Δομές Δεδομένων

Project: “DATA STRUCTURES 2021”

*Φοιτητές:*

* Λαμπροπούλου Σοφία

ΑΜ: 1072606, email: [up1072606@upnet.gr](mailto:up1072606@upnet.gr)

* Παρασκευόπουλος Κωνσταντίνος

ΑΜ: 1072608, email: [up1072608@upnet.gr](mailto:up1072608@upnet.gr)

* Χαλούλου Δανάη

ΑΜ: 1072596, email: [up1072596@upnet.gr](mailto:up1072596@upnet.gr)

**PART I: “Sorting and Searching Algorithms”**

**Σημείωση:** Σχεδόν σε όλα τα ερωτήματα χρησιμοποιήσαμε συναρτήσεις από την βιβλιοθήκη InputFile.h. Η βιβλιοθήκη είναι η παρακάτω

*#ifndef \_INPUTFILE\_H*

*#define \_INPUTFILE\_H*

*#include<iostream>*

*#include<fstream>*

*#define AGN\_SIZE 3239*

*#define AINV\_SIZE 3201*

*#define ALE\_SIZE 3201*

*#define TEST\_SIZE 7*

*using namespace std;*

*/\**

*Δημιουργία struct μετοχής με όλα τα παιδιά που περιέχονται σε κάθε γραμμή των αρχείων*

*\*/*

*int getDatefromKeyboard(){*

*int y = 0;*

*int m = 0;*

*int d = 0;*

*int fulldate;*

*start: cout<<"Give Wanted Date (With Spaces or Not between the numbers)\nFormat(yyyy mm dd) ";*

*cout<<"Date: ";*

*cin>>y;*

*if(y>3000){*

*fulldate = y;*

*}*

*else{*

*cin>>m>>d;*

*fulldate = 10000\*y+100\*m+d;*

*}*

*if (fulldate>30000000){*

*goto start;*

*}*

*return fulldate;*

*}*

*struct metoxh{*

*long int Date;*

*float Open;*

*float High;*

*float Low;*

*long int Volume;*

*float OpenInt;*

*float Close;*

*//Date,Open,High,Low,Close,Volume,OpenInt*

*};*

*void StructAssignTo(struct metoxh &s1, struct metoxh &s2){ //StructAssignTo(s1[i], R[n])---->R[n]=s1[i] ,,,,,,,,,,,,,,,,,,,,,,s2=s1*

*s2.Date=s1.Date;*

*s2.Open=s1.Open;*

*s2.High=s1.High;*

*s2.Low=s1.Low;*

*s2.OpenInt=s1.OpenInt;*

*s2.Volume=s1.Volume;*

*s2.Close=s1.Close;*

*}*

*/\**

*ReadValuesfromFile(ifstream &fin, struct metoxh testarray)*

*;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*;*

*1o όρισμα αρχείο τύπου ifstream π.χ. ifstream finput("agn.us.txt"); όπου finput μπαίνει ως όρισμα*

*2o όρισμα τύπου πίνακα από structs τύπου "metoxh"*

*\*/*

*void ReadValuesfromFile(ifstream &fin, struct metoxh StocksArray[]){*

*int i=0;*

*if (fin.good()){*

*while(fin >> StocksArray[i].Date >> StocksArray[i].Open >> StocksArray[i].High >> StocksArray[i].Low >> StocksArray[i].Close >> StocksArray[i].Volume >> StocksArray[i].OpenInt){*

*//cout <<"Date: "<< StocksArray[i].Date <<" Open: "<< StocksArray[i].Open <<" High: "<< StocksArray[i].High <<" Low: "<< StocksArray[i].Low <<" Close: "<< StocksArray[i].Close <<" Volume: "<< StocksArray[i].Volume <<" OpenInt: "<< StocksArray[i].OpenInt<<endl;*

*//εμφάνιση όλων των περιεχομένων του κάθε πίνακα στο terminal*

*i++;*

*}*

*//cout<<"\n\nLast input: "<<i;*

*/\*εάν θέλετε να δείτε το μέγεθος των πινάκων, αφαιρέστε από τα σχόλια την γραμμή κώδικα από πάνω\*/*

*fin.close();*

*}*

*else{*

*cout<<"File Corrupted or doesn't Exist!"<<endl;*

*}*

*}*

*void PrintValuesfromFile(ifstream &fin, struct metoxh StocksArray[]){*

*int i=0;*

*if (fin.good()){*

*while(fin >> StocksArray[i].Date >> StocksArray[i].Open >> StocksArray[i].High >> StocksArray[i].Low >> StocksArray[i].Close >> StocksArray[i].Volume >> StocksArray[i].OpenInt){*

*cout <<"Date: "<< StocksArray[i].Date <<" Open: "<< StocksArray[i].Open <<" High: "<< StocksArray[i].High <<" Low: "<< StocksArray[i].Low <<" Close: "<< StocksArray[i].Close <<" Volume: "<< StocksArray[i].Volume <<" OpenInt: "<< StocksArray[i].OpenInt<<endl;*

*//εμφάνιση όλων των περιεχομένων του κάθε πίνακα στο terminal*

*i++;*

*}*

*//cout<<"\n\nLast input: "<<i;*

*/\*εάν θέλετε να δείτε το μέγεθος των πινάκων, αφαιρέστε από τα σχόλια την γραμμή κώδικα από πάνω\*/*

