///@file Source.cpp

#include <iostream>

#include <string>

#include <time.h>

#include <vector>

#include <algorithm> //for random\_shuffle

#include <chrono> //for chronotime

#include <math.h> // for pow and log

using namespace std;

long long int steps=0;

long long int stepCount=0;

long double cApprox = 0;

//InsertionSort algorithm

void insertionSort(vector<int> &arr)

{

int i, key, j;

stepCount += 4;

for (i = 1; i < arr.size(); i++)

{

stepCount += 2;

key = arr[i];

j = i - 1;

stepCount += 2;

while (j >= 0 && arr[j] > key)

{

stepCount += 2;

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

j = j - 1;

stepCount += 2;

}

stepCount++;

arr[j + 1] = key;

stepCount++;

}

stepCount++;

}

// merges two subarrays of array[].

void merge(vector<int> &arr, int start, int middle, int end) {

vector<int> leftArray(middle - start + 1);

vector<int> rightArray(end - middle);

steps += 2;

// fill in left array

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

{

steps += 2;

leftArray[i] = arr[start + i];

steps++;

}

// fill in right array

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

{

steps += 2;

rightArray[i] = arr[middle + 1 + i];

steps++;

}

/\* Merge the temp arrays \*/

// initial indexes of first and second subarrays

int leftIndex = 0, rightIndex = 0;

steps += 2;

// the index we will start at when adding the subarrays back into the main array

int currentIndex = start;

steps++;

// compare each index of the subarrays adding the lowest value to the currentIndex

while (leftIndex < leftArray.size() && rightIndex < rightArray.size()) {

steps += 2;

if (leftArray[leftIndex] <= rightArray[rightIndex]) {

steps++;

arr[currentIndex] = leftArray[leftIndex];

leftIndex++;

steps += 2;

}

else {

arr[currentIndex] = rightArray[rightIndex];

rightIndex++;

steps += 2;

}

currentIndex++;

steps++;

}

// copy remaining elements of leftArray[] if any

while (leftIndex < leftArray.size())

{

steps++;

arr[currentIndex++] = leftArray[leftIndex++];

steps++;

}

// copy remaining elements of rightArray[] if any

while (rightIndex < rightArray.size())

{

steps++;

arr[currentIndex++] = rightArray[rightIndex++];

steps++;

}

}

// main function that sorts array[start..end] using merge()

void mergeSort(vector<int> &arr, int start, int end) {

// base case

if (start < end) {

steps++;

// find the middle point

int middle = (start + end) / 2;

steps++;

mergeSort(arr, start, middle); // sort first half

mergeSort(arr, middle + 1, end); // sort second half

steps += 2;

// merge the sorted halves

merge(arr, start, middle, end);

steps++;

}

}

//Printing the vector arrays function

void printArray(vector<int> &arr)

{

for (int x : arr)

{

cout << x << " ";

}

}

//Generating sorted arrays for n input

void generateSort(vector<int> &arr, int n)

{

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

{

arr.push\_back(i);

}

}

//Generate reverse arrays for n input

void generateReverse(vector<int> &arr, int n)

{

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

{

arr.push\_back(i);

}

}

//Generate random permutation arrays for n input

void generateRandom(vector<int> &arr, int n)

{

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

{

arr.push\_back(i);

}

random\_shuffle(arr.begin(), arr.end());

}

int main()

{

// Array of inputs n

int inputs[8] = { 100, 200, 300, 400, 500, 1000, 4000, 10000 };

//InsertionSort for Sorted Arrays, Reverse Arrays, Random Permutation Arrays

//C approximation will be stepCount/n^2

cout << "---------------------------------INSERTION SORT----------------------------------------" << endl;

cout << "SORTED ARRAYS" << endl;

for (int i = 0; i < sizeof(inputs) / sizeof(inputs[0]); i++)

