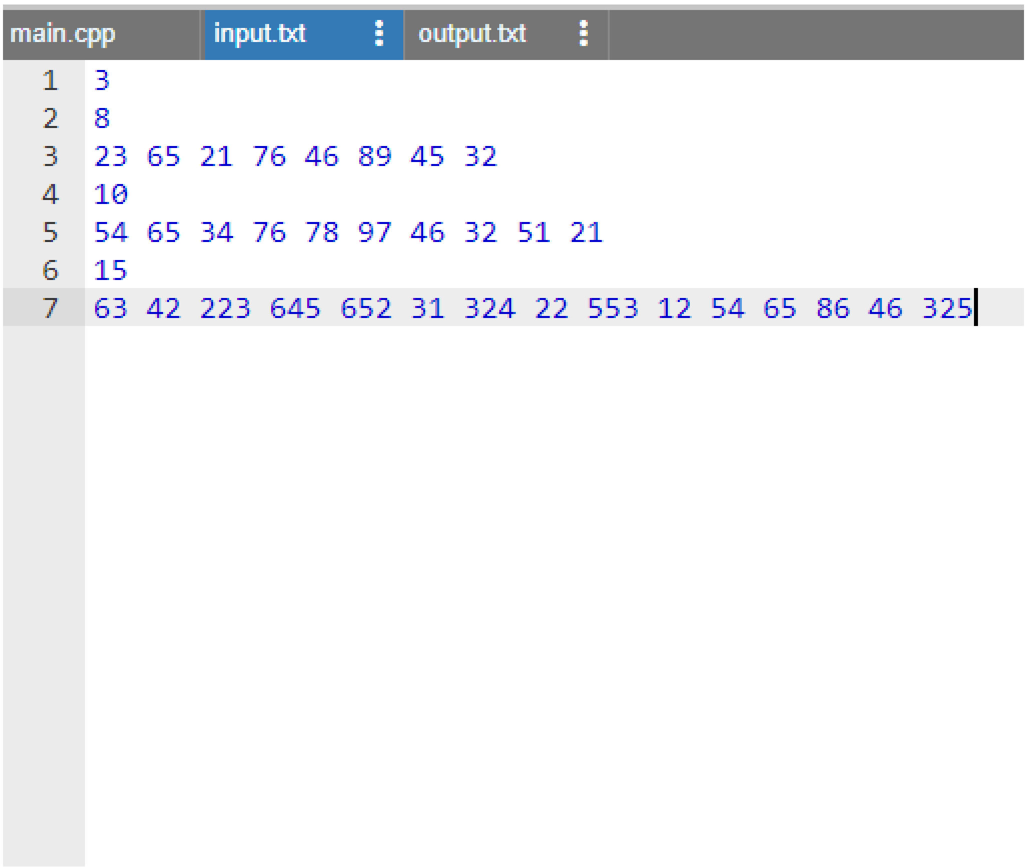
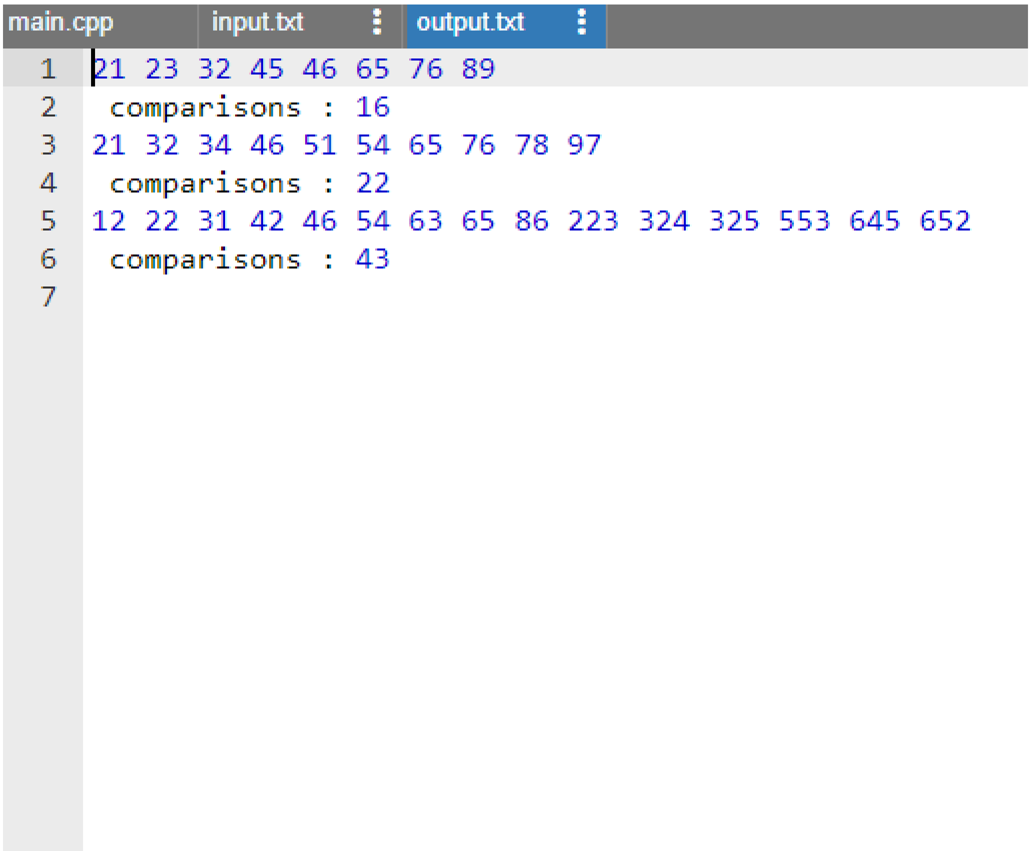
WEEK 4