*fin.close();*

*}*

*else{*

*cout<<"File Corrupted or doesn't Exist!"<<endl;*

*}*

*}*

*int CountValuesfromFile(ifstream &fin, struct metoxh StocksArray[]){*

*int i=0;*

*if (fin.good()){*

*while(fin >> StocksArray[i].Date >> StocksArray[i].Open >> StocksArray[i].High >> StocksArray[i].Low >> StocksArray[i].Close >> StocksArray[i].Volume >> StocksArray[i].OpenInt){*

*//cout <<"Date: "<< StocksArray[i].Date <<" Open: "<< StocksArray[i].Open <<" High: "<< StocksArray[i].High <<" Low: "<< StocksArray[i].Low <<" Close: "<< StocksArray[i].Close <<" Volume: "<< StocksArray[i].Volume <<" OpenInt: "<< StocksArray[i].OpenInt<<endl;*

*//εμφάνιση όλων των περιεχομένων του κάθε πίνακα στο terminal*

*i++;*

*}*

*cout<<"\n\nLast input: "<<i<<endl;*

*/\*εάν θέλετε να δείτε το μέγεθος των πινάκων, αφαιρέστε από τα σχόλια την γραμμή κώδικα από πάνω\*/*

*fin.close();*

*}*

*else{*

*cout<<"File Corrupted or doesn't Exist!"<<endl;*

*}*

*return i;*

*}*

*void swapStructs(struct metoxh &s1, struct metoxh &s2){*

*struct metoxh temp;*

*temp.Date=s1.Date;*

*temp.Open=s1.Open;*

*temp.High=s1.High;*

*temp.Low=s1.Low;*

*temp.OpenInt=s1.OpenInt;*

*temp.Volume=s1.Volume;*

*temp.Close=s1.Close;*

*s1.Date=s2.Date;*

*s1.Open=s2.Open;*

*s1.High=s2.High;*

*s1.Low=s2.Low;*

*s1.OpenInt=s2.OpenInt;*

*s1.Volume=s2.Volume;*

*s1.Close=s2.Close;*

*s2.Date=temp.Date;*

*s2.Open=temp.Open;*

*s2.High=temp.High;*

*s2.Low=temp.Low;*

*s2.OpenInt=temp.OpenInt;*

*s2.Volume=temp.Volume;*

*s2.Close=temp.Close;*

*}*

*void mergedifferent(struct metoxh arr[], int left, int mid, int right)*

*{*

*int i, j, k;*

*int n1 = mid - left + 1;*

*int n2 = right - mid;*

*/\* δημιουργώ τα temp arrays \*/*

*struct metoxh L[n1], R[n2];*

*/\* αντιγράφω τα δεδομένα στα temp arrays L[] και R[], χωρίζω τον πίνακα arr στην μέση \*/*

*//while(g=1){*

*for (i = 0; i < n1; i++)*

*StructAssignTo(arr[left+i], L[i]); //L[i] = arr[l + i];*

*for (j = 0; j < n2; j++)*

*StructAssignTo(arr[mid + 1 + j],R[j]);//R[j] = arr[m + 1 + j];*

*//}*

*/\* συγχωνεύω τα temp arrays στον arr[l..r]\* και φτιάχνω τους νέους πίνακες L και R \*/*

*i = 0; // αρχικό index του πρώτου subarray*

*j = 0; // αρχικό index του δεύτερου subarray*

*k = left; // αρχικό index του συγχωνευμένου subarray*

*while (i < n1 && j < n2) {*

*if (L[i].Date <= R[j].Date) {*

*StructAssignTo( L[i], arr[k] ); //arr[k] = L[i]*

*i++;*

*}*

*else {*

*StructAssignTo( R[j],arr[k] ); //arr[k] = R[j];*

*j++;*

*}*

*k++;*

*}*

*/\* αντιγράφω τα υπόλοιπα στοιχεία του L[] (εάν υπάρχουν)\*/*

*while (i < n1) {*

*StructAssignTo( L[i],arr[k] );//arr[k] =* *L[i];*

*i++;*

*k++;*

*}*

*/\* αντιγράφω τα υπόλοιπα στοιχεία του R[] (εάν υπάρχουν)\*/*

*while (j < n2) {*

*StructAssignTo( R[j],arr[k] );//arr[k] = R[j];*

*j++;*

*k++;*

*}*

*}*

*void mergeSortdifferent(struct metoxh met[], int left, int right)*

*{*

*if (left >= right) {*

*return;*

*}*

*// Same as (l+r)/2, but avoids overflow for*

*// large l and h*

*int mid = left + (right - left) / 2;*

*// ταξινόμηση του πρώτου και δεύτερου μισού*

*mergeSortdifferent(met, left, mid);*

*mergeSortdifferent(met, mid + 1, right);*

*mergedifferent(met, left, mid, right);*

*//cout<<"Done"<<endl; //Debug Help*

*}*

*void PrintArray(struct metoxh StocksArray[], int size){*

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

*cout <<"Date: "<< StocksArray[i].Date <<" Open: "<< StocksArray[i].Open <<" High: "<< StocksArray[i].High <<" Low: "<< StocksArray[i].Low <<" Close: "<< StocksArray[i].Close <<" Volume: "<< StocksArray[i].Volume <<" OpenInt: "<< StocksArray[i].OpenInt<<endl;*

