# Γραφικές Παραστάσεις

# Αρχείο Χρόνου (TIMES.txt):

EXECUTION TIMES(seconds)

BubbleSortSelectSortInsertSort QuickSort

============================== =========

22.772 7.014 4.372 0.011

EXECUTION TIMES(seconds)

BubbleSortSelectSortInsertSort QuickSort

============================== =========

23.321 7.02 3.26 0.008

EXECUTION TIMES(seconds)

BubbleSortSelectSortInsertSort QuickSort

============================== =========

8.723 7.088 0 0.003

EXECUTION TIMES(seconds)

BubbleSortSelectSortInsertSort QuickSort

============================== =========

16.487 8.288 8.832 0.003

EXECUTION TIMES(seconds)

BubbleSortSelectSortInsertSort QuickSort

============================== =========

91.978 28.356 17.667 0.025

EXECUTION TIMES(seconds)

BubbleSortSelectSortInsertSort QuickSort

============================== =========

94.417 28.351 13.138 0.02

EXECUTION TIMES(seconds)

BubbleSortSelectSortInsertSort QuickSort

============================== =========

35.072 28.31 0.001 0.006

EXECUTION TIMES(seconds)

BubbleSortSelectSortInsertSort QuickSort

============================== =========

66.052 33.148 35.203 0.004

# Τελικά Συμπεράσματα

Η quick sort είναι σταθερά γρήγορη σε όλες τις περιπτώσεις και μεγεθή πινάκων.

Η bubble sort είναι η πιο αργή από όλους τους άλλος αλγορίθμους, αλλά όταν ειναι sorted ascending ο πίνακας ολοκληρώνει την εκτέλεσή της σε συγκριτικά γρηγορότερο χρόνο.

Η select sort παίρνει σχεδόν τον ίδιο χρόνο να ταξινομήσει τον πίνακα ανεξαρτήτως του τρόπου δημιουργίας του αλλά συγκριτικά με την insert και την quick είναι πιο αργή.

Η insert sort, είναι η δεύτερη πιο γρήγορη μέθοδος μετά απο την quick, με μία εξαίρεση, όταν ο πίνακας είναι sorted ascending η insert είναι πιο γρήγορη απο όλες. Επίσης, όταν ειναι sorted descending, ειναι η δεύτερη πιο αργή, μετά από την bubble.

# Κώδικας Προγράμματος

main.cpp

#include "ArrayGen.h"

#include <iostream>

#include <ctime>

#include <fstream>

#include <iomanip>

using namespace std;

void printData();

int arraySize();

int arrayChoice();

void createFile(float sec1,float sec2, float sec3, float sec4);

int main(int argc, char\*\* argv) {

int size,choice;

float sec1,sec2,sec3,sec4;

clock\_t ct0,ct1;

ArrayGen original,copy;

printData();

size=arraySize();

original.setArraySize(size);

original.createArray();

choice=arrayChoice();

switch(choice)

{

case(1):

original.fill100();

break;

case(2):

original.fill50();

break;

case(3):

original.fillASC();

break;

case(4):

original.fillDESC();

break;

default:

cout << "Something went wrong, try again";

}

cout << "Sorting ==> ";

copy.setArraySize(size);

copy.createArray();

original.cpArray(copy);

//bubblesort

cout << "Bubble Sort... ";

ct0=clock();

copy.bubbleSort();

ct1=clock();

sec1=(float)(ct1-ct0)/CLOCKS\_PER\_SEC;

//selectsort

cout << "Select Sort... ";

original.cpArray(copy);

ct0=clock();

copy.selectSort();

ct1=clock();

sec2=(float)(ct1-ct0)/CLOCKS\_PER\_SEC;

//insertsort

cout << "Insert Sort... ";

original.cpArray(copy);

ct0=clock();

copy.insertSort();

ct1=clock();

sec3=(float)(ct1-ct0)/CLOCKS\_PER\_SEC;

//quicksort

cout << "Quick Sort... ";

original.cpArray(copy);

ct0=clock();

copy.quickSort(0,size);

ct1=clock();

sec4=(float)(ct1-ct0)/CLOCKS\_PER\_SEC;

original.freeArrayMem();

copy.freeArrayMem();

createFile(sec1,sec2,sec3,sec4);

return 0;

}

void printData()