1.Given an unsorted array of integers, design an algorithm and implement it using a program to sort an array of elements by dividing the array into two subarrays and combining these subarrays after sorting each one of them. Your program should also find number of comparisons and inversions during sorting the array

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

void mergeArray(vector<int>&arr, int l, int mid, int h,int &c) {
                                                                        // array size
    int n1 = mid - l + 1;
    int n2 = h - mid;
    int a[n1], b[n2];
    for(int i = 0; i < n1; i++) {
        a[i] = arr[l + i];
    }
    for(int i = 0; i < n2; i++) {
        b[i] = arr[mid + 1 + i];
    }
    int i = 0, j = 0, k = l;
                                                                        //sorting
    while(i < n1 && j < n2) {
        c++;
        if(a[i] <= b[j]) {
            arr[k] = a[i];
            i++;
        }
        else {
            arr[k] = b[j];
            j++;
        }
        k++;
    }
    while(i < n1) {
        arr[k] = a[i];
        i++; k++;
    }
    while(j < n2) {
        arr[k] = b[j];
        j++; k++;
    }
}

void mergeSort(vector<int>&arr, int l, int h,int &c) {
    if(l < h) {
        int mid = l + (h - l) / 2;
        mergeSort(arr, l, mid,c);
        mergeSort(arr, mid + 1, h,c);
        mergeArray(arr, l, mid, h,c);
    }
}



int main() {
                                                                  // Open input and output files
    ifstream fin("input.txt");
    ofstream fout("output.txt");
    // Check if files are opened successfully
    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }
    int t;
 // Number of test cases
    fin >> t;
    while (t--) {
        int n;
        fin >> n;
        vector<int> arr(n);

                                                            // Input array elements
        for (int i = 0; i < n; i++) {
            fin >> arr[i];
        }
        int c=0;

                                                            // Perform merge sort
        mergeSort(arr,0,n-1,c);

        //Output array elements
        for(int i=0;i<n;i++){
            fout<<arr[i]<<' ';
        }
        fout<<"\n comparisons : "<<c<<endl;
    }