*}*

*}*

*#endif*

**ΕΠΕΞΗΓΗΣΗ ΒΑΣΙΚΩΝ ΣΗΜΕΙΩΝ**

#define AGN\_SIZE 3239

#define AINV\_SIZE 3201

#define ALE\_SIZE 3201

**:Έπειτα από πειράματα με την συνάρτηση** CountValuesfromFile() **βρέθηκαν τα μεγέθη των αρχείων και ορίζονται ως σταθερές προς χρήση για όλα τα προγράμματα.**

**Σημείωση:**  Θα μπορούσε η κάθε σταθερά να προσδιορίζεται σε κάθε πρόγραμμα με την χρήση της συνάρτησης CountValuesfromFile() ωστόσο χάριν ευκολίας βρήκαμε πειραματικά αυτούς τους αριθμούς και τους θέσαμε ως σταθερές.

**int** getDatefromKeyboard **:Η συνάρτηση αυτή ζητά από τον χρήστη να εισάγει μια ημερομηνία με την μορφή yyyy mm dd με κενά ή χωρίς ανάμεσα από τις επιμέρους μεταβλητές και επιστρέφει την ημερομηνία ως έναν ακέραιο της μορφής yyyymmdd δηλαδή year\*10000+month\*100+day**

**struct** metoxh **:Δομή με μεταβλητές αντίστοιχες με τα δεδομένα των δοσμένων αρχείων**

**void** StructAssignTo**(struct** metoxh **&**s1**,** **struct** metoxh **&**s2**) :Συνάρτηση η οποία κάνει την διαδικασία** s2 =s1; **για structs τύπου metoxh**

**void** ReadValuesfromFile**(**ifstream **&**fin**,** **struct** metoxh StocksArray**[])** **:Συνάρτηση η οποία διαβάζει αρχεία από ένα infile τύπου ifstream και αποθηκεύει τα δεδομένα του σε ένα πίνακα για structs με σταθερό μέγεθος όσο το μέγεθος που βρήκαμε πειραματικά για το κάθε αρχείο (δεν εκτυπώνει τίποτα)**

**void** PrintValuesfromFile**(**ifstream **&**fin**,** **struct** metoxh StocksArray**[]) :Συνάρτηση η οποία διαβάζει αρχεία από ένα infile τύπου ifstream και αποθηκεύει τα δεδομένα του σε ένα πίνακα για structs με σταθερό μέγεθος όσο το μέγεθος που βρήκαμε πειραματικά για το κάθε αρχείο (εκτυπώνει τα περιεχόμενα του πίνακα με τις αντίστοιχες μεταβλητές του)**

**int** CountValuesfromFile**(**ifstream **&**fin**,** **struct** metoxh StocksArray**[]) :Συνάρτηση η οποία διαβάζει αρχεία από ένα infile τύπου ifstream και αποθηκεύει τα δεδομένα του σε ένα πίνακα για structs με σταθερό μέγεθος όσο το μέγεθος που βρήκαμε πειραματικά για το κάθε αρχείο (εκτυπώνει το μέγεθος του αρχείου σε γραμμές και χρησιμοποιείται για πειραματική διαπίστωση του μεγέθους των πινάκων).**

**ΕΡΩΤΗΜΑ 1Ο**

***Ταξινόμηση κατά αύξουσα σειρά των ημερομηνιών (πεδίο Date) με βάση τις τιμές ανοίγματος (πεδίο Open) των μετοχών με χρήση του αλγορίθμου MERGE SORT***

* Για το αρχείο *agn.us.txt* ο κώδικας είναι:

*#include"InputFile.h"*

*#include<cstdlib>*

*#include<string>*

*#include <iostream>*

*using namespace std;*

*void merge(struct metoxh arr[], int left, int mid, int right)*

*{*

*int i, j, k;*

*int n1 = mid - left + 1;*

*int n2 = right - mid;*

*/\* δημιουργώ τα temp arrays \*/*

*struct metoxh L[n1], R[n2];*

*/\* Αντιγράφω τα δεδομένα στα temp arrays L[] και R[] , χωρίζω τον πίνακα arr στην μέση \*/*

*//while(g=1){*

*for (i = 0; i < n1; i++)*

*StructAssignTo(arr[left+i], L[i]); //L[i] = arr[l + i];*

*for (j = 0; j < n2; j++)*

*StructAssignTo(arr[mid + 1 + j],R[j]); //R[j] = arr[m + 1 + j];*

*//}*

*/\** *Δημιουργώ τους νέους πίνακες L[] και R[] \*/*

*i = 0; /\** *Αρχικός δείκτης(index) του πρώτου subarray\*/*

*j = 0; /\* Αρχικός δείκτης(index) του δεύτερου subarray\*/*

*k = left; /\** *Αρχικός δείκτης(index) του merged subarray\*/*

*while (i < n1 && j < n2) {*

*if (L[i].Open <= R[j].Open) { /\* Συγκρίνω τα δύο subarrays του πεδίου Open\*/*

*StructAssignTo( L[i], arr[k] ); //arr[k] = L[i]*

*i++;*

*}*

*else {*

*StructAssignTo( R[j],arr[k] ); //arr[k] = R[j];*

*j++;*

*}*

*k++;*

*}*

*/\** *Αντιγράφω τα υπόλοιπα στοιχεία (εάν υπάρχουν) του πίκανα L[] \*/*

*while (i < n1) {*

*StructAssignTo( L[i],arr[k] ); //arr[k] = L[i];*

*i++;*

*k++;*

*}*

*/\** *Αντιγράφω τα υπόλοιπα στοιχεία (εάν υπάρχουν) του πίκανα R[] \*/*

*while (j < n2) {*

*StructAssignTo( R[j],arr[k] ); //arr[k] = R[j];*

*j++;*

*k++;*

*}*

*}*

*/\* το left είναι για το αριστερό index και το right είναι για το δεξί index του*

