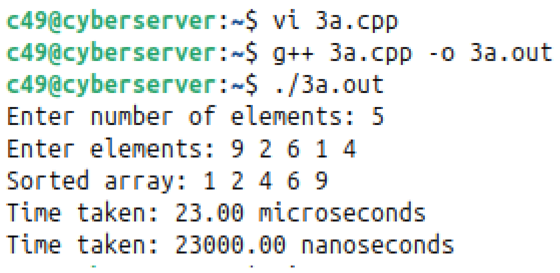
Program 3

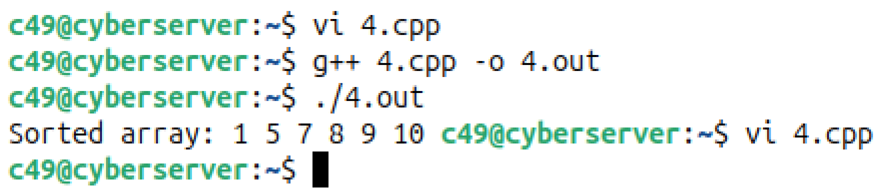
```
#include<iostream>
#include<iomanip>
#include<time.h>
#include<vector>

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> leftarr(n1);
	vector<int> rightarr(n2);

	for(int i=0;i<n1;i++){
		leftarr[i]=arr[left+i];
	}
	for(int j=0;j<n2;j++){
		rightarr[j]=arr[mid+1+j];
	}

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

	while(i<n1 && j<n2){
		arr[k++]=(leftarr[i]<rightarr[j])? leftarr[i++]: rightarr[j++];
	}

	while(i<n1){
		arr[k++]=leftarr[i++];
	}

	while(j<n2){
		arr[k++]=rightarr[j++];
	}

}

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

	mergesort(arr,left,mid);
	mergesort(arr,mid+1,right);
	merge(arr,left,mid,right);
}

int main(){
	int n;
	cout<<"Merge Sort"<<endl;
	cout<<"Enter no of elements"<<endl;
	cin>>n;
	vector<int> arr(n);

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

	clock_t start_time = clock();

	mergesort(arr,0,n-1);

	clock_t end_time =clock();

	cout<<"Sorted Array"<<endl;

	for(int i=0;i<n;i++){
		cout<<arr[i]<<" ";
	}

	double cpu_time = (double(end_time-start_time)/CLOCKS_PER_SEC)*1e9;

	    cout << "Time taken: " << cpu_time << " nanoseconds\n";
	return 0;
}
```



```
c49@cyberserver:~$ vi 3a.cpp
c49@cyberserver:~$ g++ 3a.cpp -o 3a.out
c49@cyberserver:~$ ./3a.out
Enter number of elements: 5
Enter elements: 9 2 6 1 4
Sorted array: 1 2 4 6 9
Time taken: 23.00 microseconds
Time taken: 23000.00 nanoseconds
```

Program 4

```
#include <iostream>
#include<time.h>
#include <vector>

using namespace std;

int partition(vector<int> &arr, int low, int high) {
    int pivot = arr[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            swap(arr[++i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;
}

void quickSort(vector<int> &arr, int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int main() {
    vector<int> arr = {10, 7, 8, 9, 1, 5};
    int n = arr.size();
clock_t start_time = clock();
    quickSort(arr, 0, n - 1);
        clock_t end_time =clock();


    cout << "Sorted array: ";
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
        double cpu_time = (double(end_time-start_time)/CLOCKS_PER_SEC)*1e9;

            cout << "Time taken: " << cpu_time << " nanoseconds\n";

return 0;
}

~
~
~
```