                                                            // Close input and output files
    fin.close();
    fout.close();
}
```



```
3
8
23 65 21 76 46 89 45 32
10
54 65 34 76 78 97 46 32 51 21
15
63 42 223 645 652 31 324 22 553 12 54 65 86 46 325
```



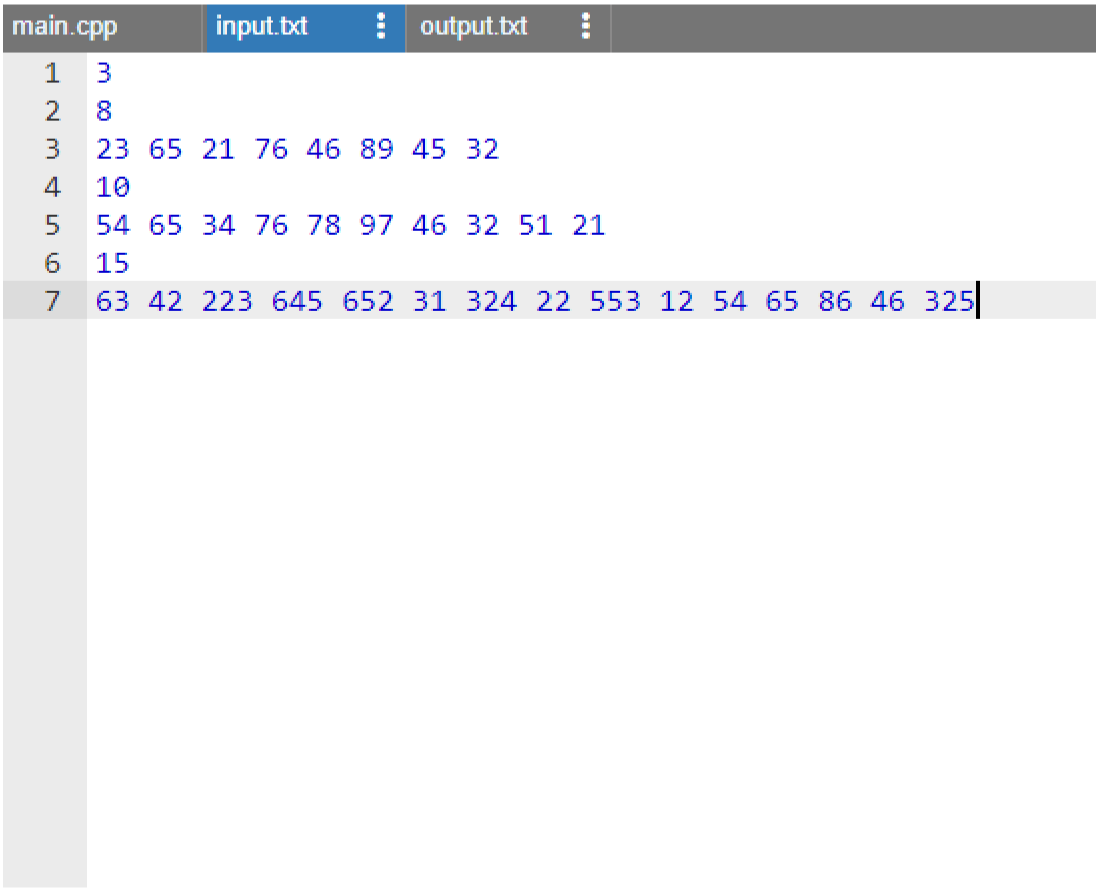
```
21 23 32 45 46 65 76 89
 comparisons : 16
21 32 34 46 51 54 65 76 78 97
 comparisons : 22
12 22 31 42 46 54 63 65 86 223 324 325 553 645 652
 comparisons : 43
```

2 . Given an unsorted array of integers, design an algorithm and implement it using a program to sort an array of elements by partitioning the array into two subarrays based on a pivot element such that one of the sub array holds values smaller than the pivot element while another sub array holds values greater than the pivot element. Pivot element should be selected randomly from the array. Your program should also find number of comparisons and swaps required for sorting the array.

