Week-3:

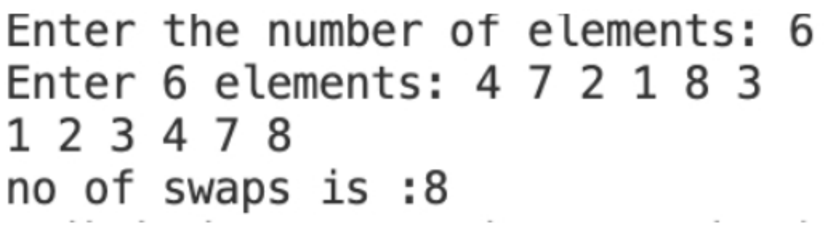
Sorting Algorithms

Q1. Implement the generic bubble sort algorithm to sort an array of elements in ascending order. Modify the bubble sort implementation to count the number of swaps made during the sorting process. Print the total number of swaps after sorting the array.

Code:

```
#include <bits/stdc++.h>
using namespace std;
int main(){
 int k;
    cout << "Enter the number of elements: ";
    cin >> k;

    vector<int> nums(k);

    cout << "Enter " << k << " elements: ";
    for (int i = 0; i < k; ++i)
    {
        cin >> nums[i];
    }
    int count =0;

    for(int i =0 ; i< k ; i++){
        int flag =0;
        for(int j = 0; j< k-i-1; j++){
            if(nums[j+1]<nums[j])
                { swap(nums[j],
                nums[j+1]); count++;
                flag =1;
            }
        }
        if(flag ==0){
            break;
        }
    }
     for (int i = 0; i < k; ++i)
    {
        cout<< nums[i]<<" ";
    }
    cout<<endl<<"no of swaps is :"<<count<<endl;
}
```

Output:



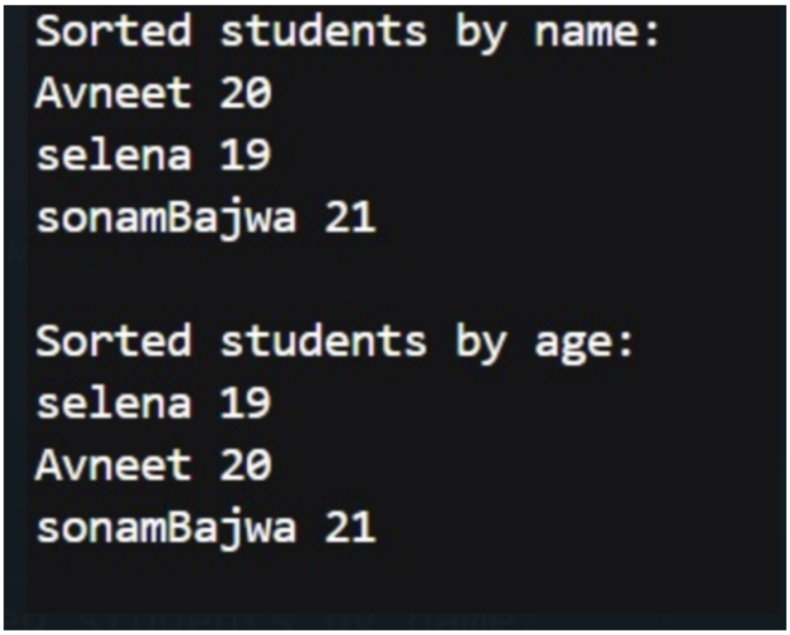
```
Enter the number of elements: 6
Enter 6 elements: 4 7 2 1 8 3
1 2 3 4 7 8
no of swaps is :8
```

Q2. Implement the insertion sort algorithm to sort an array of integers in ascending order. Modify the insertion sort implementation to sort an array of students' structures [Problem statement 6 of Week 1] in lexicographical order (dictionary or alphabetical order) of student names.

Code:

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
struct Student
{
    string name;
    int age;
};
void insertionSort(vector<Student> &arr)
{
    int n = arr.size();
    for (int i = 1; i < n; ++i)
    {
        Student key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j].name > key.name)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
void insertionsort_age(vector<Student> &arr)
{
    int n = arr.size();

    for (int i = 0; i < n; ++i)
    {
        Student key = arr[i];
        int ii = i - 1;
        while (ii >= 0 && arr[ii].age > key.age)
        {
            arr[ii + 1] = arr[ii];
            ii = ii - 1;
        }
        arr[ii + 1] = key;
    }
}
int main()
{
    vector<Student> students = {{"Avneet", 20}, {"selena",
19}, {"sonamBajwa", 21}};
    insertionSort(students);
    cout << "Sorted students by name:\n";
    for (const Student &student : students)
    {
        cout << student.name << " " << student.age << endl;
    }
    cout << "\n";
    insertionsort_age(students);
    cout << "Sorted students by age:\n";
    for (const Student &student : students)
    {
        cout << student.name << " " << student.age << endl;
    }
    return 0;
}
```

Output:



