**DESIGN AND ANALYSIS OF ALGORITHM: PPS1**

**PRATEEK MOHANTY**

**20BCE1482**

**Q1**

**SELECTION SORT**

**ALGORITHM**

1. Step 1: Repeat Steps 2 **and** 3 **for** i = 0 to n-1
2. Step 2: CALL SMALLEST(arr, i, n, pos)
3. Step 3: SWAP arr[i] with arr[pos]
4. [END OF LOOP]
5. Step 4: EXIT
7. SMALLEST (arr, i, n, pos)
8. Step 1: [INITIALIZE] SET SMALL = arr[i]
9. Step 2: [INITIALIZE] SET pos = i
10. Step 3: Repeat **for** j = i+1 to n
11. **if** (SMALL > arr[j])
12. SET SMALL = arr[j]
13. SET pos = j
14. [END OF **if**]
15. [END OF LOOP]
16. Step 4: RETURN pos

**CODE**

#include <iostream>

#include <vector>

using namespace std;

// Selection sort

void swap(int \*a, int \*b)

{

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

int main()

{

    int n;

    cout << "Enter number of elements in the array" << endl;

    cin >> n;

    vector<int> vec1;

    for (int i = 0; i < n; i++)

    {

        int a;

        cin >> a;

        vec1.push\_back(a);

    }

    cout << "Vector before sorting" << endl;

    for (int i = 0; i < n; i++)

    {

        cout << vec1[i] << " ";

    }

    cout << endl;

    for (int i = 0; i < n - 1; i++)

    {

        int index = i;

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

        {

            if (vec1[j] < vec1[index])

                index = j;

        }

        swap(vec1[i], vec1[index]);

    }

    cout << "Vector after sorting" << endl;

    for (int i = 0; i < n; i++)

    {

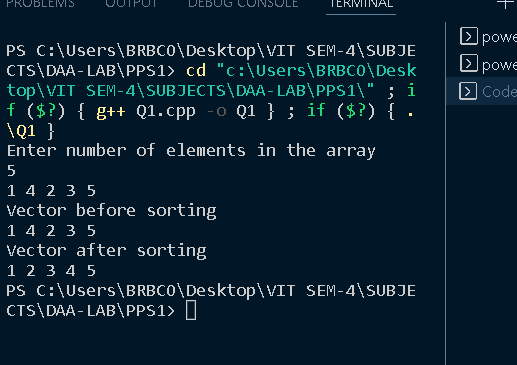
        cout << vec1[i] << " ";

    }

    return 0;

}

**OUTPUT**

****

**TIME COMPLEXITY : O(N^2)**

**INSERTION SORT TIME COMPLEXITY : O(N^2)**

Q2

ALGORITHM

begin BubbleSort(list)

for all elements of list

if list[i] > list[i+1]

swap(list[i], list[i+1])

end if

end for

return list

end BubbleSort

**CODE**

#include <iostream>

#include <vector>

using namespace std;

void swap(int \*a, int \*b)

{

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

int main()

{

    int n;

    cout << "Enter number of elements in the array" << endl;

    cin >> n;

    vector<int> vec1;

    for (int i = 0; i < n; i++)

    {

        int a;

        cin >> a;

        vec1.push\_back(a);

    }

    cout << "Vector before sorting" << endl;

    for (int i = 0; i < n; i++)

    {

        cout << vec1[i] << " ";

    }

    cout << endl;

    for (int i = 0; i < n - 1; i++)

    {

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

        {

            if (vec1[j] > vec1[j + 1])

            {

                swap(vec1[j + 1], vec1[j]);

            }

        }

    }

    cout << "Vector after sorting" << endl;

    for (int i = 0; i < n; i++)

    {

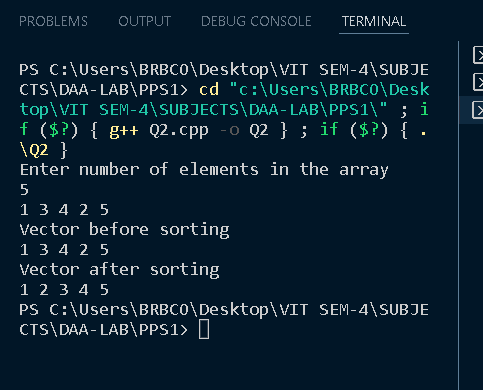
        cout << vec1[i] << " ";

    }

    return 0;

}

**OUTPUT**



**TIME COMPLEXITY : O(N^2)**

**INSERTION SORT TIME COMPLEXITY : O(N^2)**

Q3

ALGORITHM

merge-and-count(A,B)  
    A,B two input lists (sorted)  
    C  output list  
    i,j current pointers to each list, start at beginning  
    a\_i, b\_j elements pointed by i, j  
    count number of inversion, initially 0  
  
  while A,B != empty  
    append min(a\_i,b\_j) to C  
    if b\_j < a\_i  
       count += number of element remaining in A  
       j++  
    else  
       i++  
  now one list is empty  
  append the remainder of the list to C  
  return count, C

**CODE**

#include <iostream>

#include <vector>

using namespace std;

int main()

{

    int n;

    cout << "Enter number of elements in the array" << endl;

    cin >> n;

    vector<int> vec1;

    for (int i = 0; i < n; i++)

    {

        int a;

        cin >> a;

        vec1.push\_back(a);

