Experiment -2

AIM

To implement following algorithms using array as data structure and analyse its time complexity

1.) Merge Sort

2.) Quick Sort

theory

→ Merge Sort

Algorithm

```
merge - sort (A, p, r)
 if $p < r$
     $q = [(p/r) / 2]$
     merge-sort (A, p, q)
     merge-sort (A, q+1, r)
     merge (A, p, q, r)
merge (A, p, q, r)
 $n_1 = q-p+1$
 $n_2 = r-q$
let L [1... $n_1$+1] and R [1$n_2$+1] be the new arrays
 for i=1 to $n_1$
    $L[i] = A[p+i-1]$
  for j=1 to $n_2$
      $R[j] = A[q+j]$
      $L[n_1+1] = \infty$
      $R[n_2+1] = \infty$
```

```
i = 1
j = 1
for k = p to r
if  L[i] ≤ R[j]
    A[k] = L[i]
    i = i+1
  else A[k] = R[j]
       j = j+1
```

Time Complexity Analysis

Case I · Best Case Complexity

when array is sorted

$\Rightarrow O(n \log n)$

Case II : Worst Case Time complexity

$\Rightarrow O(n \log n)$

Case III : Average Case time complexity

$\Rightarrow O(n \log n)$

→ Quick Sort

Algorethm

```
  quicksort ( arr, p, r)
  If (p<r)
     q = partitia (arr, p, r)
    Quicksort (arr, p, q-1)
    Quicksort (arr, q+1, r)

partition ( arr, p, r)
  x = arr [ r ]
  i = p-1
  for j = p, ... r
  If arr [j] <= x
      i = i + 1
     temp = arr [i]
     arr [i] = arr [j]
     arr [j] = temp
 temp = arr [i+1]
  arr [i+1] = arr [ r ]
  arr [r] = temp
```