```
Sorted students by name:
Avneet 20
selena 19
sonamBajwa 21

Sorted students by age:
selena 19
Avneet 20
sonamBajwa 21
```

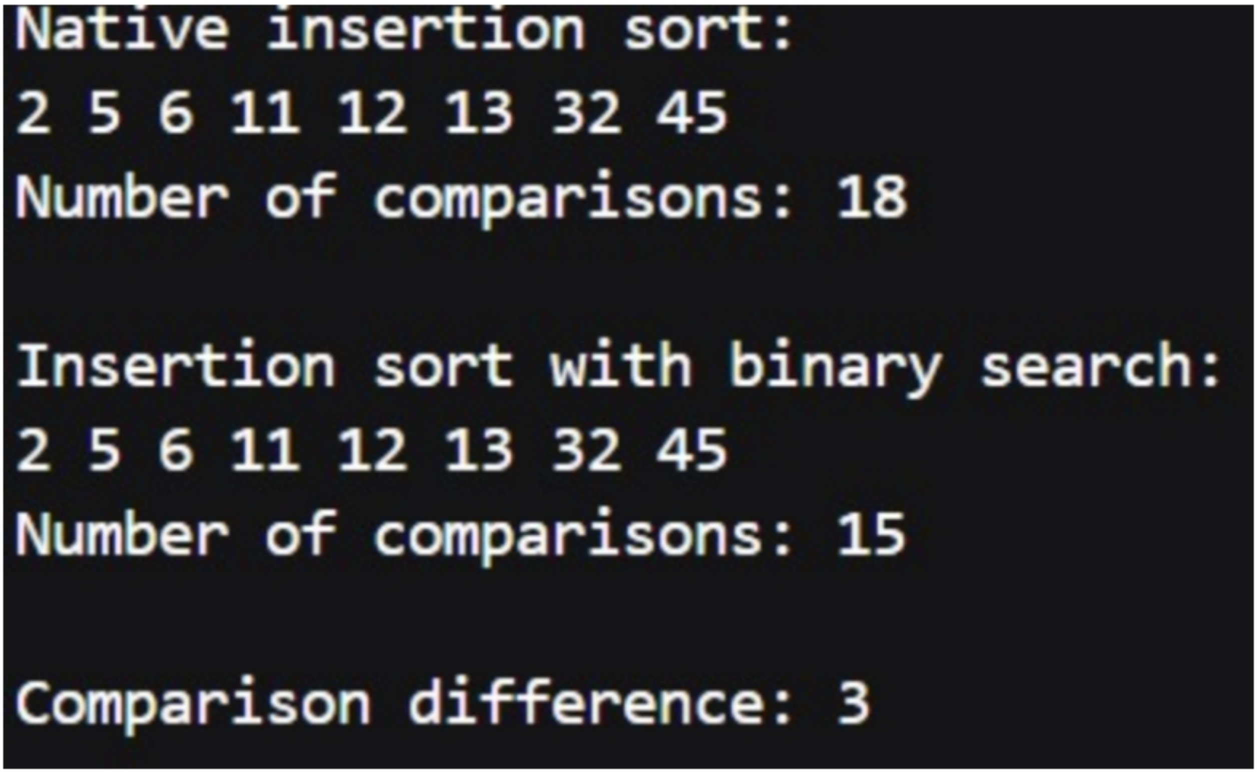
Q3. Modify the insertion sort implementation to employ binary search to find the correct position of the element to be placed in the sorted subarray to decrease the # of comparisons required. Run the native and modified insertion sort on the same inputs and assess the performance of the modified algorithm under different data distributions.

Code:

```
#include <iostream>
#include <vector>
using namespace std;
int comparison_count = 0;
int binarySearch(const vector<int> & arr, int key, int low,
int high)
{
    while (low <= high)
    {
        int mid = low + (high - low) / 2;
        comparison_count++;
        if (arr[mid] == key)
            return mid;
        else if (arr[mid] < key)
```