    }

    int count = 0;

    cout << "Finding the inversion elements we get" << endl;

    for (int i = 0; i < n - 1; i++)

    {

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

        {

            if (vec1[i] > vec1[j])

            {

                cout << "(" << i + 1 << "," << j + 1 << ")" << endl;

                count++;

            }

        }

    }

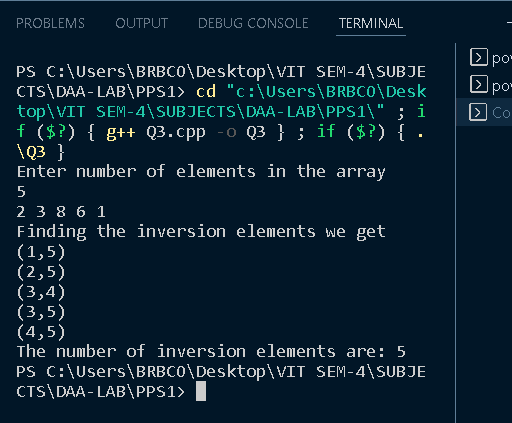
    cout << "The number of inversion elements are: ";

    cout << count;

    return 0;

}

**OUTPUT**



**TIME COMPLEXITY : O(N^2)**

**INSERTION SORT TIME COMPLEXITY : O(N^2)**

Q4

ALGORITHM

* Using two loops iterate through to all the element till be reach greater than n.
* Now till both the roots satisfy , increment the count
* The count is the number of pairs , satisfying the given inequality

**CODE**

#include <iostream>

using namespace std;

int Sol(int n)

{

    int ans = 0;

    for (int x = 0; x \* x < n; x++)

        for (int y = 0; x \* x + y \* y < n; y++)

            ans++;

    return ans;

}

int main()

{

    int n;

    cout << "Enter value n: ";

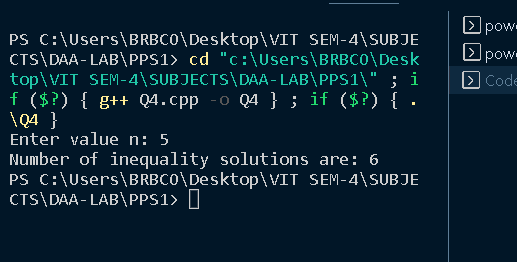
    cin >> n;

    cout << "Number of inequality solutions are: " << Sol(n) << endl;

    return 0;

}

**OUTPUT**



Q9

ALGORITHM

insertionSort(array)

mark first element as sorted

for each unsorted element X

'extract' the element X

for j <- lastSortedIndex down to 0

if current element j > X

move sorted element to the right by 1

break loop and insert X here

end insertionSort

* Now for binary search, if left pointer if less than right pointer we make left as mid-1 if target is less than mid
* And make right as mid+1 if target is greater than mid
* If left pointer and right pointer have crossed over , if target is greater than left pointer value return left+1
* Else return s

CODE

#include <iostream>

#include <vector>

#include <iterator>

#include <sstream>

#include <string>

using namespace std;

int BinarySearch(vector<int> &a, int s, int d, int t)

{

    if (s < d)

    {

        int mid = s + (d - s) / 2;

        if (a[mid] == t)

            return mid;

        else if (t < a[mid])

            return BinarySearch(a, s, mid - 1, t);

        else

            return BinarySearch(a, mid + 1, d, t);

    }

    else if (t > a[s])

        return s + 1;

    else

        return s;

}

int main()

{

    string input;

    getline(cin, input);

    istringstream is(input);

    vector<int> a((istream\_iterator<int>(is)), std::istream\_iterator<int>());

    int n = a.size();

    for (int i = 1; i < n; i++)

    {

        int key = a[i];

        int j = i - 1;

        int index = BinarySearch(a, 0, j, key);

        while (j >= index)

        {

            a[j + 1] = a[j];

            j--;

        }

        a[j + 1] = key;

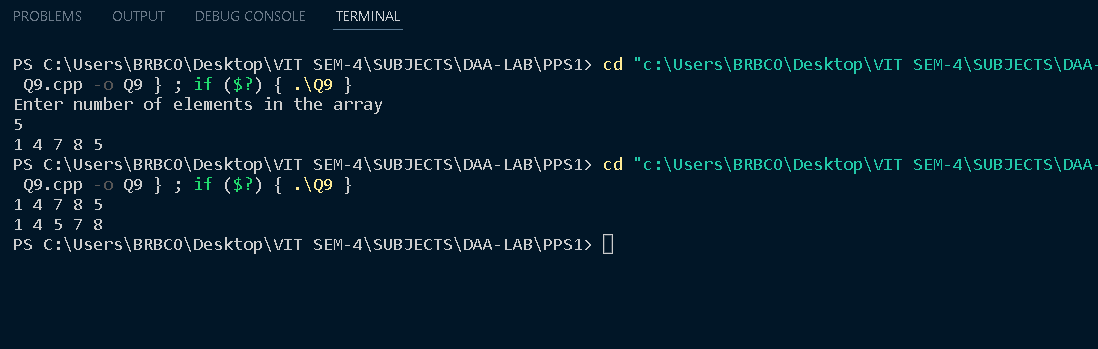
    }

    for (auto i : a)

        cout << i << ' ';

}

OUTPUT



TIME COMPLEXITY : O(N^2LOGN)