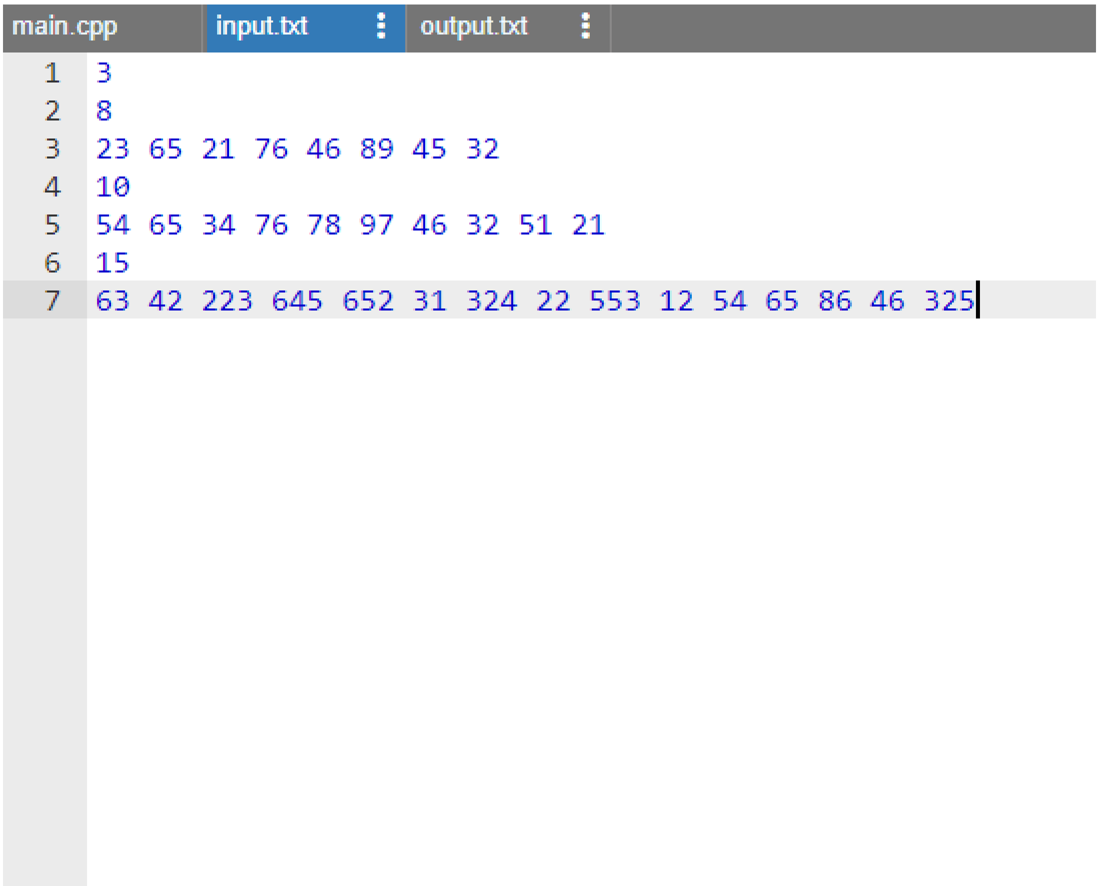
```
#include <iostream>
#include <vector>
#include <fstream>
using namespace std;
                                                                //partition
int partition(vector<int>&arr, int l, int h,int &c,int &s) {
   int pivot = arr[h];
   int i = l - 1;
   for(int j = l; j <= h - 1; j++) {
      c++;
      if(arr[j] < pivot) {
         i++;
         s++;
         swap(arr[i], arr[j]);
      }
   }
   s++;
   swap(arr[i + 1], arr[h]);
   return i + 1;
}
void quickSort(vector<int>&arr, int l, int h,int &c,int &s) {
   if(l < h) {
      int pivot = partition(arr, l, h,c,s);
      quickSort(arr, l, pivot - 1,c,s);
      quickSort(arr, pivot + 1, h,c,s);
   }
}
```