{

cout << "Student" << endl;

cout << "=======" << endl;

cout << "Name: Ioannis Allousi" << endl;

cout << "AEM: 4591" << endl << endl;

cout << "Computer Characteristics" << endl;

cout << "========================" << endl;

cout << "CPU: Intel" << endl;

cout << "Intel i5 9600k 3.7Ghz" << endl;

cout << "RAM: 12GB" << endl;

cout << "Virtualized: Yes (KVM)" <<endl << endl;

}

int arraySize()

{

int size;

cout << "Array Size? ";

cin >> size;

cout << endl << endl;

return(size);

}

int arrayChoice()

{

int choice;

cout << "Array Types" << endl;

cout << " 1. Random 100%" << endl;

cout << " 2. Sorted 50%, Random 50%" << endl;

cout << " 3. Sorted ASC" << endl;

cout << " 4. Sorted DESC" << endl;

cout << "Choose array type: ";

cin >> choice;

return(choice);

}

void createFile(float sec1,float sec2,float sec3,float sec4)

{

ofstream results;

results.open("TIMES.txt",ios\_base::app);

results << setw(20)<<"EXECUTION TIMES(seconds)" << endl;

results << "BubbleSort"<<setw(10)<<"SelectSort"<<setw(10)<<"InsertSort"<<setw(10)<<"QuickSort"<<endl;

results << "=========="<<setw(10)<<"=========="<<setw(10)<<"=========="<<setw(10)<<"========="<<endl;

results << setw(10) << setprecision(5) <<sec1 << setw(10) << setprecision(5) << sec2 << setw(10) << setprecision(5) << sec3 << setw(10) << setprecision(5) << sec4 <<endl;

results.close();

}

ArrayGen.h

#ifndef ARRAYGEN\_H

#define ARRAYGEN\_H

class ArrayGen

{

private:

int \*p;

int n;

public:

void setArraySize(int n1);

void createArray();

void fill100();

void fill50();

void fillASC();

void fillDESC();

void freeArrayMem();

void cpArray(ArrayGen ant);

void bubbleSort();

void selectSort();

void insertSort();

void quickSort(int left, int right);

};

#endif

ArrayGen.cpp

#include "ArrayGen.h"

#include "RandMT.h"

#include <ctime>

#include <cstdlib>

#include <iostream>

using namespace std;

void ArrayGen::setArraySize(int n1)

{

n=n1;

}

void ArrayGen::createArray()

{

p = new int[n];

}

void ArrayGen::fill100()

{

int i;

RandMT random;

RandMT (time(NULL));

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

{

p[i]=random();

}

}

void ArrayGen::fillASC()

{

int i;

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

{

p[i]=i;

}

}

void ArrayGen::fillDESC()

{

int i;

for(i=n;i>0;i--)

{

p[n-i]=i;

}

}

void ArrayGen::fill50()

{

int i;

RandMT random;

int m=n/2;

RandMT(time(NULL));

for(i=0;i<m;i++)

{

p[i]=random();

}

for(i=m;i<n;i++)

{

p[i]=i;

}

}

void ArrayGen::freeArrayMem()

{

delete []p;

}

void ArrayGen::cpArray(ArrayGen ant)

{

int i;

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

ant.p[i] = p[i];

}

void ArrayGen::bubbleSort()

{

int i,j,temp;

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

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

if(p[j-1] > p[j])

{

temp = p[j-1];

p[j-1] = p[j];

p[j] = temp;

}

}

void ArrayGen::selectSort()

{

int i,j,k,min;

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

{

k=i;

min=p[i];

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

{

if(p[j]<min)

{

k=j;

min=p[j];

}

}

p[k]=p[i];

p[i]=min;

}

}

void ArrayGen::insertSort()

{

int i,j,x;

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

{

x = p[i];

j = i-1;

while( j>=0 && p[j]>x)

{

p[j+1] = p[j];

j = j-1;

}

p[j+1] = x;

}

}

void ArrayGen::quickSort(int left, int right)

{

int i,j,x,mid;

if(left < right)

{

i = left;

j = right;

mid = (left+right)/2;

x = p[mid];

while(i < j)

{

while(p[i] < x)

i++;

while(p[j] > x)

j--;

if(i < j)

{

if(p[i] == p[j])

{

if(i < mid)

i++;

if(j > mid)

j--;

}

else

{

swap(p[i],p[j]);

}

}

}

quickSort(left,j-1);

quickSort(j+1,right);

}

}

# Παραδείγματα Εκτέλεσης:

