# 1
# C++ Program to Implement Variable Length Array

```
1. #include <iostream>
2. #include <string>
3. using namespace std;
4.
5. int main()
6. {
7.     int *array, size;
8.     cout<<"Enter size of array: ";
9.     cin>>size;
10.           array = new int [size];
11.           for (int i = 0; i < size; i++)
12.           {
13.               cout<<"Enter an integer to be inserted: ";
14.               cin>>array[i];
15.           }
16.           for(int i = 0; i < size; i++)
17.           {
18.               cout<<array[i]<<"  ";
19.           }
20.           cout<<endl;
21.           delete []array;
22.           return 0;
23.       }
```

RESULTS

```
Enter size of array: 5
Enter an integer to be inserted: 3
Enter an integer to be inserted: 4
Enter an integer to be inserted: 2
Enter an integer to be inserted: 5
Enter an integer to be inserted: 1
3  4  2  5  1
```

# 2

# C++ Program to Implement Linear Search

```
1. #include <iostream>
2. using namespace std;
3. class LS
4. {
5.     public:
6.         void LinearSearch(int arr[], int value, int i, int n)
7.         {   int found = 0;
8.             for (i = 0; i < n ; i++)
9.             {
10.                         if (value == arr[i] )
11.                         {
12.                             found = 1;
13.                             break;
14.                         }
15.                 }
16.                 if (found == 1)
17.                 {
18.                     cout<<"Element is present in the array at
   position   "<<i+1;
19.                 }
20.                 else
21.                 {
22.                     cout<<"Element is not present in the array.";
23.                 }
24.             }
25.     };
26.     int  main()
27.     {  int num;
28.         int i,  keynum, found = 0;
29.         cout<<"Enter the number of elements   ";
30.         cin>>num;
31.         int array[num];
32.         cout<<"Enter the elements one by one \n";
33.         for (i = 0; i < num; i++)
34.         {
35.             cin>> array[i];
36.         }
37.         cout<<"Enter the element to be searched   ";
38.         cin>>keynum;
39.         /*  Linear search begins */
40.         LS l1;
41.         l1.LinearSearch(array,keynum,i,num);
42.         return 0;
43.     }
```

# RESULTS

```
1. Enter the number of elements   6
   Enter the elements one by one
   4
   6
   1
   2
   5
   3
   Enter the element to be searched   4
   Element is present in the array at position   1

2. Enter the number of elements   3
   Enter the elements one by one
   66
   -3
   31
   Enter the element to be searched   31
   Element is present in the array at position   3

3. Enter the number of elements   5
   Enter the elements one by one
   1
   3
   6
   1
   9
   Enter the element to be searched   10
   Element is not present in the array.
```

# 3
# C++ Program to Implement Binary Search using Iteration

```
#include <iostream>
using namespace std;
class BS
{
public:
    /*
     * Binary Search function
     */
    void BinarySearch(int array[], int keynum, int num)
          {
              int low = 1;
              int high = num;
              int mid;
          do
          {
              mid = (low + high) / 2;
              if (keynum < array[mid])
              {
                  high = mid - 1;
              }
              else if (keynum > array[mid])
              {
                  low = mid + 1;
              }
              }
              while (keynum != array[mid] && low <= high);
              if (keynum == array[mid])
              {
                  cout<<"SEARCH SUCCESSFUL \n";
              }
              else
              {
                  cout<<"SEARCH FAILED \n";
              }
          }
      };
      int main()
      {
          int array[10];
          int i, j, num, temp, keynum;
          int low, mid, high;
          cout<<"Enter the value of num \n";
          cin>>num;
          cout<<"Enter the elements one by one \n";
          for (i = 0; i < num; i++)
          {
              cin>>array[i];
          }
```

```
49.             /*
50.              * Bubble sort
51.              */
52.             for (i = 0; i < num; i++)
53.             {
54.                 for (j = 0; j < (num - i - 1); j++)
55.                 {
56.                     if (array[j] > array[j + 1])
57.                     {
58.                         temp = array[j];
59.                         array[j] = array[j + 1];
60.                         array[j + 1] = temp;
61.                     }
62.                 }
63.             }
64.             cout<<"Enter the element to be searched \n";
65.             cin>>keynum;
66.             // Binary searching begins
67.             BS b1;
68.             b1.BinarySearch(array, keynum, num);
69.             return 0;
70.         }
```