{

stepCount = 0;

cApprox = 0;

vector<int> arr;

generateSort(arr, inputs[i]);

auto t1 = chrono::high\_resolution\_clock::now();

insertionSort(arr);

auto t2 = chrono::high\_resolution\_clock::now();

chrono::duration<double, milli> timer = t2 - t1;

cApprox = (stepCount) / (pow(inputs[i], 2));

cout << arr.size() << " ELEMENTS: Steps = " << stepCount << " | Runtime = " << timer.count() << " miliseconds. | C Approximation: " << cApprox << endl;

arr.clear();

}

cout << "REVERSED ARRAYS" << endl;

for (int i = 0; i < sizeof(inputs) / sizeof(inputs[0]); i++)

{

stepCount = 0;

cApprox = 0;

vector<int> arr;

generateReverse(arr, inputs[i]);

auto t1 = chrono::high\_resolution\_clock::now();

insertionSort(arr);

auto t2 = chrono::high\_resolution\_clock::now();

chrono::duration<double, milli> timer = t2 - t1;

cApprox = (stepCount) / (pow(inputs[i], 2));

cout << arr.size() << " ELEMENTS: Steps = " << stepCount << " | Runtime = " << timer.count() << " miliseconds. | C Approximation: " << cApprox << endl;

arr.clear();

}

cout << "RANDOM PERMUTATION ARRAYS" << endl;

for (int i = 0; i < sizeof(inputs) / sizeof(inputs[0]); i++)

{

stepCount = 0;

cApprox = 0;

vector<int> arr;

generateRandom(arr, inputs[i]);

auto t1 = chrono::high\_resolution\_clock::now();

insertionSort(arr);

auto t2 = chrono::high\_resolution\_clock::now();

chrono::duration<double, milli> timer = t2 - t1;

cApprox = (stepCount) / (pow(inputs[i], 2));

cout << arr.size() << " ELEMENTS: Steps = " << stepCount << " | Runtime = " << timer.count() << " miliseconds. | C Approximation: " << cApprox << endl;

arr.clear();

}

//MergeSort for Sorted Arrays, Reverse Arrays, Random Permutation Arrays

//C approximation will be steps/n\*log(n)

cout << endl << endl << "---------------------------------MERGE SORT----------------------------------------" << endl;

cout << "SORTED ARRAYS" << endl;

for (int i = 0; i < sizeof(inputs) / sizeof(inputs[0]); i++)

{

steps = 0;

cApprox = 0;

vector<int> arr;

generateSort(arr, inputs[i]);

auto t1 = chrono::high\_resolution\_clock::now();

mergeSort(arr, 0, (arr.size() - 1));

auto t2 = chrono::high\_resolution\_clock::now();

chrono::duration<double, milli> timer = t2 - t1;

cApprox = (steps) / (inputs[i] \* log(inputs[i]));

cout << arr.size() << " ELEMENTS: Steps = " << steps << " | Runtime = " << timer.count() << " miliseconds. | C Approximation: " << cApprox << endl;

arr.clear();

}

cout << "REVERSED ARRAYS" << endl;

for (int i = 0; i < sizeof(inputs) / sizeof(inputs[0]); i++)

{

steps = 0;

cApprox = 0;

vector<int> arr;

generateReverse(arr, inputs[i]);

auto t1 = chrono::high\_resolution\_clock::now();

mergeSort(arr, 0, (arr.size() - 1));

auto t2 = chrono::high\_resolution\_clock::now();

chrono::duration<double, milli> timer = t2 - t1;

cApprox = (steps) / (inputs[i] \* log(inputs[i]));

cout << arr.size() << " ELEMENTS: Steps = " << steps << " | Runtime = " << timer.count() << " miliseconds. | C Approximation: " << cApprox << endl;

arr.clear();

}

cout << "RANDOM PERMUTATION ARRAYS" << endl;

for (int i = 0; i < sizeof(inputs) / sizeof(inputs[0]); i++)

{

steps = 0;

cApprox = 0;

vector<int> arr;

generateRandom(arr, inputs[i]);

auto t1 = chrono::high\_resolution\_clock::now();

mergeSort(arr, 0, (arr.size() - 1));

auto t2 = chrono::high\_resolution\_clock::now();

chrono::duration<double, milli> timer = t2 - t1;

cApprox = (steps) / (inputs[i] \* log(inputs[i]));

cout << arr.size() << " ELEMENTS: Steps = " << steps << " | Runtime = " << timer.count() << " miliseconds. | C Approximation: " << cApprox << endl;

arr.clear();

