**Code**:

#include <iostream>

#include <vector>

#include <unordered\_map>

#include <cstdlib>

#include <ctime>

#include <chrono>

using namespace std;

typedef int (\*SortFunction)(vector<int>&);

std::chrono::duration<double, std::micro> measureExecutionTime(vector<int>& arr, SortFunction sortFunction) {

vector<int> arrCopy = arr;

auto start = std::chrono::high\_resolution\_clock::now();

sortFunction(arrCopy);

auto end = std::chrono::high\_resolution\_clock::now();

//Microseconds

auto duration = std::chrono::duration\_cast<std::chrono::duration<double, std::micro>>(end - start);

return std::chrono::duration<double, std::micro>(duration.count() \* 10.0);

}

int bubble\_sort(vector<int>& arr) {

int comparisons = 0;

int n = arr.size();

for (int i = 0; i < n; i++) {

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

comparisons++;

if (arr[j + 1] < arr[j]) {

std::swap(arr[j + 1], arr[j]);

}

}

}

return comparisons;

}

int selection\_sort(vector<int>& arr) {

int comparisons = 0;

int n = arr.size();

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

int min = i;

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

comparisons++;

if (arr[j] < arr[min]) {

min = j;

}

}

if (min != i) {

std::swap(arr[i], arr[min]);

}

}

return comparisons;

}

int insertion\_sort(vector<int>& arr) {

int comparisons = 0;

int n = arr.size();

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

int key = arr[i];

int j = i - 1;

while (j >= 0) {

comparisons++;

if (arr[j] > key) {

arr[j + 1] = arr[j];

j--;

}

else {

break;

}

}

arr[j + 1] = key;

}

return comparisons;

}

void heapify(vector<int>& arr, int n, int i, int& comparisons) {

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

comparisons++;

if (left < n && arr[left] > arr[largest]) {

largest = left;

}

comparisons++;

if (right < n && arr[right] > arr[largest]) {

largest = right;

}

if (largest != i) {

std::swap(arr[i], arr[largest]);

heapify(arr, n, largest, comparisons);

}

}

int heapSort(vector<int>& arr) {

int comparisons = 0;

int n = arr.size();

for (int i = n / 2 - 1; i >= 0; --i) {

heapify(arr, n, i, comparisons);

}

for (int i = n - 1; i > 0; --i) {

std::swap(arr[0], arr[i]);

heapify(arr, i, 0, comparisons);

}

return comparisons;

}

void test\_comparisons() {

srand(time(0));

vector<vector<int>> arrayRandom = {

{42},

{17, 85},

{63, 29, 71},

{45, 92, 13, 78},

{34, 88, 51, 19, 73},

{61, 27, 84, 39, 16, 75},

{52, 98, 33, 47, 12, 68, 91},

{25, 56, 83, 14, 70, 36, 82, 15},

{49, 67, 23, 94, 58, 11, 76, 42, 89},

{27, 63, 5, 51, 38, 95, 19, 74, 31, 56},

{92, 18, 47, 83, 26, 65, 11, 39, 97, 52, 14},

{70, 45, 88, 33, 61, 7, 96, 22, 54, 81, 16, 73},

{29, 67, 13, 85, 41, 59, 94, 28, 76, 50, 32, 69, 15},

{63, 87, 21, 48, 95, 34, 72, 9, 57, 83, 25, 61, 38, 90},

{17, 53, 79, 31, 66, 12, 84, 46, 93, 28, 75, 40, 58, 22, 89},

{35, 71, 14, 62, 87, 26, 49, 95, 33, 77, 51, 19, 82, 44, 68, 11},

{73, 29, 56, 91, 38, 64, 15, 80, 42, 97, 23, 59, 85, 31, 67, 13, 88},

{45, 92, 18, 74, 36, 61, 27, 83, 50, 96, 21, 57, 89, 34, 70, 16, 52, 78},

{25, 63, 87, 41, 69, 14, 58, 93, 32, 76, 48, 81, 19, 65, 37, 84, 53, 28, 91},

{17, 72, 39, 85, 26, 61, 94, 43, 68, 15, 79, 33, 57, 88, 22, 46, 74, 30, 63, 95},