```
            low = mid + 1;
        else
            high = mid - 1;
    }
    return low;
}
void insertionSortWithBinarySearch(vector<int> &arr)
{
    comparison_count = 0;
    int n = arr.size();
    for (int i = 1; i < n; ++i)
    {
        int key = arr[i];
        int j = i - 1;
        int pos = binarySearch(arr, key, 0, j);
        while (j >= pos)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
void insertionSort(vector<int> &arr)
{
    comparison_count = 0;
    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 = j - 1;
            comparison_count++;
        }
        arr[j + 1] = key;
    }
}
int main()
{
    vector<int> nums = {32, 2, 45, 12, 11, 13, 5, 6};
    cout << "Native insertion sort:"
         << "\n";
    insertionSort(nums);
    for (int num : nums)
    {
        cout << num << " ";
    }
    cout << "\n";
    cout << "Number of comparisons: " << comparison_count <<
"\n";
    int x = comparison_count;
    cout << "\nInsertion sort with binary search:"
         << "\n";
    vector<int> nums2 = {32, 2, 45, 12, 11, 13, 5, 6};
    insertionSortWithBinarySearch(nums2);
    for (int num : nums2)
    {
        cout << num << " ";
    }
    cout << "\n";
    cout << "Number of comparisons: " << comparison_count <<
"\n";
    int y = comparison_count;
    int d = x - y;
    cout << "\n";
    cout << "Comparison difference: " << d << "\n";
    return 0;
}
```

Output:



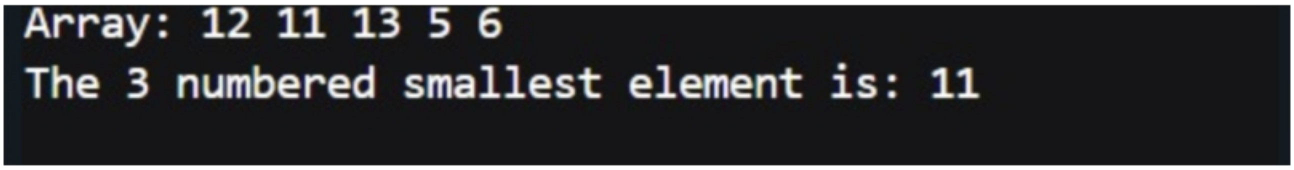
```
Native insertion sort:
2 5 6 11 12 13 32 45
Number of comparisons: 18

Insertion sort with binary search:
2 5 6 11 12 13 32 45
Number of comparisons: 15

Comparison difference: 3
```

Q4. Implement the selection sort algorithm to sort an array of integers in ascending order. Modify the selection sort implementation to find the kth smallest element in an array.

Code:

