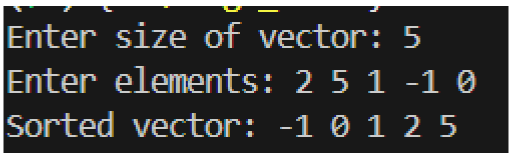
ASSIGNMENT 7

1.A) MERGE SORT

```
#include <iostream>

#include <vector>

using namespace std;


void merge(vector<int>& arr, int left, int mid, int right) {

    int n1 = mid - left + 1;

    int n2 = right - mid;

    vector<int> L(n1), R(n2);


    // Copy data to temp arrays

    for(int i = 0; i < n1; i++)

        L[i] = arr[left + i];

    for(int i = 0; i < n2; i++)

        R[i] = arr[mid + 1 + i];


    // Merge the temp arrays back into arr

    int i = 0, j = 0, k = left;


    while(i < n1 && j < n2) {

        if(L[i] <= R[j]) {

            arr[k] = L[i];

            i++;

        } else {

            arr[k] = R[j];

            j++;
        }
        k++;
    }

    // Copy remaining elements of L
    while(i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    // Copy remaining elements of R
    while(j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(vector<int>& arr, int left, int right) {
    if(left < right) {
        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);      // Sort left half
        mergeSort(arr, mid + 1, right); // Sort right half
        merge(arr, left, mid, right);   // Merge them
    }
}
```

```
int main() {
    int n;
    cout << "Enter size of vector: ";
    cin >> n;

    vector<int> arr(n);
    cout << "Enter elements: ";
    for(int i = 0; i < n; i++)
        cin >> arr[i];

    mergeSort(arr, 0, n - 1);

    cout << "Sorted vector: ";
    for(int x : arr)
        cout << x << " ";

    return 0;
}
```

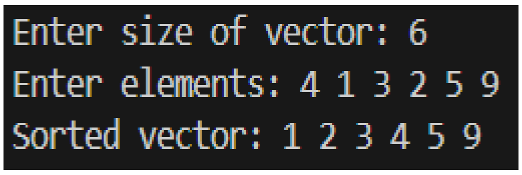


```
Enter size of vector: 5
Enter elements: 2 5 1 -1 0
Sorted vector: -1 0 1 2 5
```

B) QUICK SORT

```
#include <iostream>

#include <vector>

using namespace std;


int partition(vector<int>& arr, int low, int high) {

    int pivot = arr[high];   // pivot element

    int i = low - 1;         // index of smaller element


    for(int j = low; j < high; j++) {

        if(arr[j] < pivot) {

            i++;

            swap(arr[i], arr[j]);

        }

    }


    swap(arr[i + 1], arr[high]); // place pivot in correct position

    return i + 1;

}


void quickSort(vector<int>& arr, int low, int high) {

    if(low < high) {

        int pi = partition(arr, low, high);


        // Recursively sort elements before and after partition

        quickSort(arr, low, pi - 1);

        quickSort(arr, pi + 1, high);
    }
}

int main() {
    int n;
    cout << "Enter size of vector: ";
    cin >> n;

    vector<int> arr(n);
    cout << "Enter elements: ";
    for(int i = 0; i < n; i++)
        cin >> arr[i];

    quickSort(arr, 0, n - 1);

    cout << "Sorted vector: ";
    for(int x : arr)
        cout << x << " ";

    return 0;
}



        // Recursively sort elements before and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

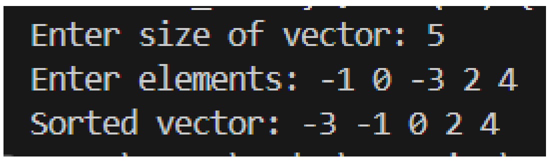
```
int main() {
    int n;
    cout << "Enter size of vector: ";
    cin >> n;

    vector<int> arr(n);
    cout << "Enter elements: ";
    for(int i = 0; i < n; i++)
        cin >> arr[i];
    quickSort(arr, 0, n - 1);
    cout << "Sorted vector: ";
    for(int x : arr)
        cout << x << " ";
    return 0;
}
```



```
Enter size of vector: 6
Enter elements: 4 1 3 2 5 9
Sorted vector: 1 2 3 4 5 9
```

C) BUBBLE SORT

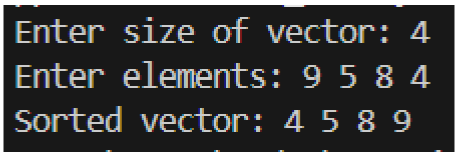
```
#include <iostream>

#include <vector>

using namespace std;


void bubbleSort(vector<int>& arr) {

    int n = arr.size();

    for(int i = 0; i < n - 1; i++) {

        bool swapped = false;

        for(int j = 0; j < n - i - 1; j++) {

            if(arr[j] > arr[j + 1]) {

                swap(arr[j], arr[j + 1]);

                swapped = true;