{12, 58, 83, 37, 91, 25, 69, 42, 76, 19, 54, 87, 31, 65, 93, 28, 71, 45, 82, 16, 59},

{34, 77, 21, 66, 89, 43, 97, 15, 52, 85, 29, 73, 48, 91, 26, 62, 38, 80, 13, 57, 94, 32},

{68, 41, 86, 23, 75, 49, 92, 17, 61, 35, 78, 53, 96, 28, 71, 44, 87, 19, 64, 39, 82, 56, 93},

{25, 70, 33, 88, 51, 95, 22, 67, 41, 84, 16, 59, 73, 47, 91, 29, 63, 38, 76, 12, 55, 89, 31, 68},

{42, 86, 19, 74, 57, 93, 28, 61, 35, 79, 52, 97, 24, 66, 43, 87, 15, 58, 82, 39, 71, 26, 64, 48, 91},

{17, 53, 88, 32, 76, 49, 95, 21, 65, 37, 83, 56, 92, 28, 72, 45, 89, 13, 59, 34, 77, 51, 96, 23, 68, 41},

{85, 29, 63, 38, 81, 54, 97, 16, 71, 43, 87, 25, 69, 33, 78, 52, 94, 19, 62, 36, 84, 57, 91, 27, 73, 46, 89},

{22, 66, 31, 75, 48, 93, 15, 58, 82, 39, 86, 24, 67, 41, 95, 28, 72, 35, 79, 53, 88, 26, 61, 44, 87, 19, 64, 33},

{77, 51, 96, 18, 62, 34, 83, 55, 92, 27, 71, 45, 89, 21, 65, 38, 84, 57, 93, 29, 76, 42, 87, 15, 59, 32, 78, 53, 91},

{25, 69, 43, 88, 16, 61, 35, 82, 56, 94, 28, 73, 47, 91, 22, 67, 39, 85, 52, 97, 31, 76, 44, 89, 18, 63, 37, 84, 58, 95}

};

// Create sorted arrays from 1 to N

vector<vector<int>> arraySorted;

for (int i = 1; i <= 30; i++) {

vector<int> temp;

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

temp.push\_back(j);

}

arraySorted.push\_back(temp);

}

// Create inverted arrays from N to 1

vector<vector<int>> arrayInverted;

for (int i = 1; i <= 30; i++) {

vector<int> temp;

for (int j = i; j >= 1; j--) {

temp.push\_back(j);

}

arrayInverted.push\_back(temp);

}

// Maps to store results

vector<pair<int, double>> bubble\_sort\_random(30);

vector<pair<int, double>> selection\_sort\_random(30);

vector<pair<int, double>> insertion\_sort\_random(30);

vector<pair<int, double>> heap\_sort\_random(30);

vector<pair<int, double>> bubble\_sort\_sorted(30);

vector<pair<int, double>> selection\_sort\_sorted(30);

vector<pair<int, double>> insertion\_sort\_sorted(30);

vector<pair<int, double>> heap\_sort\_sorted(30);

vector<pair<int, double>> bubble\_sort\_inverted(30);

vector<pair<int, double>> selection\_sort\_inverted(30);

vector<pair<int, double>> insertion\_sort\_inverted(30);

vector<pair<int, double>> heap\_sort\_inverted(30);

// Test random arrays

for (int i = 0; i < 30; i++) {

bubble\_sort\_random[i].first = bubble\_sort(arrayRandom[i]);

bubble\_sort\_random[i].second = measureExecutionTime(arrayRandom[i], bubble\_sort).count();

selection\_sort\_random[i].first = selection\_sort(arrayRandom[i]);

selection\_sort\_random[i].second = measureExecutionTime(arrayRandom[i], selection\_sort).count();

insertion\_sort\_random[i].first = insertion\_sort(arrayRandom[i]);

insertion\_sort\_random[i].second = measureExecutionTime(arrayRandom[i], insertion\_sort).count();

heap\_sort\_random[i].first = heapSort(arrayRandom[i]);

heap\_sort\_random[i].second = measureExecutionTime(arrayRandom[i], heapSort).count();