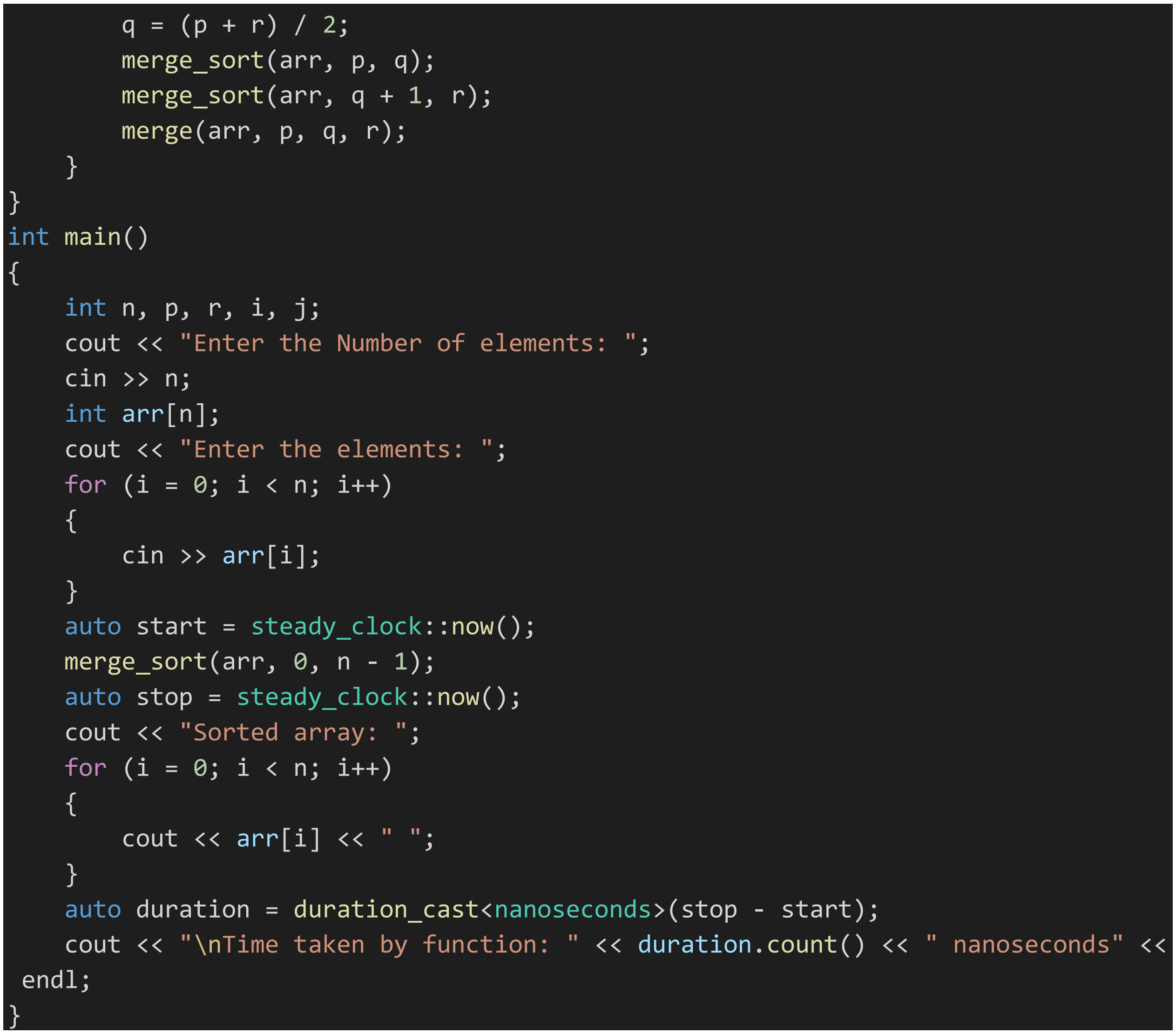
→ Time Complexity Analysis

- Best case time complexity i.e array is sorted = O n log (n)
- Worst case time complexity i.e array is reverse sorted = O(n2)
- Average case time complexity i.e array is partially sorted = On(log(n))

Result Merge sort and Quick sort was successfully Implemented

Code (Merge Sort)

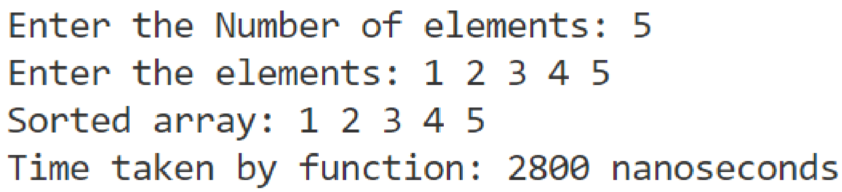
```
#include <iostream>
#include <bits/stdc++.h>
#include <chrono>
using namespace std;
using namespace std::chrono;
void merge(int arr[], int p, int q, int r)
{
    int n1, n2, i, j, k;
    n1 = q - p + 1;
    n2 = r - q;
    int left[n1 + 1], right[n2 + 1];
    for (i = 1; i < n1 + 1; i++)
    {
        left[i] = arr[p + i - 1];
    }
    for (i = 1; i < n2 + 1; i++)
    {
        right[i] = arr[q + i];
    }
    left[n1 + 1] = 1000000;
    right[n2 + 1] = 100000;
    i = 1;
    j = 1;
    for (k = p; k < r + 1; k++)
    {
        if (left[i] <= right[j])
        {
            arr[k] = left[i];
            i += 1;
        }
        else
        {
            arr[k] = right[j];
            j += 1;
        }
    }
}
void merge_sort(int arr[], int p, int r)
{
    int q;
    if (p < r)
    {
```



```
        q = (p + r) / 2;
        merge_sort(arr, p, q);
        merge_sort(arr, q + 1, r);
        merge(arr, p, q, r);
    }
}
int main()
{
    int n, p, r, i, j;
    cout << "Enter the Number of elements: ";
    cin >> n;
    int arr[n];
    cout << "Enter the elements: ";
    for (i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    auto start = steady_clock::now();
    merge_sort(arr, 0, n - 1);
    auto stop = steady_clock::now();
    cout << "Sorted array: ";
    for (i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    auto duration = duration_cast<nanoseconds>(stop - start);
    cout << "\nTime taken by function: " << duration.count() << " nanoseconds" <<
 endl;
}
```

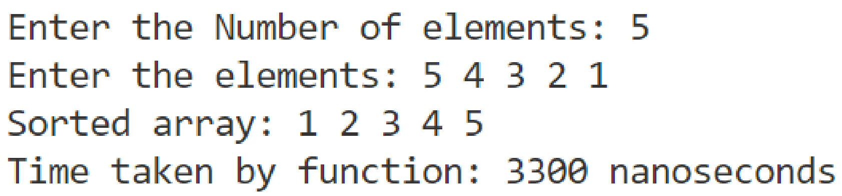
Output (Merge Sort)