}

cout << endl << endl << "---------------------------------50 INSTANCES----------------------------------------" << endl;

//50 instances of Random inputs InsertionSort

cout << "50 INSTANCES INSERTION SORT" << endl;

for (int i = 0; i < sizeof(inputs) / sizeof(inputs[0]); i++)

{

stepCount = 0;

cApprox = 0;

double totalT = 0;

for (int x = 0; x < 50; x++)

{

vector<int> arr;

for (int j = 1; j <= inputs[i]; j++)

{

arr.push\_back(rand() % inputs[i] + 1);

}

auto t1 = chrono::high\_resolution\_clock::now();

insertionSort(arr);

auto t2 = chrono::high\_resolution\_clock::now();

chrono::duration<double, milli> timer = t2 - t1;

totalT += timer.count();

arr.clear();

}

cApprox = (stepCount / 50) / (pow(inputs[i],2));

cout << "50 INSTANCES OF " << inputs[i] << " ELEMENTS: Steps = " << stepCount << " | Runtime = " << totalT << " miliseconds | Average time: " << totalT/50 << " miliseconds. | C Approximation: " << cApprox << endl;

}

//50 instances of random inputs mergeSort

cout << "50 INSTANCES MERGE SORT" << endl;

for (int i = 0; i < sizeof(inputs) / sizeof(inputs[0]); i++)

{

steps = 0;

cApprox = 0;

double totalT = 0;

for (int x = 0; x < 50; x++)

{

vector<int> arr;

for (int j = 1; j <= inputs[i]; j++)

{

arr.push\_back(rand() % inputs[i] + 1);

}

auto t1 = chrono::high\_resolution\_clock::now();

mergeSort(arr, 0, (arr.size() - 1));

auto t2 = chrono::high\_resolution\_clock::now();

chrono::duration<double, milli> timer = t2 - t1;

totalT += timer.count();

arr.clear();

}

cApprox = (steps / 50) / (inputs[i] \* log(inputs[i]));

cout << "50 INSTANCES OF " << inputs[i] << " ELEMENTS: Steps = " << steps << " | Runtime = " << totalT << " miliseconds | Average time: " << totalT/50 << " miliseconds. | C Approximation: " << cApprox << endl;

}

system("PAUSE");

return 0;