}

// Test sorted arrays

for (int i = 0; i < 30; i++) {

bubble\_sort\_sorted[i].first = bubble\_sort(arraySorted[i]);

bubble\_sort\_sorted[i].second = measureExecutionTime(arraySorted[i], bubble\_sort).count();

selection\_sort\_sorted[i].first = selection\_sort(arraySorted[i]);

selection\_sort\_sorted[i].second = measureExecutionTime(arraySorted[i], selection\_sort).count();

insertion\_sort\_sorted[i].first = insertion\_sort(arraySorted[i]);

insertion\_sort\_sorted[i].second = measureExecutionTime(arraySorted[i], insertion\_sort).count();

heap\_sort\_sorted[i].first = heapSort(arraySorted[i]);

heap\_sort\_sorted[i].second = measureExecutionTime(arraySorted[i], heapSort).count();

}

// Test inverted arrays

for (int i = 0; i < 30; i++) {

bubble\_sort\_inverted[i].first = bubble\_sort(arrayInverted[i]);

bubble\_sort\_inverted[i].second = measureExecutionTime(arrayInverted[i], bubble\_sort).count();

selection\_sort\_inverted[i].first = selection\_sort(arrayInverted[i]);

selection\_sort\_inverted[i].second = measureExecutionTime(arrayInverted[i], selection\_sort).count();

insertion\_sort\_inverted[i].first = insertion\_sort(arrayInverted[i]);

insertion\_sort\_inverted[i].second = measureExecutionTime(arrayInverted[i], insertion\_sort).count();

heap\_sort\_inverted[i].first = heapSort(arrayInverted[i]);

heap\_sort\_inverted[i].second = measureExecutionTime(arrayInverted[i], heapSort).count();

}

cout << "\n=== Random Array Results ===\n";

cout << "Size\tBubble Sort\t\tSelection Sort\t\tInsertion Sort\t\tHeap Sort\n";

cout << " \tComps\tTime(s)\t\tComps\tTime(s)\t\tComps\tTime(s)\t\tComps\tTime(s)\n";

for (int i = 0; i < 30; i++) {

cout << (i + 1) << "\t"

<< bubble\_sort\_random[i].first << "\t" << bubble\_sort\_random[i].second << "\t\t"

<< selection\_sort\_random[i].first << "\t" << selection\_sort\_random[i].second << "\t\t"

<< insertion\_sort\_random[i].first << "\t" << insertion\_sort\_random[i].second << "\t\t"

<< heap\_sort\_random[i].first << "\t" << heap\_sort\_random[i].second << "\n";

}

cout << "\n=== Sorted Array Results ===\n";

cout << "Size\tBubble Sort\t\tSelection Sort\t\tInsertion Sort\t\tHeap Sort\n";

cout << " \tComps\tTime(s)\t\tComps\tTime(s)\t\tComps\tTime(s)\t\tComps\tTime(s)\n";

for (int i = 0; i < 30; i++) {

cout << (i + 1) << "\t"

<< bubble\_sort\_sorted[i].first << "\t" << bubble\_sort\_sorted[i].second << "\t\t"

<< selection\_sort\_sorted[i].first << "\t" << selection\_sort\_sorted[i].second << "\t\t"

<< insertion\_sort\_sorted[i].first << "\t" << insertion\_sort\_sorted[i].second << "\t\t"

<< heap\_sort\_sorted[i].first << "\t" << heap\_sort\_sorted[i].second << "\n";

}

cout << "\n=== Inverted Array Results ===\n";

cout << "Size\tBubble Sort\t\tSelection Sort\t\tInsertion Sort\t\tHeap Sort\n";

cout << " \tComps\tTime(s)\t\tComps\tTime(s)\t\tComps\tTime(s)\t\tComps\tTime(s)\n";

for (int i = 0; i < 30; i++) {

cout << (i + 1) << "\t"

<< bubble\_sort\_inverted[i].first << "\t" << bubble\_sort\_inverted[i].second << "\t\t"

<< selection\_sort\_inverted[i].first << "\t" << selection\_sort\_inverted[i].second << "\t\t"

<< insertion\_sort\_inverted[i].first << "\t" << insertion\_sort\_inverted[i].second << "\t\t"

<< heap\_sort\_inverted[i].first << "\t" << heap\_sort\_inverted[i].second << "\n";

}

}

int main() {

test\_comparisons();

return 0;

}