*sub-array του πίνακα που πρέπει να ταξινομηθεί \*/*

*void mergeSort(struct metoxh met[], int left, int right)*

*{*

*if (left >= right) {*

*return;*

*}*

*// Same as (l+r)/2, but avoids overflow for*

*// large l and h*

*int mid = left + (right - left) / 2;*

*/\* Ταξινόμηση του πρώτου και του δεύτερου μισού του πίνακα \*/*

*mergeSort(met, left, mid);*

*mergeSort(met, mid + 1, right);*

*merge(met, left, mid, right);*

*//cout<<"Done"<<endl;*

*}*

*void printArray(struct metoxh arr[], int size)*

*{*

*int i;*

*int j=1;*

*for (i = 0; i < size; i++){*

*cout<<arr[i].Date<<"||"<<arr[i].Open<<endl;*

*}*

*}*

*int main()*

*{*

*struct metoxh test[AGN\_SIZE];*

*ifstream file1("**agn.us.txt");*

*ReadValuesfromFile(file1, test);*

*mergeSort(test, 0, AGN\_SIZE - 1);*

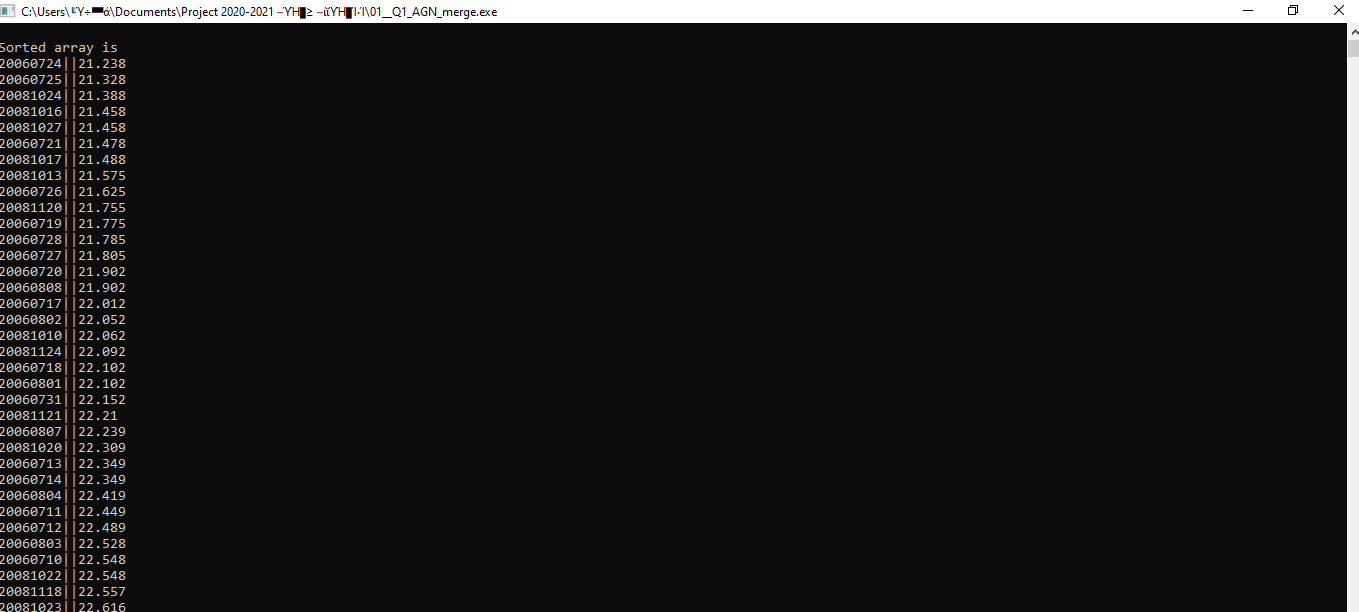
*cout<<"\nSorted array is \n";*

*printArray(test, AGN\_SIZE);*

*return 0;*

*}*

Αν τρέξουμε τον κώδικα θα έχουμε τα εξής



* Για το αρχείο *ainv.us.txt* ο κώδικας είναι:

*#include"E:\KONSTANTINOS\лахглата\4О еналгмо\долес дедолемым\PROJECTS\_2020\_2021\MY PROGRAMMS\InputFile.h"*

