//The purpose of this file is to combine all the 3 algorithms on a datasets ranging from 1000 to 50,000 and get the run time of each using the ctime library.

//In this file we will try to implement insertion sort

#define SIZE\_MAX 50000

#define RAND\_MAX 32568

#define DEFAULT\_SIZE 1000

#include <iostream>

#include <ctime>

#include <cstdlib>

#include <vector>

#include <fstream>

#include <iterator>

#include <string>

#include <chrono>

using namespace std;

using namespace std::chrono;

//quick sort function prototype

void quick\_sort(vector<int> &vect, int left, int right);

//vector<int>\* quick\_sort(vector<int> &vect);

int partition(vector<int> &vect, int left, int right);

//heap\_sort function prototype

void swap(vector<int> &vect, int i, int j);

void maxHeapify(vector<int> &vect, int heapSize, int i);

void buildMaxHeap(vector<int> &vect);

void heap\_sort(vector<int> &vect);

int parent(int i);

int left(int i);

int right(int i);

//insertion sort prototype

void insertion\_sort(vector<int> &vect);

//insertion sort

void insertion\_sort(vector<int> &vect)

{

int temp = 0;

int current = 0;

for (int i = 1; i < vect.size(); i++) {

current = vect[i];

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

if (vect[j] > current) {

temp = vect[j];

vect[j] = current;

vect[j + 1] = temp;

}

}

}

}

//for heap sort

int parent(int i) {

return(i - 1) / 2;

}

int left(int i) {

return((2 \* i) + 1);

}

int right(int i) {

return 2 \* (i + 1);

}

void swap(vector<int> &vect, int i, int j)

{

int temp = vect[j];

vect[j] = vect[i];

vect[i] = temp;

}

void buildMaxHeap(vector<int> &vect)

{

int heapSize = vect.size();

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

{

maxHeapify(vect, heapSize, i);

}

}

void heap\_sort(vector<int> &vect)

{

buildMaxHeap(vect);

int heapSize = vect.size() - 1;

while (heapSize >= 0) {

swap(vect, 0, heapSize);

maxHeapify(vect, heapSize, 0);

heapSize = heapSize - 1;

}

}

void maxHeapify(vector<int> &vect, int heapSize, int i)

{

int leftChildIndex = left(i);

int rightChildIndex = right(i);

int indexOfLargest = i;

if (leftChildIndex < heapSize && vect[leftChildIndex] > vect[i]) {

indexOfLargest = leftChildIndex;

}

/\*else {

indexOfLargest = i;

}\*/

if (rightChildIndex < heapSize && vect[rightChildIndex]> vect[indexOfLargest]) {

indexOfLargest = rightChildIndex;

}

if (indexOfLargest != i) {

swap(vect, i, indexOfLargest);

maxHeapify(vect, heapSize, indexOfLargest);

}

}

//for quicksort

int partition(vector<int> &vect, int left, int right) {

int i = left, j = right;

int tmp;

int pivot = vect[(left + right) / 2];

while (i <= j) {

while (vect[i] < pivot) {

i++;

}

while (vect[j] > pivot) {

j--;

}

if (i <= j) {

swap(vect, i, j);

i++;

j--;

}

};

return i;

}

void quick\_sort(vector<int> &vect, int left, int right) {

int index = partition(vect, left, right);

if (left < index - 1) {

quick\_sort(vect, left, index - 1);

}

if (index < right) {

quick\_sort(vect, index, right);

}

}

/\*sources: https://www.geeksforgeeks.org/measure-execution-time-function-cpp/ \*/

int main() {

srand(time(NULL));

vector<int> v;

vector<int> heapsort;

vector<int> insertionsort;

vector<int> quicksort;

v.resize(DEFAULT\_SIZE);

while (v.size() <= SIZE\_MAX) {

cout << "Iteration :" << (v.size()) / DEFAULT\_SIZE << endl;

//populating the vector with random values

for (int i = 0; i<v.size(); i++) {

v[i] = rand() % RAND\_MAX;

}

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

quick\_sort(v, 0, v.size() - 1);

auto stop = high\_resolution\_clock::now();

auto duration = duration\_cast<milliseconds>(stop - start);

quicksort.push\_back(duration.count());

cout << "The duration-time for quick sort in milliseconds is: " << duration.count() << endl;

start = high\_resolution\_clock::now();

insertion\_sort(v);

stop = high\_resolution\_clock::now();

duration = duration\_cast<milliseconds>(stop - start);

insertionsort.push\_back(duration.count());

cout << "The duration-time for insertion sort in milliseconds is: " << duration.count() << endl;

start = high\_resolution\_clock::now();

heap\_sort(v);

stop = high\_resolution\_clock::now();

duration = duration\_cast<milliseconds>(stop - start);

heapsort.push\_back(duration.count());

cout << "The duration-time for heap sort in milliseconds is: " << duration.count() << endl;

v.resize(v.size() + 1000);

cout << "\n\n";

}

cout << "\n\nHeapsort time data: ";

for (int i : heapsort) {

cout << i << ", ";

}

cout << "\n\nQuicksort time data: ";

for (int i : quicksort) {

cout << i << ", ";

}

cout << "\n\nInsertion-sort time data: ";

for (int i : insertionsort) {

cout << i << ", ";

}

return 0;

/\*for (int i : v) { //This print the entire contents of the vector

cout << i << ', ';

}\*/

}