again:**
   - **Increment 'i':**
     - 'i = 0': 'A[i] = 5' (less than pivot, continue incrementing).
     - 'i = 1': 'A[i] = 2' (less than pivot, continue incrementing).
     - 'i = 2': 'A[i] = 6' (equal to pivot, stop here).
   - **Decrement 'j':**
     - 'j = 3': 'A[j] = 1' (less than pivot, stop here).

- **Swap 'A[i]' and 'A[j]':**
  - Swap 'A[2]' and 'A[3]': The array becomes '[5, 2, 1, 6, 9]'.
- **Update the index variables:**
  - Increment 'i = 3'
  - Decrement 'j = 2'
- **Now 'i' (3) is not less than 'j' (2),** so the partitioning stops here.

Step 6: Determine the 3rd Smallest Element

After the initial partitioning process, the array is '[5, 2, 1, 6, 9]'. The pivot element '6' is correctly placed, and we need to focus on the left partition '[5, 2, 1]' to find the 3rd smallest element.

- Since j < K (where j = 2 and K = 2), the 3rd smallest element is within this partition.
- The pivot '6' is now correctly positioned. The elements to the left are '[5, 2, 1]', which contain the smallest elements.

Let's go through the process again, focusing on correctly finding the 3rd smallest element by continuing the partitioning process on the subarray '[5, 2, 1]'.

Step 7: Repeat the Partitioning Process on '[5, 2, 1]'

Now, we consider the subarray '[5, 2, 1]' with new bounds:

- **Lower Bound ('l'):** 'l = 0'
- **Upper Bound ('u'):** 'u = 2'

New Partitioning Process:

1. **Choose a New Pivot:**
   - Since we're focusing on the subarray '[5, 2, 1]', we choose the element at position 'K' (which is still 'K = 2' from the original array):
   - Pivot = A[K] = 1
2. **Partition the Subarray:**
   - **Initial Array:** '[5, 2, 1]'
   - **Index Variables:**
   - 'i = 0'
   - 'j = 2'
3. **Increment 'i'** until an element greater than the 'pivot' is found:
   - 'i = 0': 'A[i] = 5' (greater than pivot, stop here).
4. **Decrement 'j' until an element less than or equal to 'pivot' is found:**
   - 'j = 2': 'A[j] = 1' (equal to pivot, continue decrementing).
   - 'j = 1': 'A[j] = 2' (greater than pivot, continue decrementing).
   - 'j = 0': 'A[j] = 5' (greater than pivot, continue decrementing).
5. **Swap 'A[i]' and 'A[j]':**
   - Since 'i = 0' and 'j = 2', swap 'A[0]' and 'A[2]': The array becomes '[1, 2, 5]'.

6. **Update the index variables:**
   - Increment 'i = 1'
   - Decrement 'j = 1'

Now, 'i = 1' is not less than 'j = 1', so the partitioning stops here.

Step 8: Post-Partition Analysis

After this partitioning, the array is '[1, 2, 5]'. This partition is now sorted, and we can determine the 3rd smallest element.

Final Result

The array '[1, 2, 5, 6, 9]' is now fully sorted, and the 3rd smallest element is '5'.

Algorithm To Find the kth Smallest Element using Array Partitioning

```
Step 1:  Start
Step 2:  Read array of n elements and kth element to be found.
Step 3:  Set l=0, u=n-1, k=k-1
Step 4:  While left and right partition does not overlap (l < u) do
           (i) x:= a[k]
          (ii) Set l:= i , upper limit of left partition
               u:= j, lower limit of right partition
         (iii) While i > k and j < k do
               (a) extend left partition while elements less than x
                   (i.e, while a[i] < x do i = i + 1)
               (b) extend right partition while elements greater than or equal to x
                   (i.e, while x < a[j] do j = j - 1
          (iv) If kth smallest element is found in left partition, update upper limit u of left
               partition. i.e, if j < k then l:= i
           (v) if kth smallest element is found in right partition ,update upper limit l of
               right partition  i.e, if i < k then u:= j
Step 5:  Return the kth smallest element in position k
Step 6:  End
```