```
int main() {
                                                        // Open input and output files
    ifstream fin("input.txt");
    ofstream fout("output.txt");
                                                  // Check if files are opened successfully
    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }
    int t;
                                                                  // Number of test cases
    fin >> t;
    while (t--) {

        int n;
        fin >> n;
        vector<int> arr(n);

                                                                 // Input array elements
        for (int i = 0; i < n; i++) {
            fin >> arr[i];
        }
        int c=0,s=0;

                                                                  // Perform quicke sort
        quickSort(arr,0,n-1,c,s);

                                                                 //Output array elements
        for(int i=0;i<n;i++){
            fout<<arr[i]<<' ';
        }
        fout<<"\ncomparisons : "<<c<<endl;
        fout<<"swaps : "<<s<<endl;
    }
```

```
                                                              // Close input and output files
    fin.close();
    fout.close();
}
```



```
3
8
23 65 21 76 46 89 45 32
10
54 65 34 76 78 97 46 32 51 21
15
63 42 223 645 652 31 324 22 553 12 54 65 86 46 325
```

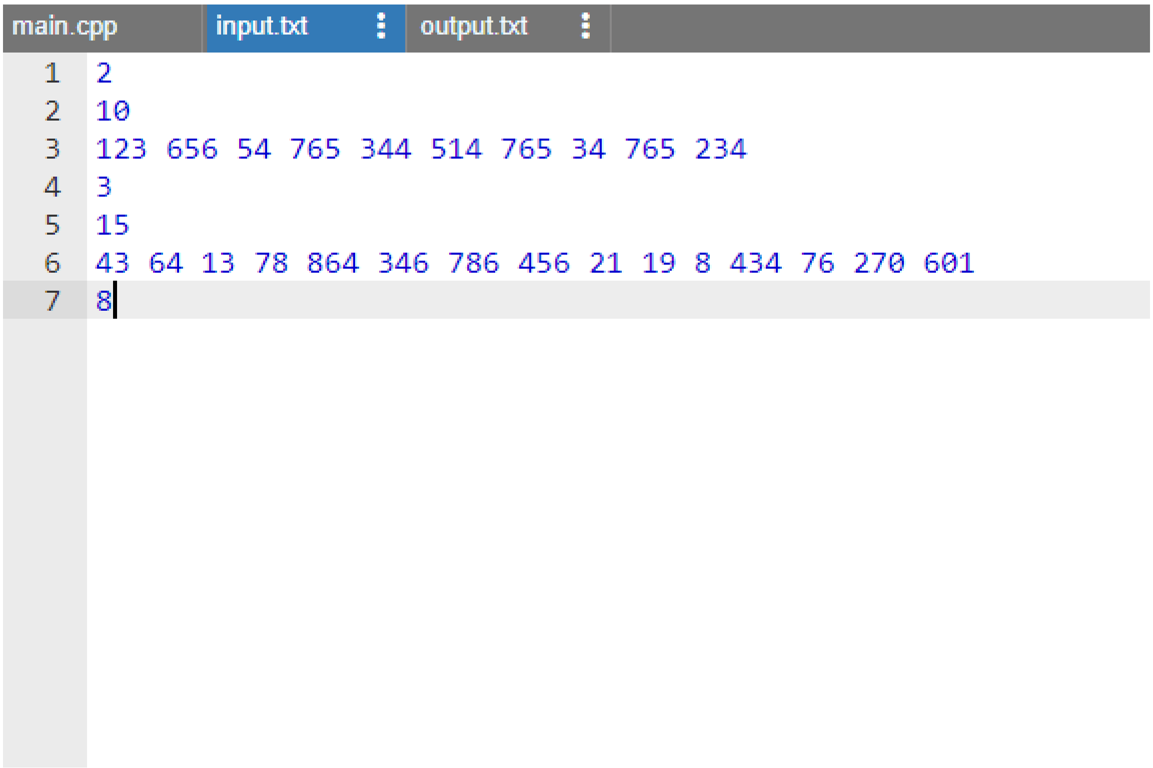