*#include<cstdlib>*

*#include<string>*

*#include <iostream>*

*/\* Συγχένευση των δύο subarrays του αρχικού πίνακα arr[].*

*Το πρώτο subarray είναι ο arr[l..m]*

*Το δεύτερο subarray είναι ο arr[m+1..r] \*/*

*using namespace std;*

*void merge(struct metoxh arr[], int left, int mid, int right)*

*{*

*int i, j, k;*

*int n1 = mid - left + 1;*

*int n2 = right - mid;*

*/\* Δημιουργώ τα temp arrays \*/*

*struct metoxh L[n1], R[n2];*

*/\* Αντιγράφω τα δεδομένα στα temp arrays L[] και R[] και χωρίζω τον πίνακα arr στην μέση \*/*

*//while(g=1){*

*for (i = 0; i < n1; i++)*

*StructAssignTo(arr[left+i], L[i]); //L[i] = arr[l + i];*

*for (j = 0; j < n2; j++)*

*StructAssignTo(arr[mid + 1 + j],R[j]); //R[j] = arr[m + 1 + j];*

*//}*

*/\* Δημιουργώ τους νέους πίνακες L[] και R[] \*/*

*i = 0; /\** *Αρχικός δείκτης(index) του πρώτου subarray \*/*

*j = 0; /\* Αρχικός δείκτης(index) του δεύτερου subarray \*/*

*k = left; /\* Αρχικός δείκτης(index) του merged subarray \*/*

*while (i < n1 && j < n2) {*

*if (L[i].Open <= R[j].Open) { /\* Συγκρίνω τα δύο subarrays του πεδίου Open \*/*

*StructAssignTo( L[i], arr[k] ); //arr[k] = L[i]*

*i++;*

*}*

*else {*

*StructAssignTo( R[j],arr[k] ); //arr[k] = R[j];*

*j++;*

*}*

*k++;*

*}*

*/\** *Αντιγράφω τα υπόλοιπα στοιχεία (εάν υπάρχουν) του πίκανα L[] \*/*

*while (i < n1) {*

*StructAssignTo( L[i],arr[k] ); //arr[k] = L[i];*

*i++;*

*k++;*

*}*

*/\* Αντιγράφω τα υπόλοιπα στοιχεία (εάν υπάρχουν) του πίκανα R[] \*/*

*while (j < n2) {*

*StructAssignTo( R[j],arr[k] ); //arr[k] = R[j];*

*j++;*

*k++;*

*}*

*}*

*/\** *το left είναι για το αριστερό index και το right είναι για το δεξί index του*

*sub-array του πίνακα που πρέπει να ταξινομηθεί \*/*

*void mergeSort(struct metoxh met[], int left, int right)*

*{*

*if (left >= right) {*

*return;*

*}*

*// Same as (l+r)/2, but avoids overflow for*

*// large l and h*

*int mid = left + (right - left) / 2;*

*/\** *Ταξινόμηση του πρώτου και του δεύτερου μισού του πίνακα\*/*

*mergeSort(met, left, mid);*

*mergeSort(met, mid + 1, right);*

*merge(met, left, mid, right);*

*//cout<<"Done"<<endl; //Debug Help*

*}*

*void printArray(struct metoxh arr[], int size)*

*{*

*int i;*

*int j=1;*

*for (i = 0; i < size; i++){*

*cout<<arr[i].Date<<"||"<<arr[i].Open<<endl;*

*}*

*}*

*int main()*

*{*

*struct metoxh test[AINV\_SIZE];*

*ifstream file1("**ainv.us.txt");*

*ReadValuesfromFile(file1, test);*

*mergeSort(test, 0, AINV\_SIZE - 1);*

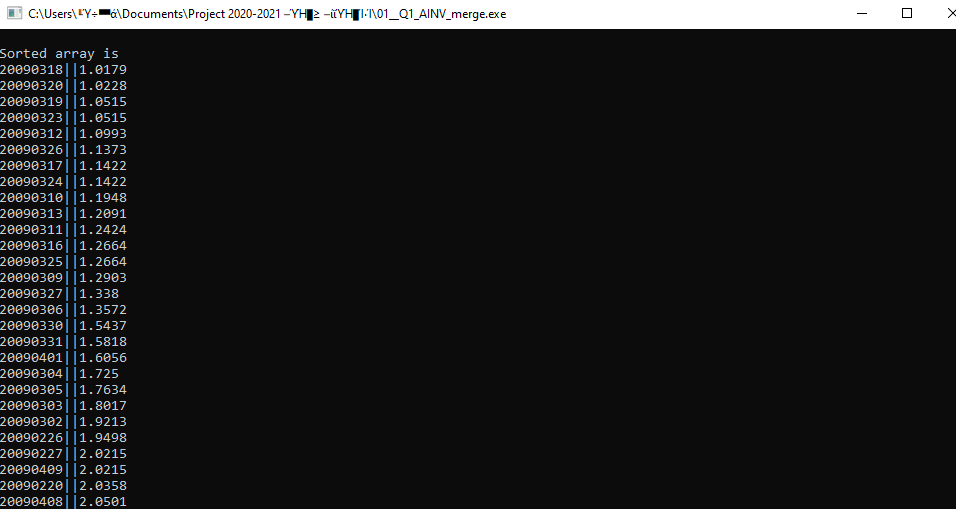
*cout<<"\nSorted array is \n";*

*printArray( test, AINV\_SIZE);*

*return 0;*

*}*

Αν τρέξουμε τον κώδικα θα έχουμε τα εξής



* Για το αρχείο *ale.us.txt* ο κώδικας είναι:

*#include"E:\KONSTANTINOS\лахглата\4О еналгмо\долес дедолемым\PROJECTS\_2020\_2021\MY PROGRAMMS\InputFile.h"*

*#include<cstdlib>*

*#include<string>*

*#include <iostream>*

*/\* Συγχένευση των δύο subarrays του αρχικού πίνακα arr[].*

*Το πρώτο subarray είναι ο arr[l..m]*

*Το δεύτερο subarray είναι ο arr[m+1..r]\*/*

*using namespace std;*

*void merge(struct metoxh arr[], int left, int mid, int right)*

*{*

*int i, j, k;*

*int n1 = mid - left + 1;*

*int n2 = right - mid;*

*/\* Δημιουργώ τα temp arrays \*/*

*struct metoxh L[n1], R[n2];*

*/\* Αντιγράφω τα δεδομένα στα temp arrays L[] και R[] , χωρίζω τον πίνακα arr στην μέση \*/*

*//while(g=1){*

*for (i = 0; i < n1; i++)*

*StructAssignTo(arr[left+i], L[i]); //L[i] = arr[l + i];*

*for (j = 0; j < n2; j++)*

*StructAssignTo(arr[mid + 1 + j],R[j]); //R[j] = arr[m + 1 + j];*

*//}*

*/\* Δημιουργώ τους νέους πίνακες L[] και R[] \*/*

*i = 0; /\* Αρχικός δείκτης(index) του πρώτου subarray \*/*

*j = 0; /\* Αρχικός δείκτης(index) του δεύτερου subarray\*/*

*k = left; /\* Αρχικός δείκτης(index) του merged subarray \*/*

*while (i < n1 && j < n2) {*

*if (L[i].Open <= R[j].Open) {* */\* Συγκρίνω τα δύο subarrays του πεδίου Open\*/*

*StructAssignTo( L[i], arr[k] ); //arr[k] = L[i]*

*i++;*

*}*

*else {*

*StructAssignTo( R[j],arr[k] ); //arr[k] = R[j];*

*j++;*

*}*

*k++;*

*}*

*/\* Αντιγράφω τα υπόλοιπα στοιχεία (εάν υπάρχουν) του πίκανα L[] \*/*

*while (i < n1) {*

*StructAssignTo( L[i],arr[k] ); //arr[k] = L[i];*

*i++;*

*k++;*

*}*

*/\* Αντιγράφω τα υπόλοιπα στοιχεία (εάν υπάρχουν) του πίκανα L[] \*/*

*while (j < n2) {*

*StructAssignTo( R[j],arr[k] ); //arr[k] = R[j];*

*j++;*

*k++;*

*}*

*}*

*/\* το left είναι για το αριστερό index και το right είναι για το δεξί index του*