Program 8 C Program to find the kth smallest element

```c
#include<stdio.h>

void main()
{

    int i,x,temp,n,k,j,l,u;

    printf("\n Enter Number of Array Elements : ");
    scanf("%d",&n);
    int arr[n];
    printf("\n Enter %d Elements : ",n);
    for(i = 0; i < n; i++)
       scanf("%d",&arr[i]);
    printf("\n Enter the Value of K : ");
    scanf("%d",&k);
    k=k-1;

    l=0;
    u=n-1;
    while(l<u)

    {
       i=l;
       j=u;
       x=arr[k];

       while(i<=k && j>=k)
       {
          while(arr[i]<x)
             i=i+1;
          while(arr[j]>x)
             j=j-1;

          temp=arr[i];
          arr[i]=arr[j];
          arr[j]=temp;

          i=i+1;
          j=j-1;
       }

       if(j<k)
          l=i;
       if(i>k)
          u=j;
    }
    printf("\n Complete Array After Partition : ");
    for(i = 0; i < n; i++)
       printf(" %d ",arr[i]);
    printf("\n Kth Smallest Element is %d at Position %d \n",arr[k],k+1);
}
```

Output 1

```
Enter Number of Array Elements : 10
Enter 10 Elements : 90 10 30 20 70 60 40 50 100 80
Enter the Value of K : 4
Complete Array After Partition :  20  10  30  40  70  60  50  80  100  90
Kth Smallest Element is 40 at Position 4
```

Output 2

```
Enter Number of Array Elements :10
Enter 10 Elements : 100 90 80 70 60 50 40 30 20 10
Enter the Value of K : 8
Complete Array After Partition :  10  20  30  70  60  50  40  80  90  100
Kth Smallest Element is 80 at Position 8
```

14.8 Multiplication of Two Matrices

Problem

If A is a matrix of dimension m × n, and B is a matrix of dimension n x p, we need to find the product AB of dimension m × p.

A matrix is a rectangular array or table of numbers, symbols, or expressions, arranged in rows and columns in mathematics. We can perform various operations on matrices such as addition, subtraction, multiplication and so on.

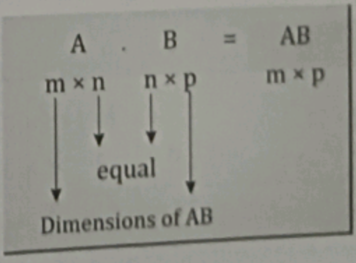
Matrix multiplication, also known as matrix product and the multiplication of two matrices, produces a single matrix. It is a type of binary operation. If A and B are the two matrices, then the product of the two matrices A and B are denoted by:

$$X = AB$$

Suppose we have two matrices A and B, the multiplication of matrix A with Matrix B can be given as (AB). That means, the resultant matrix for the multiplication of for any m × n matrix 'A' with an n × p matrix 'B', the result can be given as matrix 'C' of the order m × p.

Rule for Matrix Multiplication:

Two matrices A and B are said to be compatible if the number of columns in A is equal to the number of rows in B. That means if A is a matrix of order m×n and B is a matrix of order n×p, then we can say that matrices A and B are compatible.



For example,

(a) Multiplying a 4 × 3 matrix by a 3 × 4 matrix is valid and it gives a matrix of order 4 × 4

(b) 7 × 1 matrix and 1 × 2 matrices are compatible; the product gives a 7 × 2 matrix.

(c) Multiplication of a 4 × 3 matrix and 2 × 3 matrix is NOT possible.

