

```

// C++ program for implementation
// of Bubble sort
#include <bits/stdc++.h>
using namespace std;
#include <chrono>
using namespace std::chrono;

#include<bits/stdc++.h>
using namespace std;

// A utility function to swap the values pointed by
// the two pointers
void swapValue(int *a, int *b)
{
    int *temp = a;
    a = b;
    b = temp;
    return;
}

/* Function to sort an array using insertion sort*/
void InsertionSort(int arr[], int *begin, int *end)
{
    // Get the left and the right index of the subarray
    // to be sorted
    int left = begin - arr;
    int right = end - arr;

    for (int i = left+1; i <= right; i++)
    {
        int key = arr[i];
        int j = i-1;

        /* Move elements of arr[0..i-1], that are
           greater than key, to one position ahead
           of their current position */
        while (j >= left && arr[j] > key)
        {
            arr[j+1] = arr[j];
            j = j-1;
        }
        arr[j+1] = key;
    }

    return;
}

// A function to partition the array and return
// the partition point
int* Partition(int arr[], int low, int high)
{
    int pivot = arr[high];      // pivot
    int i = (low - 1); // Index of smaller element

    for (int j = low; j <= high- 1; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot)

```

```

    {
        // increment index of smaller element
        i++;
    }
    swap(arr[i], arr[j]);
}
swap(arr[i + 1], arr[high]);
return (arr + i + 1);
}

// A function that find the middle of the
// values pointed by the pointers a, b, c
// and return that pointer
int *MedianOfThree(int * a, int * b, int * c)
{
    if (*a < *b && *b < *c)
        return (b);

    if (*a < *c && *c <= *b)
        return (c);

    if (*b <= *a && *a < *c)
        return (a);

    if (*b < *c && *c <= *a)
        return (c);

    if (*c <= *a && *a < *b)
        return (a);

    if (*c <= *b && *b <= *a)
        return (b);
}

// A Utility function to perform intro sort
void IntrosortUtil(int arr[], int * begin,
                    int * end, int depthLimit)
{
    // Count the number of elements
    int size = end - begin;

    // If partition size is low then do insertion sort
    if (size < 16)
    {
        InsertionSort(arr, begin, end);
        return;
    }

    // If the depth is zero use heapsort
    if (depthLimit == 0)
    {
        make_heap(begin, end+1);
        sort_heap(begin, end+1);
        return;
    }

    // Else use a median-of-three concept to

```

```

// find a good pivot
int * pivot = MedianOfThree(begin, begin+size/2, end);

// Swap the values pointed by the two pointers
swapValue(pivot, end);

// Perform Quick Sort
int * partitionPoint = Partition(arr, begin-arr, end-arr);
IntrosortUtil(arr, begin, partitionPoint-1, depthLimit - 1);
IntrosortUtil(arr, partitionPoint + 1, end, depthLimit - 1);

return;
}

/* Implementation of introsort*/
void Introsort(int arr[], int *begin, int *end)
{
    int depthLimit = 2 * log(end-begin);

    // Perform a recursive Introsort
    IntrosortUtil(arr, begin, end, depthLimit);

    return;
}

// A utility function to print an array of size n
void printArray(int arr[], int n)
{
    for (int i=0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++)

        // Last i elements are already
        // in place
        for (j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1])
                swap(arr[j], arr[j + 1]);
}

// Driver code
int main()
{
    int sz = 1000000;
    int Arr[sz];
    for(int i=0;i<sz;i++)
        Arr[i]=rand()%100;
    int n = sizeof(Arr) / sizeof(Arr[0]);
    auto start = high_resolution_clock::now();
}

```

```
//IntroSort
Introsort(Arr, Arr, Arr+n-1);
//Bubble Sort
//bubbleSort(Arr,sz);
auto stop = high_resolution_clock::now();
auto duration = duration_cast<microseconds>(stop - start);
cout << "Time: " << duration.count() << endl;

return 0;
}
```