```
#include <iostream>
#include <vector>
using namespace std;
int findKthSmallest(vector<int> &arr, int k)
{
    int n = arr.size();
    if (k < 0 || k >= n)
    {
        cout << "Invalid value of k." << endl;
        return -1;
    }
    for (int i = 0; i < k; ++i)
    {
        int min_index = i;
        for (int j = i + 1; j < n; ++j)
        {
            if (arr[j] < arr[min_index])
            {
                min_index = j;
            }
        }
        if (min_index != i)
        {
            swap(arr[i], arr[min_index]);
        }
    }
    return arr[k - 1];
}
int main()
{
    vector<int> arr = {12, 11, 13, 5, 6};
    int k = 3;
    cout << "Array: ";
    for (int num : arr)
    {
        cout << num << " ";
    }
    cout << endl;
    int kth_smallest = findKthSmallest(arr, k);
    cout << "The " << k << " numbered smallest element is: " <<
kth_smallest << endl;
    return 0;
}
```

Output:



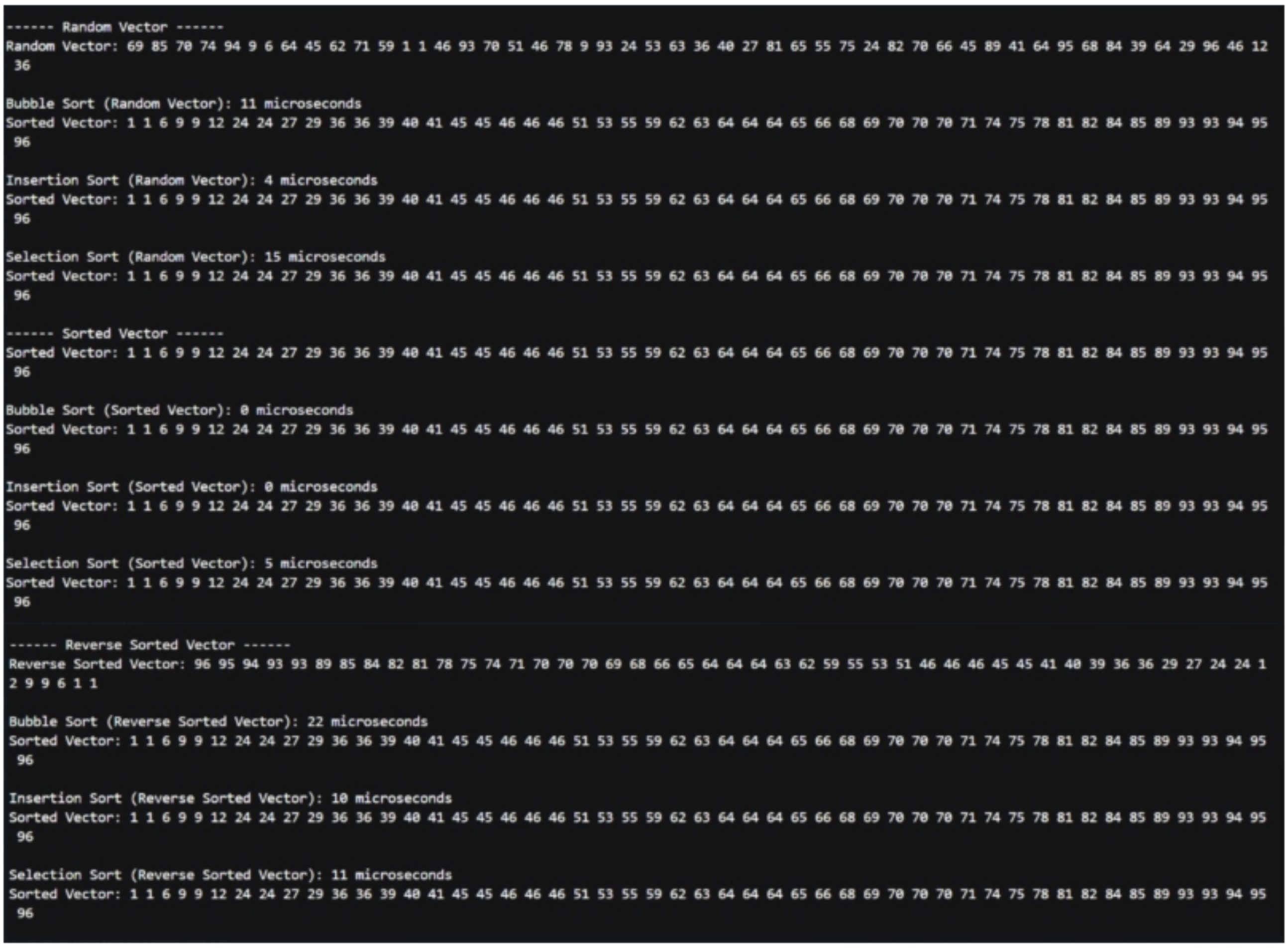
```
Array: 12 11 13 5 6
The 3 numbered smallest element is: 11
```

Q5. Compare the performance of bubble, insertion, and selection sort algorithms based on their sorting time and sensitivity to the randomly generated input data distribution. Identify the strengths and weaknesses of each algorithm under different scenarios(i.e., random collection, sorted and reversely sorted).

Code:

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <vector>
#include <chrono>
#include <algorithm>
#include <iomanip>
using namespace std;
using namespace std::chrono;
vector<int> randomVec, sortedVec, reverseSortedVec;
void generateRandomNumbers()
{
    randomVec.clear();
    sortedVec.clear();
    reverseSortedVec.clear();
    srand(time(0));
    for (int i = 0; i < 50; i++)
    {
        int num = (rand() % 100) + 1;
        randomVec.emplace_back(num);
        sortedVec.emplace_back(num);
        reverseSortedVec.emplace_back(num);
    }
    sort(sortedVec.begin(), sortedVec.end());
    sort(reverseSortedVec.begin(), reverseSortedVec.end(),
greater<int>());
}
void printVector(const vector<int> &vec, const string
&caption)
{
    cout << caption << ": ";
    for (size_t i = 0; i < vec.size(); i++)
    {
        cout << vec[i] << " ";
    }
    cout << "\n";
}
int bubble_sort(const vector<int> &vec, const string
&caption)
{
    vector<int> arr_copy = vec;
    size_t n = arr_copy.size();
    auto start = high_resolution_clock::now();

    for (size_t i = 0; i < n; ++i)
    {
        int sw = 0;
        for (size_t j = 0; j < n - i - 1; ++j)
        {
            if (arr_copy[j] > arr_copy[j + 1])
            {
                std::swap(arr_copy[j], arr_copy[j + 1]);
                sw++;
            }
        }
        if (sw == 0)
        {
            break;
        }
    }
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop -
start);
    cout << "\nBubble Sort (" << caption << "): " <<
duration.count() << " microseconds\n";
    printVector(arr_copy, "Sorted Vector");
    return 0;
}
int insertion_sort(const vector<int> &vec, const string
&caption)
{
    vector<int> arr_copy = vec;
    size_t n = arr_copy.size();
    auto start = high_resolution_clock::now();

    for (size_t i = 1; i < n; i++)
    {
        int key = arr_copy[i];
        int j = i - 1;

        while (j >= 0 && arr_copy[j] > key)
        {
            arr_copy[j + 1] = arr_copy[j];
            j--;
        }
        arr_copy[j + 1] = key;
    }
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop -
start);
    cout << "\nInsertion Sort (" << caption << "): " <<
duration.count() << " microseconds\n";
    printVector(arr_copy, "Sorted Vector");
    return 0;
}
int selection_sort(const vector<int> &vec, const string
&caption)
{
    vector<int> arr_copy = vec;
    size_t n = arr_copy.size();
    auto start = high_resolution_clock::now();

    for (size_t i = 0; i < n - 1; i++)
    {
        int min_idx = i;
        for (size_t j = i + 1; j < n; j++)
        {
            if (arr_copy[j] < arr_copy[min_idx])
            {
                min_idx = j;
            }
        }
        swap(arr_copy[min_idx], arr_copy[i]);
    }
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop -
start);
    cout << "\nSelection Sort (" << caption << "): " <<
duration.count() << " microseconds\n";
    printVector(arr_copy, "Sorted Vector");
    return 0;
}
int main()
{
    generateRandomNumbers();
    cout << "\n------ Random Vector ------\n";
    printVector(randomVec, "Random Vector");
    bubble_sort(randomVec, "Random Vector");
    insertion_sort(randomVec, "Random Vector");
    selection_sort(randomVec, "Random Vector");
    cout << "\n------ Sorted Vector ------\n";
    printVector(sortedVec, "Sorted Vector");
    bubble_sort(sortedVec, "Sorted Vector");
    insertion_sort(sortedVec, "Sorted Vector");
    selection_sort(sortedVec, "Sorted Vector");
    cout << "\n------ Reverse Sorted Vector ------\n";
    printVector(reverseSortedVec, "Reverse Sorted Vector");
    bubble_sort(reverseSortedVec, "Reverse Sorted Vector");
    insertion_sort(reverseSortedVec, "Reverse Sorted
Vector");
    selection_sort(reverseSortedVec, "Reverse Sorted
Vector");
    return 0;
}
```