*sub-array του πίνακα που πρέπει να ταξινομηθεί \*/*

*void mergeSort(struct metoxh met[], int left, int right)*

*{*

*if (left >= right) {*

*return;*

*}*

*// Same as (l+r)/2, but avoids overflow for*

*// large l and h*

*int mid = left + (right - left) / 2;*

*/\* Ταξινόμηση του πρώτου και του δεύτερου μισού του πίνακα\*/*

*mergeSort(met, left, mid);*

*mergeSort(met, mid + 1, right);*

*merge(met, left, mid, right);*

*//cout<<"Done"<<endl; //Debug Help*

*}*

*void printArray(struct metoxh arr[], int size)*

*{*

*int i;*

*int j=1;*

*for (i = 0; i < size; i++){*

*cout<<arr[i].Date<<"||"<<arr[i].Open<<endl;*

*}*

*}*

*int main()*

*{*

*struct metoxh test[ALE\_SIZE];*

*ifstream file1("ale.us.txt");*

*ReadValuesfromFile(file1, test);*

*mergeSort(test, 0, ALE\_SIZE - 1);*

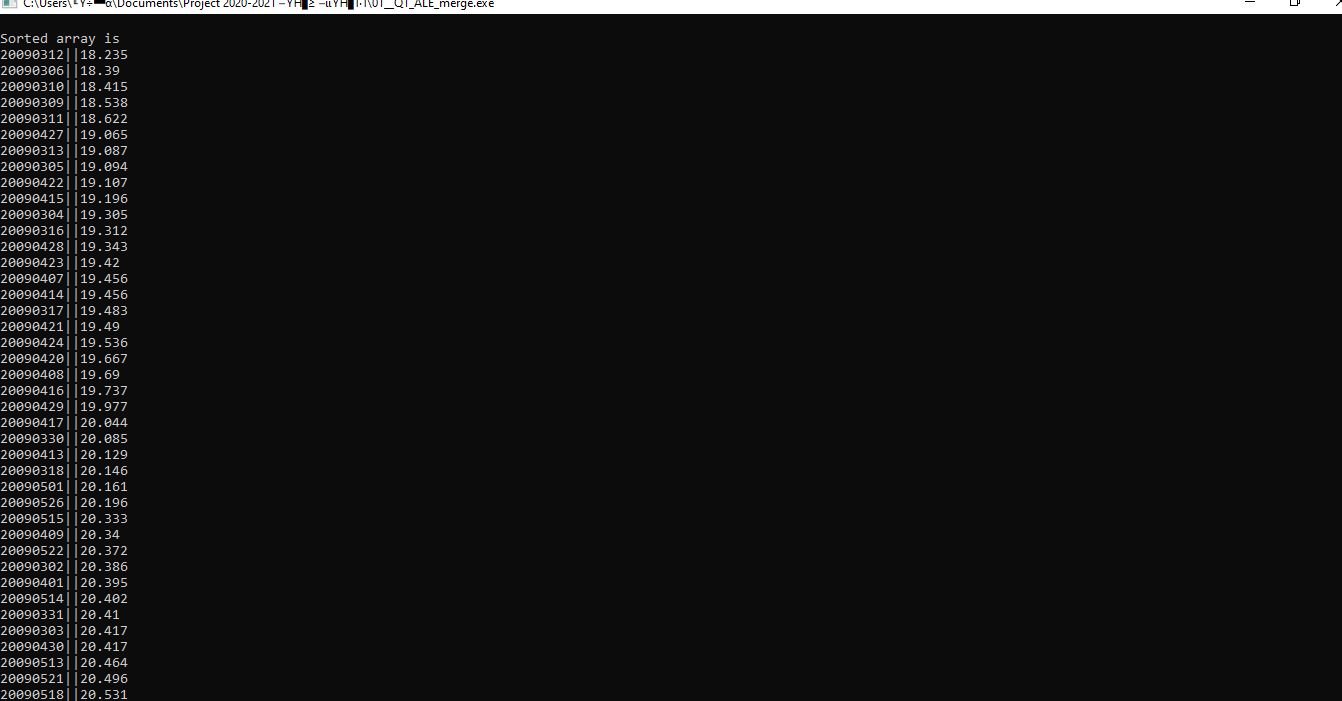
*cout<<"\nSorted array is \n";*

*printArray( test, ALE\_SIZE);*

*return 0;*

*}*

Αν τρέξουμε τον κώδικα θα έχουμε τα εξής



**Στην ουσία με τον αλγόριθμο merge sort χωρίζουμε την αταξινόμητη λίστα του πεδίου Open σε n υπολίστες, καθεμία από τις οποίες περιέχει ένα στοιχείο (μια λίστα με ένα στοιχείο θεωρείται ταξινομημένη).**

**Στην συνέχεια συγχωνεύουμε επανειλημμένα τις υπολίστες για να δημιουργήσουμε νέες ταξινομημένες υπολίστες μέχρι να απομείνει μόνο μία υπολίστα. Αυτή θα είναι η ταξινομημένη λίστα.**

***Ταξινόμηση κατά αύξουσα σειρά των ημερομηνιών (πεδίο Date) με βάση τις τιμές ανοίγματος (πεδίο Open) των μετοχών με χρήση του αλγορίθμου QUICK SORT***

* Για το αρχείο *agn.us.txt* ο κώδικας είναι:

*#include <iostream>*

*#include"InputFile.h"*

*using namespace std;*

*/\* Αυτή η συνάρτηση ορίζει το τελευταίο στοιχείο ως pivot, τοποθετεί το στοιχείο που έχει ορίσει ως pivot στην σωστή θέση στον ταξινιμημένο πίνακα. Επίσης τοποθετεί όλα τα στοιχεία, τα οποία είναι μικρότερα από το pivot στα αριστερά και όλα τα μεγαλύτερα στα δεξιά.\*/*

*int partition (struct metoxh arr[], int low, int high)*

*{*

*int pivot = arr[high].Open; // pivot*

*int i = (low - 1); /\* Είναι το Index του μικρότερου στοιχείου που έχει βρεθεί μέχρι στιγμής και δείχνει την σωστή θέση του pivot \*/*

*for (int j = low; j <= high - 1; j++)*

*{*

*/\** *εάν το στοιχείο που εξετάζουμε είναι μικρότερο του pivot \*/*

*if (arr[j].Open < pivot)*

*{*

*i++; /\* αυξάνουμε το index του μικρότερου στοιχείου \*/*

*swapStructs(arr[i], arr[j]);*

*}*

*}*

*swapStructs(arr[i + 1], arr[high]);*

*return (i + 1);*

*}*

*/\** *Η βασική συνάρτηση που υλοποιεί τον QuickSort*

*arr[] --> ο πίνακας που πρέπει να ταξινομηθεί*

*low --> Starting index*

*high --> Ending index \*/*

*void quickSort(struct metoxh arr[], int low, int high)*

*{*

*if (low < high)*

*{*

*/\** *το pi είναι το index της διαμέρισης, arr[p] είναι τώρα στην σωστή θέση\*/*