Best Case



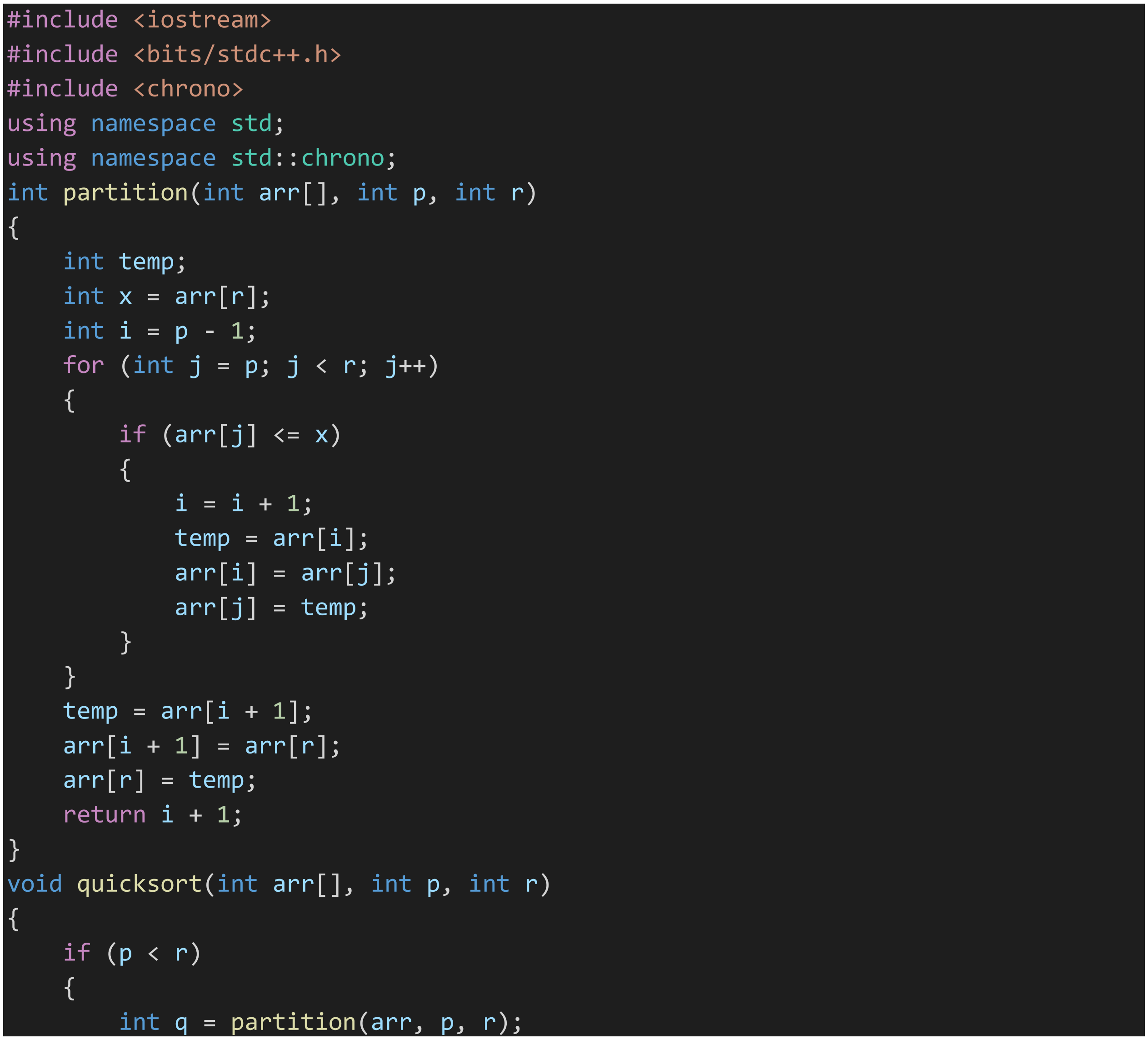
```
Enter the Number of elements: 5
Enter the elements: 1 2 3 4 5
Sorted array: 1 2 3 4 5
Time taken by function: 2800 nanoseconds
```