Matrix Multiplication Formula

We can understand the general process of matrix multiplication by the technique, "First rows are multiplied by columns (element by element) and then the rows are filled up." Consider two matrices of order 3×3 as given below,

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}\begin{bmatrix} j & k & l \\ m & n & o \\ p & q & r \end{bmatrix} = \begin{bmatrix} (aj+bm+cp) & (ak+bn+cq) & (al+bo+cr) \\ (dj+em+fp) & (dk+en+fq) & (dl+eo+fr) \\ (gj+hm+ip) & (gk+hn+iq) & (gl+ho+ir) \end{bmatrix}$$

? How to Multiply Matrices?

The steps in matrix multiplication:

1. Make sure that the number of columns in the 1st matrix equals the number of rows in the 2nd matrix (compatibility of matrices).
2. Multiply the elements of each row of the first matrix by the elements of each column in the second matrix.
3. Add the products.
4. Place the added products in the respective columns.

Example 1

$$A = \begin{bmatrix} 3 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix}_{2\times3} \text{ and } B = \begin{bmatrix} 1 & 2 \\ 0 & 5 \\ 3 & -4 \end{bmatrix}_{3\times2}$$

$$AB = \begin{bmatrix} 3 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix} \downarrow \begin{bmatrix} 1 & 2 \\ 0 & 5 \\ 3 & -4 \end{bmatrix} = \begin{bmatrix} 3\times1+1\times0+2\times3 & 3\times2+1\times5+2\times-4 \\ 1\times1+2\times0+1\times3 & 1\times2+2\times5+1\times-4 \end{bmatrix} = \begin{bmatrix} 3+0+6 & 6+5-8 \\ 1+0+3 & 2+10-4 \end{bmatrix}$$

$$AB = \begin{bmatrix} 9 & 3 \\ 4 & 8 \end{bmatrix}_{2\times2}$$

Example 2

$$A = \begin{bmatrix} 2 & 1 \\ 6 & -3 \end{bmatrix}_{2\times2} \text{ and } B = \begin{bmatrix} 1 & -2 \\ -3 & 5 \\ 2 & 4 \end{bmatrix}_{3\times2}$$

AB is not possible as number of column of A matrix is not equal to number of rows in B matrix

Approach to Find the Multiplication of Two Matrices

- Define three matrices A, B, and C.
- Read row and column numbers of Matrix A and store it in variable m and n

- Read row and column numbers of Matrix B and store it in variable p and q
- Check if the matrix can be multiplied or not, if n is not equal to p then matrices can't be multiplied and hence exit.
- Read elements of matrices A and B using nested loop.
- Multiply A and B using nested loops.
    - First Loop: A loop which goes up to m giving row elements of A, (i = 0 to m-1)
    - Second Loop : A loop which goes up to q giving column elements of B. (j = 0 to q-1)
    - At last, we define a loop which goes up to p giving row element of B. (k= 0 to p-1)
- Then, we store their corresponding multiplication by C[i][j]= C[i][j] + A[i][k] * B[k][j], which gets updated each time till k reaches p, which acts as the mathematical formula of multiplication used for matrix.

Example

Let us consider the below two matrices.

$$A = \begin{bmatrix} 3 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix}_{2\times 3} \text{ and } B = \begin{bmatrix} 1 & 2 \\ 0 & 5 \\ 3 & -4 \end{bmatrix}_{3\times 2}$$

Here, m=2, n=3, p=3 and q=2

Since n=p, multiplication of A and B is possible. The resultant matrix C will be of m * q matrix

Loop 1
i= 0 to m-1
i= 0 to 1
	Loop 2
	j= 0 to q-1
	j= 0 to 1
		Loop 3
		k = 0 to p-1
		k= 0 to 2
			C[i][j]= C[i][j]+A[i][k] * B[k][j]

i=0	j=0
	Initialize
	C[i][j]=0
	C[0][0]=0
		k=0	C[i][j]= C[i][j]+A[i][k] * B[k][j]
			C[0][0]= C[0][0]+A[0][0]* B[0][0]
			C[0][0]= 0+3* 1
			C[0][0]= 3