Output:



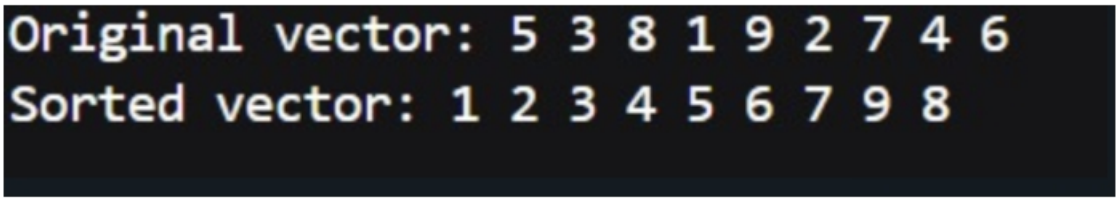
```
------ Random Vector ------
Random Vector: 69 85 70 74 94 9 6 64 45 62 71 59 1 1 46 93 70 51 46 78 9 93 24 53 63 36 40 27 81 65 55 75 24 82 70 66 45 89 41 64 95 68 84 39 64 29 96 46 12
 36

Bubble Sort (Random Vector): 11 microseconds
Sorted Vector: 1 1 6 9 9 12 24 24 27 29 36 36 39 40 41 45 45 46 46 46 51 53 55 59 62 63 64 64 64 65 66 68 69 70 70 70 71 74 75 78 81 82 84 85 89 93 93 94 95
 96

Insertion Sort (Random Vector): 4 microseconds
Sorted Vector: 1 1 6 9 9 12 24 24 27 29 36 36 39 40 41 45 45 46 46 46 51 53 55 59 62 63 64 64 64 65 66 68 69 70 70 70 71 74 75 78 81 82 84 85 89 93 93 94 95
 96

Selection Sort (Random Vector): 15 microseconds
Sorted Vector: 1 1 6 9 9 12 24 24 27 29 36 36 39 40 41 45 45 46 46 46 51 53 55 59 62 63 64 64 64 65 66 68 69 70 70 70 71 74 75 78 81 82 84 85 89 93 93 94 95
 96

------ Sorted Vector ------
Sorted Vector: 1 1 6 9 9 12 24 24 27 29 36 36 39 40 41 45 45 46 46 46 51 53 55 59 62 63 64 64 64 65 66 68 69 70 70 70 71 74 75 78 81 82 84 85 89 93 93 94 95
 96

Bubble Sort (Sorted Vector): 0 microseconds
Sorted Vector: 1 1 6 9 9 12 24 24 27 29 36 36 39 40 41 45 45 46 46 46 51 53 55 59 62 63 64 64 64 65 66 68 69 70 70 70 71 74 75 78 81 82 84 85 89 93 93 94 95
 96

Insertion Sort (Sorted Vector): 0 microseconds
Sorted Vector: 1 1 6 9 9 12 24 24 27 29 36 36 39 40 41 45 45 46 46 46 51 53 55 59 62 63 64 64 64 65 66 68 69 70 70 70 71 74 75 78 81 82 84 85 89 93 93 94 95
 96

Selection Sort (Sorted Vector): 5 microseconds
Sorted Vector: 1 1 6 9 9 12 24 24 27 29 36 36 39 40 41 45 45 46 46 46 51 53 55 59 62 63 64 64 64 65 66 68 69 70 70 70 71 74 75 78 81 82 84 85 89 93 93 94 95
 96

------ Reverse Sorted Vector ------
Reverse Sorted Vector: 96 95 94 93 93 89 85 84 82 81 78 75 74 71 70 70 70 69 68 66 65 64 64 64 63 62 59 55 53 51 46 46 46 45 45 41 40 39 36 36 29 27 24 24 1
2 9 9 6 1 1

Bubble Sort (Reverse Sorted Vector): 22 microseconds
Sorted Vector: 1 1 6 9 9 12 24 24 27 29 36 36 39 40 41 45 45 46 46 46 51 53 55 59 62 63 64 64 64 65 66 68 69 70 70 70 71 74 75 78 81 82 84 85 89 93 93 94 95
 96

Insertion Sort (Reverse Sorted Vector): 10 microseconds
Sorted Vector: 1 1 6 9 9 12 24 24 27 29 36 36 39 40 41 45 45 46 46 46 51 53 55 59 62 63 64 64 64 65 66 68 69 70 70 70 71 74 75 78 81 82 84 85 89 93 93 94 95
 96

Selection Sort (Reverse Sorted Vector): 11 microseconds
Sorted Vector: 1 1 6 9 9 12 24 24 27 29 36 36 39 40 41 45 45 46 46 46 51 53 55 59 62 63 64 64 64 65 66 68 69 70 70 70 71 74 75 78 81 82 84 85 89 93 93 94 95
 96
```