Worst Case

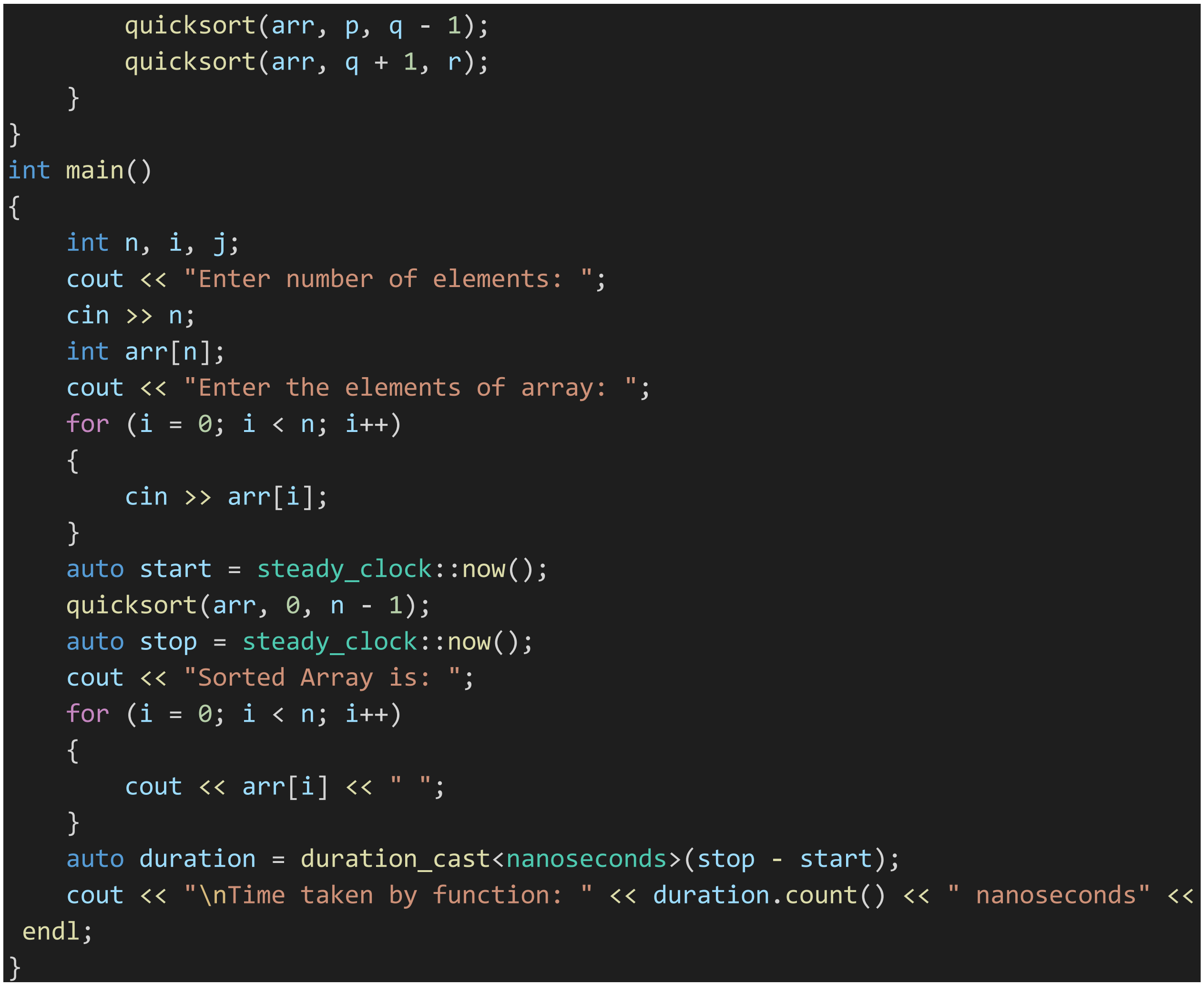


```
Enter the Number of elements: 5
Enter the elements: 5 4 3 2 1
Sorted array: 1 2 3 4 5
Time taken by function: 3300 nanoseconds
```

Code (Quick Sort)



```
#include <iostream>
#include <bits/stdc++.h>
#include <chrono>
using namespace std;
using namespace std::chrono;
int partition(int arr[], int p, int r)
{
    int temp;
    int x = arr[r];
    int i = p - 1;
    for (int j = p; j < r; j++)
    {
        if (arr[j] <= x)
        {
            i = i + 1;
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    temp = arr[i + 1];
    arr[i + 1] = arr[r];
    arr[r] = temp;
    return i + 1;
}
void quicksort(int arr[], int p, int r)
{
    if (p < r)
    {
        int q = partition(arr, p, r);
```



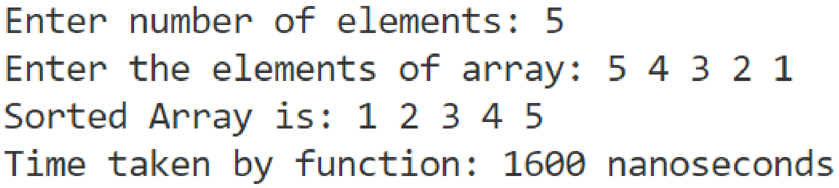
```
        quicksort(arr, p, q - 1);
        quicksort(arr, q + 1, r);
    }
}
int main()
{
    int n, i, j;
    cout << "Enter number of elements: ";
    cin >> n;
    int arr[n];
    cout << "Enter the elements of array: ";
    for (i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    auto start = steady_clock::now();
    quicksort(arr, 0, n - 1);
    auto stop = steady_clock::now();
    cout << "Sorted Array is: ";
    for (i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    auto duration = duration_cast<nanoseconds>(stop - start);
    cout << "\nTime taken by function: " << duration.count() << " nanoseconds" <<
 endl;
}
```

Output (Quick Sort)

Best Case

```
Enter number of elements: 5
Enter the elements of array: 1 2 3 4 5
Sorted Array is: 1 2 3 4 5
Time taken by function: 1200 nanoseconds
```

Worst Case



```
Enter number of elements: 5
Enter the elements of array: 5 4 3 2 1
Sorted Array is: 1 2 3 4 5
Time taken by function: 1600 nanoseconds
```