		k=1	C[i][j]= C[i][j]+A[i][k] * B[k][j]
			C[0][0]= C[0][0]+A[0][1]* B[1][0]
			C[0][0]= 3+1* 0
			C[0][0]= 3

		k=2	C[i][j]= C[i][j]+A[i][k] * B[k][j]
			C[0][0]= C[0][0]+A[0][2]* B[2][0]
			C[0][0]= 3+2* 3
			C[0][0]= 9

		k=0	C[i][j]= C[i][j]+A[i][k] * B[k][j]
			C[0][1]= C[0][1]+A[0][0]* B[0][1]
			C[0][1]= 0+3* 2
			C[0][1]= 6

	j = 1
	Initialize
	C[i][j]=0
	C[0][1]=0
		k=1	C[i][j]= C[i][j]+A[i][k] * B[k][j]
			C[0][1]= C[0][1]+A[0][1]* B[1][1]
			C[0][1]= 6+1* 5
			C[0][1]= 11

		k=2	C[i][j]= C[i][j]+A[i][k] * B[k][j]
			C[0][1]= C[0][1]+A[0][2]* B[2][1]
			C[0][1]= 11+2* -4
			C[0][1]= 3

i=1	j=0
	Initialize
	C[i][j]=0
	C[1][0]=0
		k=0	C[i][j]= C[i][j]+A[i][k] * B[k][j]
			C[1][0]= C[1][0]+A[1][0]* B[0][0]
			C[1][0]= 0+1* 1
			C[1][0]= 1

		k=1	C[i][j]= C[i][j]+A[i][k] * B[k][j]
			C[1][0]= C[1][0]+A[1][1]* B[1][0]
			C[1][0]= 1+2* 0
			C[1][0]= 1

		k=2	C[i][j]= C[i][j]+A[i][k] * B[k][j]
			C[1][0]= C[1][0]+A[1][2]* B[2][0]
			C[1][0]= 1+1* 3
			C[1][0]= 4

	j = 1
	Initialize
	C[i][j]=0
	C[1][1]=0
		k=0	C[i][j]= C[i][j]+A[i][k] * B[k][j]
			C[1][1]= C[1][1]+A[1][0]* B[0][1]
			C[1][1]= 0+1* 1
			C[1][1]= 1

		k=1	C[i][j]= C[i][j]+A[i][k] * B[k][j]
			C[1][1]= C[1][1]+A[1][1]* B[1][1]
			C[1][1]= 1+2* 5
			C[1][1]= 11

		k=2	C[i][j]= C[i][j]+A[i][k] * B[k][j]
			C[1][1]= C[1][1]+A[1][2]* B[2][1]
			C[1][1]= 11+1* -4
			C[1][1]= 8

$$\text{Resultant Matrix} C = \begin{bmatrix} C[0][0] & C[0][1] \\ C[1][0] & C[1][1] \end{bmatrix}$$

$$\text{The resultant matrix } C = AB = \begin{bmatrix} 9 & 3 \\ 4 & 8 \end{bmatrix}_{2\times2}$$

Algorithm To Find Multiplication of Two Matrices

```
Step 1: Start
Step 2: Declare three matrices A, B and C
Step 3: Declare variables m, n, p, q, i, j ,k
Step 4: Read row and column of Matrix A and store it in variables m and n.
Step 5: Read row and column of Matrix B and store it in variables p and q.
Step 6: if n is not equal to p then multiplication is not possible and hence exit.
Step 7: Read elements of Matrix A and B
Step 8: for i = 0 to m-1 do
            for j = 0 to q-1 do
               C[i][j]=0
               for k=0 to p-1 do
                  C[i][j]= C[i][j]+A[i][k] * B[k][j]
Step 9: Print the resultant matrix C
Step 7: End
```