# RESULTS

```
1. Enter the value of num
   6
   Enter the elements one by one
   1 2 3 4 5 6
   Enter the element to be searched
   6
   SEARCH SUCCESSFUL

2. Enter the value of num
   3
   Enter the elements one by one
   -3 31 66
   Enter the element to be searched
   31
   SEARCH SUCCESSFUL

3. Enter the value of num
   5
   Enter the elements one by one
   1 1 3 6 9
   Enter the element to be searched
   10
   SEARCH FAILED
```

# C++ Program to Implement Bubble Sort

```
#include <iostream>

using namespace std;

// Sort arr[] of size n using Bubble Sort.
void BubbleSort (int arr[], int n)
{
        int i, j;
        for (i = 0; i < n; ++i)
        {
                for (j = 0; j < n-i-1; ++j)
                {
                        // Comparing consecutive data and switching values if
value at j > j+1.
                        if (arr[j] > arr[j+1])
                        {
                                arr[j] = arr[j]+arr[j+1];
                                arr[j+1] = arr[j]-arr[j + 1];
                                arr[j] = arr[j]-arr[j + 1];
                        }
                }
                // Value at n-i-1 will be maximum of all the values below this
index.
        }
}

int main()
{
        int n, i;
        cout<<"\nEnter the number of data element to be sorted: ";
        cin>>n;

        int arr[n];
        for(i = 0; i < n; i++)
        {
                cout<<"Enter element "<<i+1<<": ";
                cin>>arr[i];
        }

        BubbleSort(arr, n);

        // Display the sorted data.
        cout<<"\nSorted Data ";
        for (i = 0; i < n; i++)
         cout<<"->"<<arr[i];

        return 0;
}
```

```
Case 1:(average case)
```

```
Enter the number of data element to be sorted: 5
Enter element 1: 998
Enter element 2: 451
Enter element 3: 2
Enter element 4: 35
Enter element 5: 1206

Sorted Data ->2->35->451->998->1206


Case 2:(best case)

Enter the number of data element to be sorted: 5
Enter element 1: 2
Enter element 2: 332
Enter element 3: 456
Enter element 4: 1024
Enter element 5: 16565

Sorted Data ->2->332->456->1024->16565


case 3: (worst case)

Enter the number of data element to be sorted: 5
Enter element 1: 99845
Enter element 2: 564
Enter element 3: 332
Enter element 4: 86
Enter element 5: 1

Sorted Data ->1->86->332->564->99845
```

# 6
# C++ Program to Implement Merge Sort

```
#include <iostream>

using namespace std;

// A function to merge the two half into a sorted data.
void Merge(int *a, int low, int high, int mid)
{
        // We have low to mid and mid+1 to high already sorted.
        int i, j, k, temp[high-low+1];
        i = low;
        k = 0;
        j = mid + 1;

        // Merge the two parts into temp[].
        while (i <= mid && j <= high)
        {
```

```
                if (a[i] < a[j])
                {
                        temp[k] = a[i];
                        k++;
                        i++;
                }
                else
                {
                        temp[k] = a[j];
                        k++;
                        j++;
                }
        }

        // Insert all the remaining values from i to mid into temp[].
        while (i <= mid)
        {
                temp[k] = a[i];
                k++;
                i++;
        }

        // Insert all the remaining values from j to high into temp[].
        while (j <= high)
        {
                temp[k] = a[j];
                k++;
                j++;
        }


        // Assign sorted data stored in temp[] to a[].
        for (i = low; i <= high; i++)
        {
                a[i] = temp[i-low];
        }
}

// A function to split array into two parts.
void MergeSort(int *a, int low, int high)
{
        int mid;
        if (low < high)
        {
                mid=(low+high)/2;
                // Split the data into two half.
                MergeSort(a, low, mid);
                MergeSort(a, mid+1, high);

                // Merge them to get sorted output.
                Merge(a, low, high, mid);
        }
}

int main()
{
        int n, i;
```

```
        cout<<"\nEnter the number of data element to be sorted: ";
        cin>>n;

        int arr[n];
        for(i = 0; i < n; i++)
        {
                cout<<"Enter element "<<i+1<<": ";
                cin>>arr[i];
        }

        MergeSort(arr, 0, n-1);

        // Printing the sorted data.
        cout<<"\nSorted Data ";
        for (i = 0; i < n; i++)
         cout<<"->"<<arr[i];

        return 0;
}
```

# RESULTS

```
Case 1:

Enter the number of data element to be sorted: 10
Enter element 1: 23
Enter element 2: 987
Enter element 3: 45
Enter element 4: 65
Enter element 5: 32
Enter element 6: 9
Enter element 7: 475
Enter element 8: 1
Enter element 9: 17
Enter element 10: 3

Sorted Data ->1->3->9->17->23->32->45->65->475->987
```