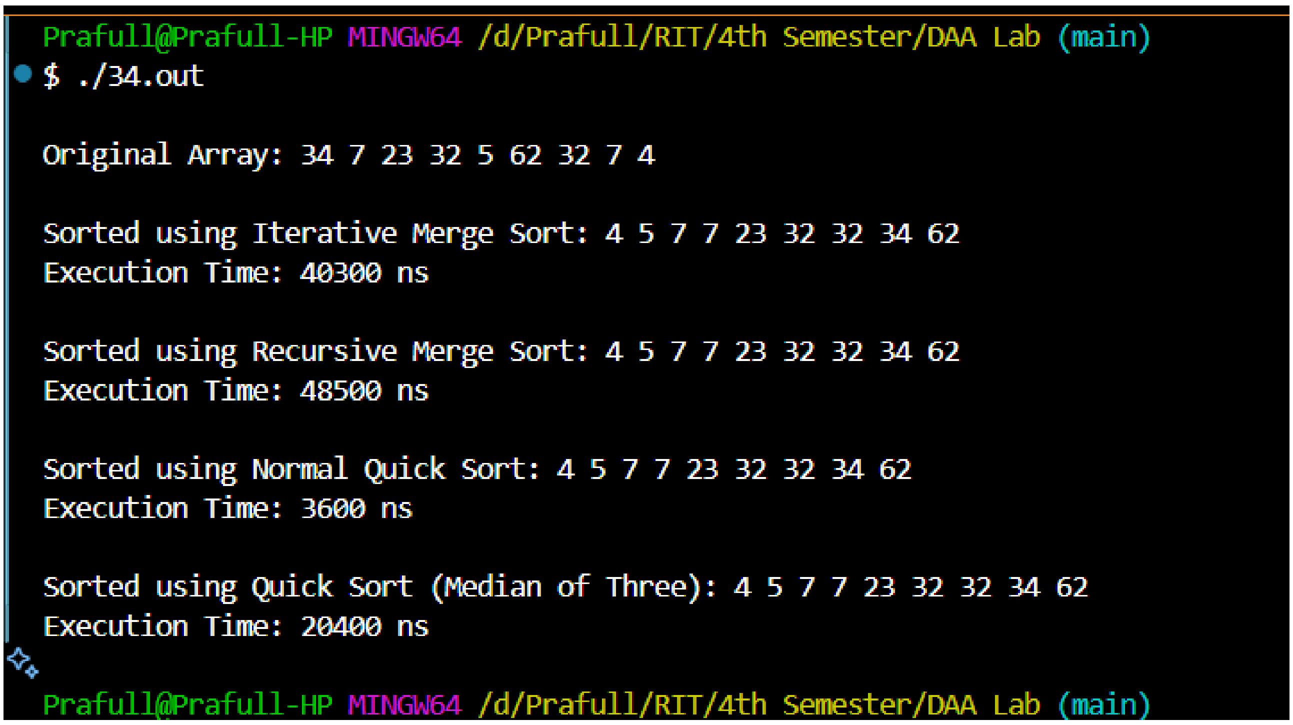


```
c49@cyberserver:~$ vi 4.cpp
c49@cyberserver:~$ g++ 4.cpp -o 4.out
c49@cyberserver:~$ ./4.out
Sorted array: 1 5 7 8 9 10 c49@cyberserver:~$ vi 4.cpp
c49@cyberserver:~$ 
```

Using Median Of 3 and including all cases of 4 and 4

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>

using namespace std;
using namespace std::chrono;

void printArray(const vector<int>& arr) {
    for (int num : arr)
        cout << num << " ";
    cout << endl;
}

void merge(vector<int>& arr, int left, int mid, int right) {
    int n1 = mid - left + 1, n2 = right - mid;
    vector<int> L(n1), R(n2);

    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];

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2)
        arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];

    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];
}

void mergeSortRecursive(vector<int>& arr, int left, int right) {
    if (left >= right) return;
    int mid = left + (right - left) / 2;
    mergeSortRecursive(arr, left, mid);
    mergeSortRecursive(arr, mid + 1, right);
    merge(arr, left, mid, right);
}

void mergeSortIterative(vector<int>& arr) {
    int n = arr.size();
    for (int size = 1; size < n; size *= 2) {
        for (int left = 0; left < n - 1; left += 2 * size) {
            int mid = min(left + size - 1, n - 1);
            int right = min(left + 2 * size - 1, n - 1);
            merge(arr, left, mid, right);
        }
    }
}

int partition(vector<int>& arr, int low, int high) {
    int pivot = arr[low], i = low + 1, j = high;
    while (i <= j) {
        while (i <= j && arr[i] <= pivot) i++;
        while (i <= j && arr[j] > pivot) j--;
        if (i < j) swap(arr[i], arr[j]);
    }
    swap(arr[low], arr[j]);
    return j;
}
```