Program 9 **Matrix Multiplication**

```
#include<stdio.h>

int main()
{

    int i,j,k,m,n,p,q;
    printf("\n Enter Number of rows and columns for Matrix A : ");
    scanf("%d %d",&m,&n);
    int A[m][n];

    printf("\n Enter Number of rows and columns for Matrix B : ");
    scanf("%d %d",&p,&q);
    int B[p][q];
    if(n!=p)
    {
       printf("\nMultiplication is not possible");
       return 0;
    }

    int C[m][q];
    for(i=0;i<m;i++)
       for(j=0;j<n;j++)
       {
          printf("Enter the Element [%d][%d] of A : ", i+1, j+1);
          scanf("%d", &A[i][j]) ;
       }

    for(i=0;i<p;i++)
       for (j=0;j<q;j++)
       {
          printf("Enter the element [%d][%d] of B : ", i+1, j+1);
          scanf("%d", &B[i][j]);
       }
    for(i=0;i<m;i++)
       for (j=0;j<q;j++)
       {
          C[i][j]=0;
          for (k=0;k<p;k++)
             C[i][j]=C[i][j] + A[i][k] * B[k][j];
       }
    printf("\n\t Product of the matrices is \n");
    for (i=0; i<m; i++)
    {
       for (j=0; j<q; j++)
          printf ("\t%d", C[i][j]);
       printf ("\n");
    }
    return 0;
}
```

Output

```
Enter Number of rows and columns for Matrix A : 2 3
Enter Number of rows and columns for Matrix B : 3 3
Enter the Element [1][1] of A : 2
Enter the Element [1][2] of A : 1
Enter the Element [1][3] of A : 4
Enter the Element [2][1] of A : 7
Enter the Element [2][2] of A : 3
Enter the Element [2][3] of A : 6
Enter the element [1][1] of B : 6
Enter the element [1][2] of B : 4
Enter the element [1][3] of B : 3
Enter the element [2][1] of B : 3
Enter the element [2][2] of B : 2
Enter the element [2][3] of B : 5
Enter the element [3][1] of B : 7
Enter the element [3][2] of B : 3
Enter the element [3][3] of B : 1
Product of the matrices is
    43      22      15
    93      52      42
```

14.9 Review Questions

1. Write an Algorithm to Reverse Elements of an Array.
2. Write a C Program to Reverse Elements of an Array.
3. What is Histogram?
4. What is Frequency Histogram?
5. Write an Algorithm to Display the Frequency Count of Each Element in an Array.
6. Given a set of n students' examination marks in the range 0 to 100. Write an Algorithm to find out the frequency count of the number of students that obtained each possible mark.
7. Given a set of n students' examination marks in the range 0 to 100. Write a C Program to find out the frequency count of the number of students that obtained each possible mark.
8. Write a C Program to print the frequency count of the number of students that obtained each possible mark in the Horizontal Frequency Histogram graph.
9. Write an Algorithm to Find the Maximum Number in an Array of 'n' Elements.
10. Write a C Program to Find the Maximum Number in an Array of 'n' Elements.

11. Write an Algorithm to Remove Duplicate Elements from an Ordered Array using Temporary Array.
12. Write a C Program to Remove Duplicate Elements from an Ordered Array using Temporary Array.
13. Write an Algorithm to Remove Duplicate Elements from an Ordered Array without using Temporary Array.
14. Write a C Program to Remove Duplicate Elements from an Ordered Array without using Temporary Array.
15. Write an Algorithm to Partition the elements of a given unordered array of n elements into two subsets so that elements less than or equal to x are in one subset and elements greater than x are in other subset.
16. Write a C Program to Partition the elements of a given unordered array of n elements into two subsets so that elements less than or equal to x are in one subset and elements greater than x are in other subset.
17. Write an Algorithm to Find the kth Smallest Element using Array Partitioning.
18. Write a C Program to Find the kth Smallest Element using Array Partitioning.
19. Write an Algorithm to Find the Multiplication of Two Matrices.
20. Write a C Program to Find the Multiplication of Two Matrices.