Q6. Implement a strategy to terminate the selection sorting process early if the array becomes sorted before completing all iterations. Track whether swaps were made in each iteration and stop sorting if no swaps occur.

Code:

```
#include <iostream>
#include <vector>
using namespace std;
void printVector(const vector<int> &vec)
{
    for (int num : vec)
    {
        cout << num << " ";
    }
    cout << "\n";
}
void selectionSort(vector<int> &vec)
{
    int n = vec.size();
    bool swapped;

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

        for (int j = i + 1; j < n; j++)
        {
            if (vec[j] < vec[min_idx])
            {
                min_idx = j;
            }
        }
        if (min_idx != i)
        {
            swap(vec[min_idx], vec[i]);
            swapped = true;
        }

        if(!swapped)
        {
            break; // Early termination
        }
    }
}
int main()
{
    vector<int> vec = {5, 3, 8, 1, 9, 2, 7, 4, 6};
    cout << "Original vector: ";
    printVector(vec);
    selectionSort(vec);
    cout << "Sorted vector: ";
    printVector(vec);
    return 0;
}
```

Output:



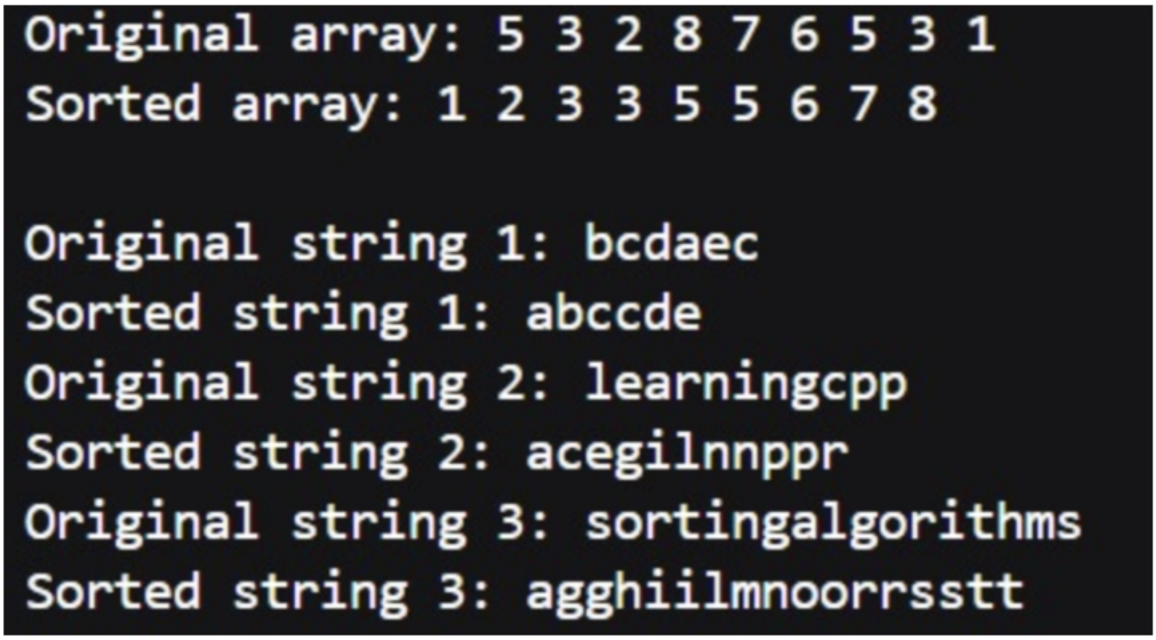
```
Original vector: 5 3 8 1 9 2 7 4 6
Sorted vector: 1 2 3 4 5 6 7 9 8
```