# 7
# C++ Program to Implement Heap Sort

```
#include <iostream>

using namespace std;

// A function to heapify the array.
void MaxHeapify(int a[], int i, int n)
{
        int j, temp;
        temp = a[i];
        j = 2*i;
```

```
        while (j <= n)
        {
                if (j < n && a[j+1] > a[j])
                j = j+1;
                // Break if parent value is already greater than child value.
                if (temp > a[j])
                        break;
                // Switching value with the parent node if temp < a[j].
                else if (temp <= a[j])
                {
                        a[j/2] = a[j];
                        j = 2*j;
                }
        }
        a[j/2] = temp;
        return;
}
void HeapSort(int a[], int n)
{
        int i, temp;
        for (i = n; i >= 2; i--)
        {
                // Storing maximum value at the end.
                temp = a[i];
                a[i] = a[1];
                a[1] = temp;
                // Building max heap of remaining element.
                MaxHeapify(a, 1, i - 1);
        }
}
void Build_MaxHeap(int a[], int n)
{
        int i;
        for(i = n/2; i >= 1; i--)
                MaxHeapify(a, i, n);
}
int main()
{
        int n, i;
        cout<<"\nEnter the number of data element to be sorted: ";
        cin>>n;
        n++;
        int arr[n];
        for(i = 1; i < n; i++)
        {
                cout<<"Enter element "<<i<<": ";
                cin>>arr[i];
        }
        // Building max heap.
        Build_MaxHeap(arr, n-1);
        HeapSort(arr, n-1);

        // Printing the sorted data.
        cout<<"\nSorted Data ";

        for (i = 1; i < n; i++)
                cout<<"->"<<arr[i];
```

```
        return 0;
}
```

# RESULTS

```
Case 1:

Enter the number of data element to be sorted: 10
Enter element 1: 9
Enter element 2: 6
Enter element 3: 4
Enter element 4: 3
Enter element 5: 8
Enter element 6: 7
Enter element 7: 5
Enter element 8: 2
Enter element 9: 0
Enter element 10: 1

Sorted Data ->0->1->2->3->4->5->6->7->8->9
```

# 8
# C++ Program to Implement Radix Sort

```
#include <iostream>

using namespace std;

// Get maximum value from array.
int getMax(int arr[], int n)
{
        int max = arr[0];
        for (int i = 1; i < n; i++)
                if (arr[i] > max)
                        max = arr[i];
        return max;
}

// Count sort of arr[].
void countSort(int arr[], int n, int exp)
{
        // Count[i] array will be counting the number of array values having
that 'i' digit at their (exp)th place.
        int output[n], i, count[10] = {0};

        // Count the number of times each digit occurred at (exp)th place in
every input.
        for (i = 0; i < n; i++)
                count[(arr[i] / exp) % 10]++;

        // Calculating their cumulative count.
        for (i = 1; i < 10; i++)
                count[i] += count[i-1];
```

```
        // Inserting values according to the digit '(arr[i] / exp) % 10'
fetched into count[(arr[i] / exp) % 10].
        for (i = n - 1; i >= 0; i--)
        {
                output[count[(arr[i] / exp) % 10] - 1] = arr[i];
                count[(arr[i] / exp) % 10]--;
        }

        // Assigning the result to the arr pointer of main().
        for (i = 0; i < n; i++)
                arr[i] = output[i];
}

// Sort arr[] of size n using Radix Sort.
void radixsort(int arr[], int n)
{
        int exp, m;
        m = getMax(arr, n);

        // Calling countSort() for digit at (exp)th place in every input.
        for (exp = 1; m/exp > 0; exp *= 10)
                countSort(arr, n, exp);
}

int main()
{
        int n, i;
        cout<<"\nEnter the number of data element to be sorted: ";
        cin>>n;

        int arr[n];
        for(i = 0; i < n; i++)
        {
                cout<<"Enter element "<<i+1<<": ";
                cin>>arr[i];
        }

        radixsort(arr, n);

        // Printing the sorted data.
        cout<<"\nSorted Data ";
        for (i = 0; i < n; i++)
                cout<<"->"<<arr[i];
        return 0;
}
```

# RESULTS

```
Enter the number of data element to be sorted: 10
Enter element 1: 886
Enter element 2: 542
Enter element 3: 12
Enter element 4: 3
```

```
Enter element 5: 96
Enter element 6: 1125
Enter element 7: 54
Enter element 8: 129
Enter element 9: 3125
Enter element 10: 1

Sorted Data ->1->3->12->54->96->129->542->886->1125->3125
```