```
3
8
23 65 21 76 46 89 45 32
10
54 65 34 76 78 97 46 32 51 21
15
63 42 223 645 652 31 324 22 553 12 54 65 86 46 325
```

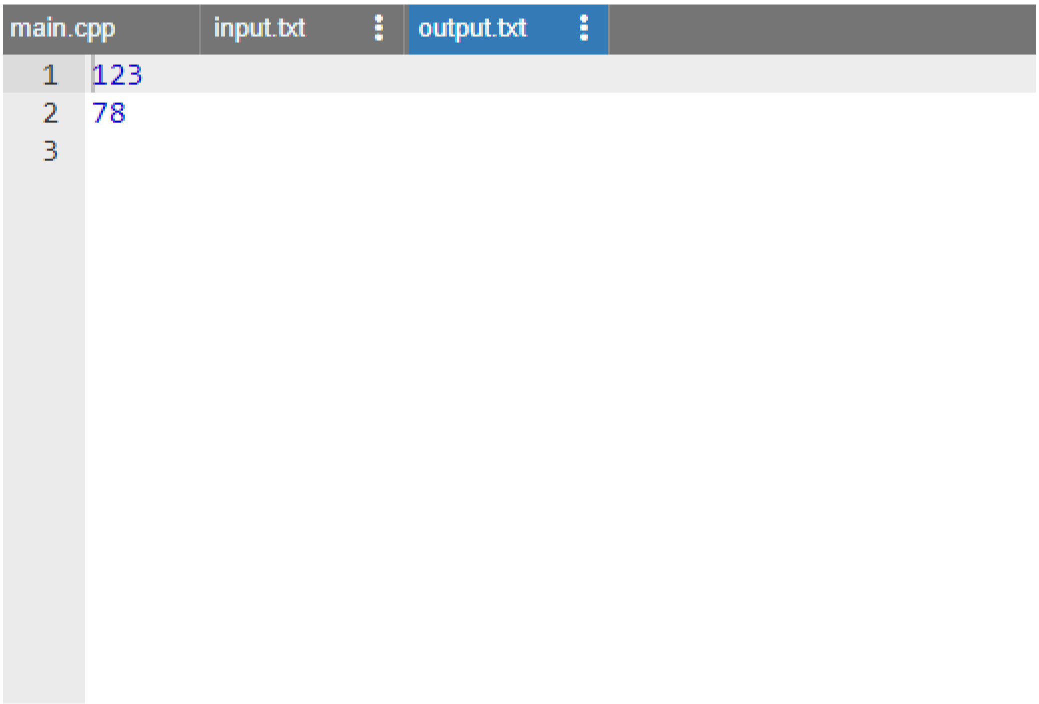
3 . Given an unsorted array of integers, design an algorithm and implement it using a program to find Kth smallest or largest element in the array. (Worst case Time Complexity = O(n))

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



                                                                       //function to find the kth largest
void kthlargest(vector<int>& arr,int k,int n,ofstream &fout) {

   priority_queue<int> pq;
   for(int i = 0; i < k; i++) {
      pq.push(arr[i]);
   }
   for(int i = k; i < n; i++) {
      pq.push(arr[i]);
      if(pq.size() > k) pq.pop();
   }
   if(pq.empty()) fout<<"Not present"<<endl;
   else fout<<pq.top()<<endl;

}
```

```
int main() {
                                                        // Open input and output files
    ifstream fin("input.txt");
    ofstream fout("output.txt");
                                                // Check if files are opened successfully
    if (!fin.is_open() || !fout.is_open()) {
        cout << "Error occurred while opening files.\n";
        return 0;
    }
    int t;
                                                              // Number of test cases
    fin >> t;
    while (t--) {

        int n;
        fin >> n;
        vector<int> arr(n);

                                                              // Input array elements
        for (int i = 0; i < n; i++) {
            fin >> arr[i];
        }
        int k;
        fin>>k;

                                                               // finding kth largest
        kthlargest(arr,k,n,fout);
    }
                                                        // Close input and output files
    fin.close();
    fout.close();
}
```



```
2
10
123 656 54 765 344 514 765 34 765 234
3
15
43 64 13 78 864 346 786 456 21 19 8 434 76 270 601
8
```



```
123
78
```