            }

        }

        if(!swapped)

            break;  // array already sorted

    }

}


int main() {

    int n;

    cout << "Enter size of vector: ";

    cin >> n;


    vector<int> arr(n);

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

    bubbleSort(arr);

    cout << "Sorted vector: ";
    for(int x : arr)
        cout << x << " ";

    return 0;
}
```



```
Enter size of vector: 5
Enter elements: -1 0 -3 2 4
Sorted vector: -3 -1 0 2 4
```

D) INSERTION SORT

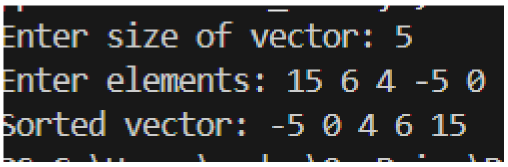
```
#include <iostream>
#include <vector>
using namespace std;

void insertionSort(vector<int>& arr) {
    int n = arr.size();
    for(int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;


        while(j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

int main() {
    int n;
    cout << "Enter size of vector: ";
    cin >> n;

    vector<int> arr(n);
    cout << "Enter elements: ";
    for(int i = 0; i < n; i++)
        cin >> arr[i];

    insertionSort(arr);

    cout << "Sorted vector: ";
    for(int x : arr)
        cout << x << " ";

    return 0;
}
```



```
Enter size of vector: 4
Enter elements: 9 5 8 4
Sorted vector: 4 5 8 9
```

E) SELECTION SORT

```
#include <iostream>

#include <vector>

using namespace std;


void selectionSort(vector<int>& arr) {

    int n = arr.size();


    for(int i = 0; i < n - 1; i++) {

        int minIndex = i;


        // Find the index of the minimum element in the remaining array

        for(int j = i + 1; j < n; j++) {

            if(arr[j] < arr[minIndex]) {

                minIndex = j;

            }

        }


        // Swap the found minimum element with the first element

        if(minIndex != i) {

            swap(arr[i], arr[minIndex]);

        }

    }

}


int main() {
    int n;

    cout << "Enter size of vector: ";

    cin >> n;


    vector<int> arr(n);

    cout << "Enter elements: ";

    for(int i = 0; i < n; i++)

        cin >> arr[i];


    selectionSort(arr);


    cout << "Sorted vector: ";

    for(int x : arr)

        cout << x << " ";


    return 0;

}
```



```
Enter size of vector: 5
Enter elements: 15 6 4 -5 0
Sorted vector: -5 0 4 6 15
```

2. A slightly improved selection sort – We know that selection sort algorithm takes the minimum on

every pass on the array, and place it at its correct position. The idea is to take also the maximum on

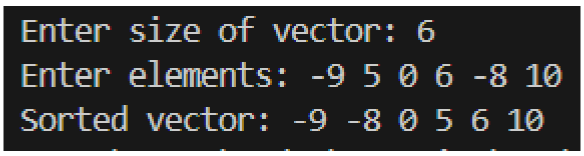
every pass and place it at its correct position. So in every pass, we keep track of both maximum and

minimum and array becomes sorted from both ends. Implement this logic.

```
#include <iostream>

#include <vector>

using namespace std;


void improvedSelectionSort(vector<int>& arr) {

  int left = 0;

  int right = arr.size() - 1;


  while (left < right) {

    int minIndex = left;

    int maxIndex = right;


    // If the elements at ends are inverted, fix them

    if (arr[minIndex] > arr[maxIndex]) {

      swap(arr[minIndex], arr[maxIndex]);

    }


    // Find min and max in the remaining part

    for (int i = left + 1; i < right; i++) {

      if (arr[i] < arr[minIndex])
```

```
                minIndex = i;
            else if (arr[i] > arr[maxIndex])
                maxIndex = i;
        }

        // Place minimum at beginning
        swap(arr[left], arr[minIndex]);

        // If max element was moved because it was at left
        if (maxIndex == left)
            maxIndex = minIndex;

        // Place maximum at end
        swap(arr[right], arr[maxIndex]);

        left++;
        right--;
    }
}

int main() {
    int n;
    cout << "Enter size of vector: ";
    cin >> n;

    vector<int> arr(n);
    cout << "Enter elements: ";
    for (int i = 0; i < n; i++)
```

```
        cin >> arr[i];

    improvedSelectionSort(arr);

    cout << "Sorted vector: ";
    for (int x : arr)
        cout << x << " ";

    return 0;
}
```



```
Enter size of vector: 6
Enter elements: -9 5 0 6 -8 10
Sorted vector: -9 -8 0 5 6 10
```