Q 7. Implement the counting sort algorithm to sort an array of positive integers in ascending order. Also, the counting sort implementation should be modified to sort an array of characters in lexicographical order. Test the implementation with various strings and verify its correctness.

Code:

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;
void countingSort(vector<int>& arr) {
    int n = arr.size();
    int max_val = *max_element(arr.begin(), arr.end());
    vector<int> count(max_val + 1, 0);
    vector<int> output(n);
    for (int num : arr) {
        count[num]++;
    }
    for (int i = 1; i <= max_val; i++) {
        count[i] += count[i - 1];
    }
    for (int i = n - 1; i >= 0; i--)
        { output[count[arr[i]] - 1] =
        arr[i]; count[arr[i]]--;
    }
    for (int i = 0; i < n; i++) {
        arr[i] = output[i];
    }
}
void countingSort(string& str)
    { int n = str.length();
    vector<int> count(256, 0);
    vector<char> output(n);
    for (char c : str) {
        count[c]++;
    }
    for (int i = 1; i < 256; i++) {
        count[i] += count[i - 1];
    }
    for (int i = n - 1; i >= 0; i--)
        { output[count[str[i]] - 1] =
        str[i]; count[str[i]]--;
    }
    for (int i = 0; i < n; i++) {
        str[i] = output[i];
    }
}
int main() {
    vector<int> arr = {5, 3, 2, 8, 7, 6, 5, 3, 1};
    cout << "Original array: ";
    for (int num : arr) {
        cout << num << " ";
    }
    cout << "\n";
    countingSort(arr);
    cout << "Sorted array: ";
    for (int num : arr) {
        cout << num << " ";
    }
    cout << "\n\n";
    string str1 = "bcdaec";
    string str2 = "learningcpp";
    string str3 = "sortingalgorithms";
    cout << "Original string 1: " << str1 << "\n";
    countingSort(str1);
    cout << "Sorted string 1: " << str1 << "\n";
    cout << "Original string 2: " << str2 << "\n";
    countingSort(str2);
    cout << "Sorted string 2: " << str2 << "\n";
    cout << "Original string 3: " << str3 << "\n";
    countingSort(str3);
    cout << "Sorted string 3: " << str3 << "\n";
    return 0;
}
```

Output:



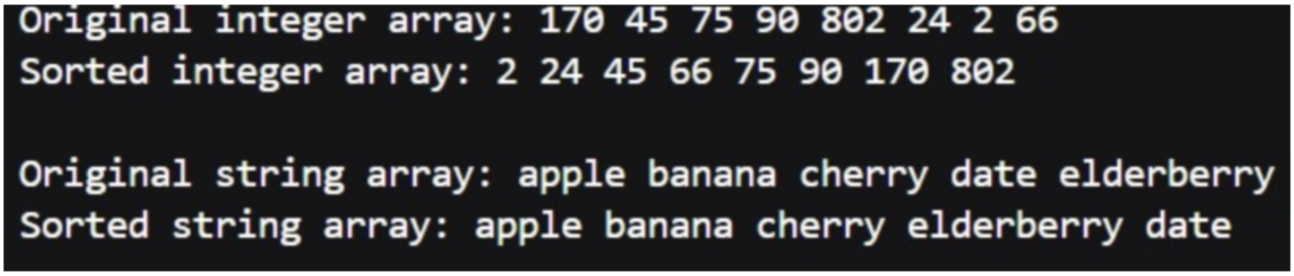
```
Original array: 5 3 2 8 7 6 5 3 1
Sorted array: 1 2 3 3 5 5 6 7 8