} //end main

**A screen shot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generatedOutput:**

---------------------------------INSERTION SORT----------------------------------------

SORTED ARRAYS

100 ELEMENTS: Steps = 599 | Runtime = 0.2228 miliseconds. | C Approximation: 0.0599

200 ELEMENTS: Steps = 1199 | Runtime = 0.4576 miliseconds. | C Approximation: 0.029975

300 ELEMENTS: Steps = 1799 | Runtime = 0.6374 miliseconds. | C Approximation: 0.0199889

400 ELEMENTS: Steps = 2399 | Runtime = 0.7231 miliseconds. | C Approximation: 0.0149938

500 ELEMENTS: Steps = 2999 | Runtime = 0.9314 miliseconds. | C Approximation: 0.011996

1000 ELEMENTS: Steps = 5999 | Runtime = 2.1042 miliseconds. | C Approximation: 0.005999

4000 ELEMENTS: Steps = 23999 | Runtime = 8.4642 miliseconds. | C Approximation: 0.00149994

10000 ELEMENTS: Steps = 59999 | Runtime = 23.7712 miliseconds. | C Approximation: 0.00059999

REVERSED ARRAYS

100 ELEMENTS: Steps = 20399 | Runtime = 7.8396 miliseconds. | C Approximation: 2.0399

200 ELEMENTS: Steps = 80799 | Runtime = 32.0542 miliseconds. | C Approximation: 2.01998

300 ELEMENTS: Steps = 181199 | Runtime = 58.8599 miliseconds. | C Approximation: 2.01332

400 ELEMENTS: Steps = 321599 | Runtime = 101.132 miliseconds. | C Approximation: 2.00999

500 ELEMENTS: Steps = 501999 | Runtime = 157.65 miliseconds. | C Approximation: 2.008

1000 ELEMENTS: Steps = 2003999 | Runtime = 671.8 miliseconds. | C Approximation: 2.004

4000 ELEMENTS: Steps = 32015999 | Runtime = 10914.1 miliseconds. | C Approximation: 2.001

10000 ELEMENTS: Steps = 200039999 | Runtime = 67952.8 miliseconds. | C Approximation: 2.0004

RANDOM PERMUTATION ARRAYS

100 ELEMENTS: Steps = 10039 | Runtime = 5.9666 miliseconds. | C Approximation: 1.0039

200 ELEMENTS: Steps = 43219 | Runtime = 12.8792 miliseconds. | C Approximation: 1.08048

300 ELEMENTS: Steps = 92555 | Runtime = 43.1252 miliseconds. | C Approximation: 1.02839

400 ELEMENTS: Steps = 160691 | Runtime = 70.7684 miliseconds. | C Approximation: 1.00432

500 ELEMENTS: Steps = 259711 | Runtime = 106.263 miliseconds. | C Approximation: 1.03884

1000 ELEMENTS: Steps = 1000019 | Runtime = 340.8 miliseconds. | C Approximation: 1.00002

4000 ELEMENTS: Steps = 15846627 | Runtime = 5319.75 miliseconds. | C Approximation: 0.990414

10000 ELEMENTS: Steps = 99309943 | Runtime = 33848.4 miliseconds. | C Approximation: 0.993099

---------------------------------MERGE SORT----------------------------------------

SORTED ARRAYS

100 ELEMENTS: Steps = 5774 | Runtime = 3.2449 miliseconds. | C Approximation: 12.5381

200 ELEMENTS: Steps = 12958 | Runtime = 7.4184 miliseconds. | C Approximation: 12.2284

300 ELEMENTS: Steps = 20662 | Runtime = 18.4022 miliseconds. | C Approximation: 12.075

400 ELEMENTS: Steps = 28726 | Runtime = 15.4998 miliseconds. | C Approximation: 11.9862

500 ELEMENTS: Steps = 36518 | Runtime = 19.7829 miliseconds. | C Approximation: 11.7523

1000 ELEMENTS: Steps = 80046 | Runtime = 45.9873 miliseconds. | C Approximation: 11.5878

4000 ELEMENTS: Steps = 376214 | Runtime = 202.264 miliseconds. | C Approximation: 11.3399

10000 ELEMENTS: Steps = 1044102 | Runtime = 543.784 miliseconds. | C Approximation: 11.3362

REVERSED ARRAYS

100 ELEMENTS: Steps = 5298 | Runtime = 3.2288 miliseconds. | C Approximation: 11.5045

200 ELEMENTS: Steps = 11906 | Runtime = 7.0397 miliseconds. | C Approximation: 11.2356

300 ELEMENTS: Steps = 18970 | Runtime = 11.0564 miliseconds. | C Approximation: 11.0862

400 ELEMENTS: Steps = 26422 | Runtime = 15.5015 miliseconds. | C Approximation: 11.0249