```cpp
void quickSortNormal(vector<int>& arr, int low, int high) {
    if (low < high) {
        int pivotIndex = partition(arr, low, high);
        quickSortNormal(arr, low, pivotIndex - 1);
        quickSortNormal(arr, pivotIndex + 1, high);
    }
}

int medianOfThree(vector<int>& arr, int low, int high) {
    int mid = low + (high - low) / 2;
    vector<int> candidates = {arr[low], arr[mid], arr[high]};
    sort(candidates.begin(), candidates.end());
    return candidates[1];
}

int partitionModified(vector<int>& arr, int low, int high) {
    int pivot = medianOfThree(arr, low, high);
    int pivotIndex = (arr[low] == pivot) ? low : (arr[high] == pivot) ? high : low + (high - low) / 2;
    swap(arr[pivotIndex], arr[low]);

    return partition(arr, low, high);
}

void quickSortModified(vector<int>& arr, int low, int high) {
    if (low < high) {
        int pivotIndex = partitionModified(arr, low, high);
        quickSortModified(arr, low, pivotIndex - 1);
        quickSortModified(arr, pivotIndex + 1, high);
    }
}

int main() {
    vector<int> originalArr = {34, 7, 23, 32, 5, 62, 32, 7, 4};

    vector<int> arr = originalArr;
    cout << "\nOriginal Array: ";
    printArray(arr);
    auto start = high_resolution_clock::now();
    mergeSortIterative(arr);
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<nanoseconds>(stop - start);

    cout << "\nSorted using Iterative Merge Sort: ";
    printArray(arr);
    cout << "Execution Time: " << duration.count() << " ns\n";

    arr = originalArr;
    start = high_resolution_clock::now();
    mergeSortRecursive(arr, 0, arr.size() - 1);
    stop = high_resolution_clock::now();
    duration = duration_cast<nanoseconds>(stop - start);
    cout << "\nSorted using Recursive Merge Sort: ";
    printArray(arr);
    cout << "Execution Time: " << duration.count() << " ns\n";

    arr = originalArr;
    start = high_resolution_clock::now();
    quickSortNormal(arr, 0, arr.size() - 1);
    stop = high_resolution_clock::now();
    duration = duration_cast<nanoseconds>(stop - start);

    cout << "\nSorted using Normal Quick Sort: ";
    printArray(arr);
    cout << "Execution Time: " << duration.count() << " ns\n";

    arr = originalArr;
    start = high_resolution_clock::now();
    quickSortModified(arr, 0, arr.size() - 1);
    stop = high_resolution_clock::now();
    duration = duration_cast<nanoseconds>(stop - start);

    cout << "\nSorted using Quick Sort (Median of Three): ";
    printArray(arr);
    cout << "Execution Time: " << duration.count() << " ns\n";

    return 0;
}
```



```
Prafull@Prafull-HP MINGW64 /d/Prafull/RIT/4th Semester/DAA Lab (main)
$ ./34.out

Original Array: 34 7 23 32 5 62 32 7 4

Sorted using Iterative Merge Sort: 4 5 7 7 23 32 32 34 62
Execution Time: 40300 ns

Sorted using Recursive Merge Sort: 4 5 7 7 23 32 32 34 62
Execution Time: 48500 ns

Sorted using Normal Quick Sort: 4 5 7 7 23 32 32 34 62
Execution Time: 3600 ns

Sorted using Quick Sort (Median of Three): 4 5 7 7 23 32 32 34 62
Execution Time: 20400 ns

Prafull@Prafull-HP MINGW64 /d/Prafull/RIT/4th Semester/DAA Lab (main)
```

I have changed the library to chrono as time.h was giving 0 ns for quicksort.