# 9
# C++ Program to Implement Insertion Sort

```
#include <iostream>

using namespace std;

// A structure to represent a node.
struct list
{
        int data;
        list *next;
};

// Function implementing insertion sort.
list* InsertinList(list *head, int n)
{
        // Creating newnode and temp node.
        list *newnode = new list;
        list *temp = new list;

        // Using newnode as the node to be inserted in the list.
        newnode->data = n;
        newnode->next = NULL;

        // If head is null then assign new node to head.
        if(head == NULL)
        {
                head = newnode;
                return head;
        }
        else
        {
                temp = head;

                // If newnode->data is lesser than head->data, then insert
newnode before head.
                if(newnode->data < head->data)
                {
                        newnode->next = head;
                        head = newnode;
                        return head;
                }
```

```
                // Traverse the list till we get value more than newnode-
>data.
                while(temp->next != NULL)
                {
                        if(newnode->data < (temp->next)->data)
                                break;

                        temp=temp->next;
                }

                // Insert newnode after temp.
                newnode->next = temp->next;
                temp->next = newnode;
                return head;
        }
}

int main()
{
        int n, i, num;
        // Declaring head of the linked list.
        list *head = new list;
        head = NULL;

        cout<<"\nEnter the number of data element to be sorted: ";
        cin>>n;

        for(i = 0; i < n; i++)
        {
                cout<<"Enter element "<<i+1<<": ";
                cin>>num;
                // Inserting num in the list.
                head = InsertinList(head, num);
        }

        // Display the sorted data.
        cout<<"\nSorted Data ";
        while(head != NULL)
        {
                cout<<"->"<<head->data;
                head = head->next;
        }

        return 0;
}
```

# RESULT

```
Case 1:(average case)

Enter the number of data element to be sorted: 5
Enter element 1: 998
Enter element 2: 451
Enter element 3: 2
Enter element 4: 35
Enter element 5: 1206
```

```
Sorted Data ->2->35->451->998->1206


Case 2:(best case)

Enter the number of data element to be sorted: 5
Enter element 1: 99845
Enter element 2: 564
Enter element 3: 332
Enter element 4: 86
Enter element 5: 1

Sorted Data ->1->86->332->564->99845


case 3: (worst case)

Enter the number of data element to be sorted: 5
Enter element 1: 2
Enter element 2: 332
Enter element 3: 456
Enter element 4: 1024
Enter element 5: 16565

Sorted Data ->2->332->456->1024->16565
```

# 10

# C++ Program to Implement Selection Sort

```
#include <iostream>

using namespace std;

// Sort arr[] of size n using Selection Sort.
void SelectionSort (int arr[], int n)
{
        int i, j;
        for (i = 0; i < n; ++i)
        {
                for (j = i+1; j < n; ++j)
                {
                        // Comparing consecutive data and switching values if
value at i > j.
                        if (arr[i] > arr[j])
                        {
                                arr[i] = arr[i]+arr[j];
                                arr[j] = arr[i]-arr[j];
                                arr[i] = arr[i]-arr[j];
                        }
                }
```

```
                // Value at i will be minimum of all the values above this
index.
        }
}

int main()
{
        int n, i;
        cout<<"\nEnter the number of data element to be sorted: ";
        cin>>n;

        int arr[n];
        for(i = 0; i < n; i++)
        {
                cout<<"Enter element "<<i+1<<": ";
                cin>>arr[i];
        }

        SelectionSort(arr, n);

        // Display the sorted data.
        cout<<"\nSorted Data ";
        for (i = 0; i < n; i++)
                cout<<"->"<<arr[i];

        return 0;
}
```

# RESULT

```
Case 1:(average case)

Enter the number of data element to be sorted: 5
Enter element 1: 998
Enter element 2: 451
Enter element 3: 2
Enter element 4: 35
Enter element 5: 1206

Sorted Data ->2->35->451->998->1206


Case 2:(best case)

Enter the number of data element to be sorted: 5
Enter element 1: 2
Enter element 2: 332
Enter element 3: 456
Enter element 4: 1024
Enter element 5: 16565

Sorted Data ->2->332->456->1024->16565


case 3: (worst case)
```

```
Enter the number of data element to be sorted: 5
Enter element 1: 99845
Enter element 2: 564
Enter element 3: 332
Enter element 4: 86
Enter element 5: 1

Sorted Data ->1->86->332->564->99845
```