500 ELEMENTS: Steps = 34078 | Runtime = 19.636 miliseconds. | C Approximation: 10.9671

1000 ELEMENTS: Steps = 74666 | Runtime = 41.971 miliseconds. | C Approximation: 10.809

4000 ELEMENTS: Steps = 350694 | Runtime = 193.132 miliseconds. | C Approximation: 10.5707

10000 ELEMENTS: Steps = 961894 | Runtime = 699.274 miliseconds. | C Approximation: 10.4436

RANDOM PERMUTATION ARRAYS

100 ELEMENTS: Steps = 6225 | Runtime = 6.6562 miliseconds. | C Approximation: 13.5174

200 ELEMENTS: Steps = 14202 | Runtime = 13.5168 miliseconds. | C Approximation: 13.4024

300 ELEMENTS: Steps = 22766 | Runtime = 21.6862 miliseconds. | C Approximation: 13.3046

400 ELEMENTS: Steps = 31873 | Runtime = 30.2726 miliseconds. | C Approximation: 13.2993

500 ELEMENTS: Steps = 40912 | Runtime = 37.8189 miliseconds. | C Approximation: 13.1664

1000 ELEMENTS: Steps = 90393 | Runtime = 79.1967 miliseconds. | C Approximation: 13.0857

4000 ELEMENTS: Steps = 429529 | Runtime = 234.725 miliseconds. | C Approximation: 12.9469

10000 ELEMENTS: Steps = 1190897 | Runtime = 616.885 miliseconds. | C Approximation: 12.93

---------------------------------50 INSTANCES INSERTION SORT----------------------------------------

50 INSTANCES INSERTION SORT

50 INSTANCES OF 100 ELEMENTS: Steps = 519010 | Runtime = 163.767 miliseconds | Average time: 3.27535 miliseconds. | C Approximation: 1.038

50 INSTANCES OF 200 ELEMENTS: Steps = 2042090 | Runtime = 725.248 miliseconds | Average time: 14.505 miliseconds. | C Approximation: 1.02103

50 INSTANCES OF 300 ELEMENTS: Steps = 4580526 | Runtime = 1516.16 miliseconds | Average time: 30.3233 miliseconds. | C Approximation: 1.01789

50 INSTANCES OF 400 ELEMENTS: Steps = 8051418 | Runtime = 2697.91 miliseconds | Average time: 53.9581 miliseconds. | C Approximation: 1.00642

50 INSTANCES OF 500 ELEMENTS: Steps = 12578690 | Runtime = 4358.85 miliseconds | Average time: 87.177 miliseconds. | C Approximation: 1.00629

50 INSTANCES OF 1000 ELEMENTS: Steps = 50177098 | Runtime = 17015.4 miliseconds | Average time: 340.309 miliseconds. | C Approximation: 1.00354

50 INSTANCES OF 4000 ELEMENTS: Steps = 802551986 | Runtime = 271134 miliseconds | Average time: 5422.68 miliseconds. | C Approximation: 1.00319

50 INSTANCES OF 10000 ELEMENTS: Steps = 5003738162 | Runtime = 1.6908e+06 miliseconds | Average time: 33815.9 miliseconds. | C Approximation: 1.00075

---------------------------------50 INSTANCES MERGE SORT----------------------------------------

50 INSTANCES MERGE SORT

50 INSTANCES OF 100 ELEMENTS: Steps = 312888 | Runtime = 189.906 miliseconds | Average time: 3.79812 miliseconds. | C Approximation: 13.5869

50 INSTANCES OF 200 ELEMENTS: Steps = 710751 | Runtime = 427.427 miliseconds | Average time: 8.54855 miliseconds. | C Approximation: 13.4146

50 INSTANCES OF 300 ELEMENTS: Steps = 1140119 | Runtime = 655.026 miliseconds | Average time: 13.1005 miliseconds. | C Approximation: 13.3257

50 INSTANCES OF 400 ELEMENTS: Steps = 1591635 | Runtime = 905.39 miliseconds | Average time: 18.1078 miliseconds. | C Approximation: 13.2822

50 INSTANCES OF 500 ELEMENTS: Steps = 2047417 | Runtime = 1174.01 miliseconds | Average time: 23.4803 miliseconds. | C Approximation: 13.178

50 INSTANCES OF 1000 ELEMENTS: Steps = 4518257 | Runtime = 2530.6 miliseconds | Average time: 50.6121 miliseconds. | C Approximation: 13.0817

50 INSTANCES OF 4000 ELEMENTS: Steps = 21477434 | Runtime = 11509.5 miliseconds | Average time: 230.19 miliseconds. | C Approximation: 12.9475

50 INSTANCES OF 10000 ELEMENTS: Steps = 59537974 | Runtime = 31380.7 miliseconds | Average time: 627.614 miliseconds. | C Approximation: 12.9285