Original string 1: bcdaec
Sorted string 1: abccde
Original string 2: learningcpp
Sorted string 2: acegilnnppr
Original string 3: sortingalgorithms
Sorted string 3: agghiilmnoorrsstt
```

Q8. Implement the radix sort algorithm to sort an array of positive integers in ascending order. Modify the radix sort implementation to sort an array of strings in lexicographical order.

Code:

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;
int getMax(vector<int>& arr) {
    int max_val = arr[0];
    for (int i = 1; i < arr.size(); i++) {
        if (arr[i] > max_val)
            max_val = arr[i];
    }
    return max_val;
}
void countSort(vector<int>& arr, int exp) {
    int n = arr.size();
    vector<int> output(n);
    int count[10] = {0};

    for (int num : arr)
        count[(num / exp) % 10]++;

    for (int i = 1; i < 10; i++)
        count[i] += count[i - 1];

    for (int i = n - 1; i >= 0; i--)
        { output[count[(arr[i] / exp) % 10] - 1] =
        arr[i]; count[(arr[i] / exp) % 10]--;
    }

    for (int i = 0; i < n; i++)
        arr[i] = output[i];
}
void radixSort(vector<int>& arr) {
    int max_val = getMax(arr);
    for (int exp = 1; max_val / exp > 0; exp *= 10)
        countSort(arr, exp);
}
int getMax(vector<string>& arr) {
    int max_len = 0;
    for (string str : arr)
        max_len = max(max_len, (int)str.length());
    return max_len;
}
void countSort(vector<string>& arr, int exp) {
    int n = arr.size();
    vector<string> output(n);
    int count[256] = {0};
    for (string str : arr)
        count[(int)(str[str.length() - exp / 8 - 1])]++;
    for (int i = 1; i < 256; i++)
        count[i] += count[i - 1];
    for (int i = n - 1; i >= 0; i--) {
        output[count[(int)(arr[i][arr[i].length() - exp / 8 -
1])] - 1] = arr[i];
        count[(int)(arr[i][arr[i].length() - exp / 8 -
1])]--;
    }
    for (int i = 0; i < n; i++)
        arr[i] = output[i];
}
void radixSort(vector<string>& arr) {
    int max_len = getMax(arr);
    for (int exp = 8; max_len / exp > 0; exp *= 8)
        countSort(arr, exp);
}
int main() {
    vector<int> intArr = {170, 45, 75, 90, 802, 24, 2, 66};
    vector<string> strArr = {"apple", "banana", "cherry",
"date", "elderberry"};
    cout << "Original integer array: ";
    for (int num : intArr)
        cout << num << " ";
    cout << "\n";
    radixSort(intArr);
    cout << "Sorted integer array: ";
    for (int num : intArr)
        cout << num << " ";
    cout << "\n\n";
    cout << "Original string array: ";
    for (string str : strArr)
        cout << str << " ";
    cout << "\n";
    radixSort(strArr);
    cout << "Sorted string array: ";
    for (string str : strArr)
        cout << str << " ";
    cout << "\n";
    return 0;
}
```

Output:



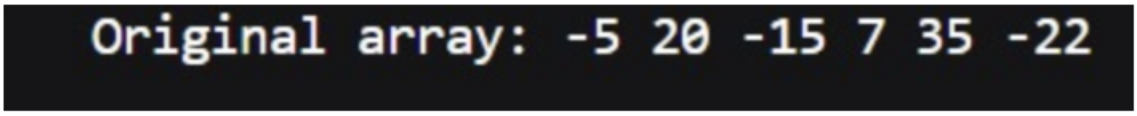
```
Original integer array: 170 45 75 90 802 24 2 66
Sorted integer array: 2 24 45 66 75 90 170 802

Original string array: apple banana cherry date elderberry
Sorted string array: apple banana cherry elderberry date
```

Q9. Implement the Radix Sort algorithm that handles input arrays with a mix of positive and negative integers.

Code:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int getMax(vector<int>& arr) {
    int max_val = arr[0];
    for (int num : arr)
        max_val = max(max_val, abs(num));
    return max_val;
}
void countSort(vector<int>& arr, int exp) {
    int n = arr.size();
    vector<int> output(n);
    int count[19] = {0};
    for (int num : arr)
        count[(num / exp) % 19 + 9]++;
    for (int i = 1; i < 19; i++)
        count[i] += count[i - 1];
    for (int i = n - 1; i >= 0; i--) {
        output[count[(arr[i] / exp) % 19 + 9] - 1] = arr[i];
        count[(arr[i] / exp) % 19 + 9]--;
    }
    for (int i = 0; i < n; i++)
        arr[i] = output[i];
}
void radixSort(vector<int>& arr) {
    int max_val = getMax(arr);

    for (int exp = 1; max_val / exp > 0; exp *= 19)
        countSort(arr, exp);
}
int main() {
    vector<int> arr = {-5, 20, -15, 7, 35, -22};
    cout << "Original array: ";
    for (int num : arr)
        cout << num << " ";
    cout << "\n";
    radixSort(arr);
    cout << "Sorted array: ";
    for (int num : arr)
        cout << num << " ";
    cout << "\n";
    return 0;
}
```

Output:



```
Original array: -5 20 -15 7 35 -22
```