*int pi = partition(arr, low, high);*

*/\** *Ξεχωριστή ταξινόμηση στοιχείων πριν τον χωρισμό του πίνακα και μετά.\*/*

*quickSort(arr, low, pi - 1);*

*quickSort(arr, pi + 1, high);*

*}*

*}*

*int main()*

*{*

*ifstream infile("agn.us.txt");*

*struct metoxh Stocks[AGN\_SIZE];*

*ReadValuesfromFile(infile, Stocks);*

*quickSort(Stocks, 0, AGN\_SIZE - 1);*

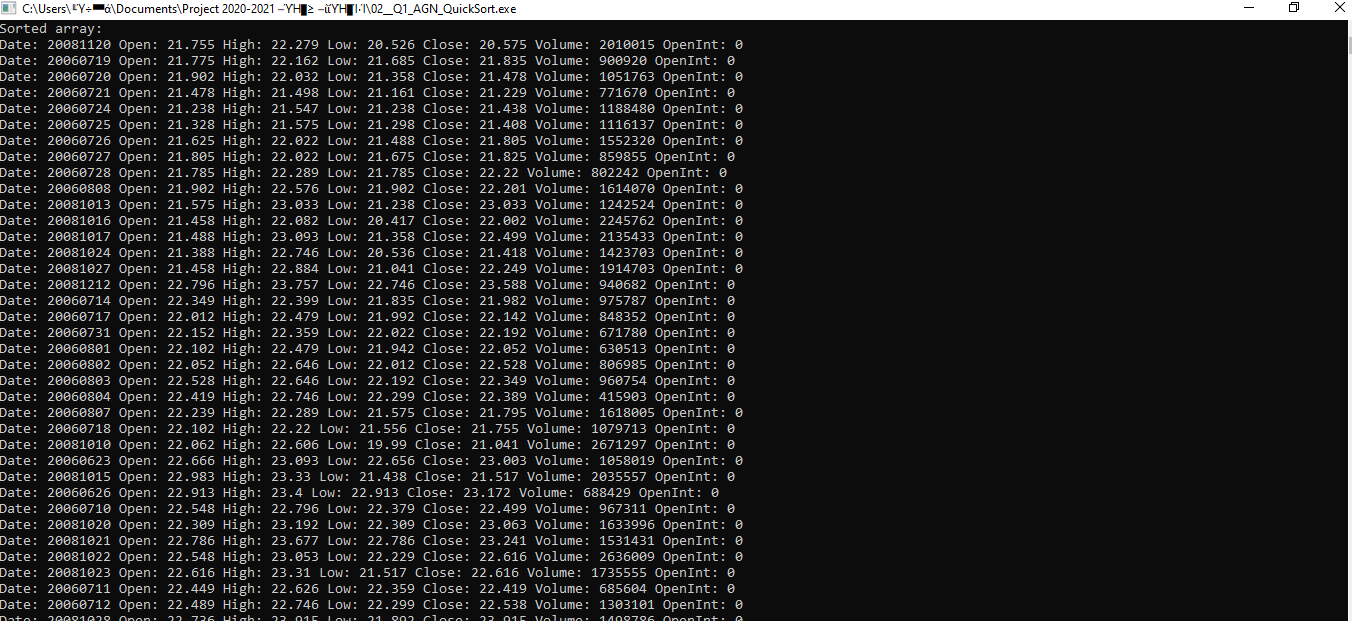
*cout << "Sorted array: \n";*

*PrintArray(Stocks, AGN\_SIZE); // Από το InputFile.h*

*return 0;*

*}*

Αν τρέξουμε τον κώδικα



* Για το αρχείο *ainv.us.txt* ο κώδικας είναι:

*#include <iostream>*

*#include"InputFile.h"*

*using namespace std;*

*/\* Αυτή η συνάρτηση ορίζει το τελευταίο στοιχείο ως pivot, τοποθετεί το στοιχείο που έχει ορίσει ως pivot στην σωστή θέση στον ταξινιμημένο πίνακα. Επίσης τοποθετεί όλα τα στοιχεία, τα οποία είναι μικρότερα από το pivot στα αριστερά και όλα τα μεγαλύτερα στα δεξιά.\*/*

*int partition (struct metoxh arr[], int low, int high)*

*{*

*int pivot = arr[high].Open; // pivot*

*int i = (low - 1); /\* Είναι το Index του μικρότερου στοιχείου που έχει βρεθεί μέχρι στιγμής και δείχνει την σωστή θέση του pivot\*/*

*for (int j = low; j <= high - 1; j++)*

*{*

*/\* εάν το στοιχείο που εξετάζουμε είναι μικρότερο του pivot \*/*

*if (arr[j].Open < pivot)*

*{*

*i++; /\** *αυξάνουμε το index του μικρότερου στοιχείου\*/*

*swapStructs(arr[i], arr[j]);*

*}*

*}*

*swapStructs(arr[i + 1], arr[high]);*

*return (i + 1);*

*}*

*/\** *Η βασική συνάρτηση που υλοποιεί τον QuickSort*

*arr[] --> ο πίνακας που πρέπει να ταξινομηθεί,*

*low --> Starting index,*

*high --> Ending index \*/*

*void quickSort(struct metoxh arr[], int low, int high)*

*{*

*if (low < high)*

*{*

*/\* το pi είναι το index της διαμέρισης, arr[p] είναι τώρα στην σωστή θέση\*/*

*int pi = partition(arr, low, high);*

*/\* Ξεχωριστή ταξινόμηση στοιχείων πριν τον χωρισμό του πίνακα και μετά.\*/*

*quickSort(arr, low, pi - 1);*

*quickSort(arr, pi + 1, high);*

*}*

*}*

*int main()*

*{*

*ifstream infile("ainv.us.txt");*

*struct metoxh Stocks[AINV\_SIZE];*

*ReadValuesfromFile(infile, Stocks);*

*quickSort(Stocks, 0, AINV\_SIZE - 1);*

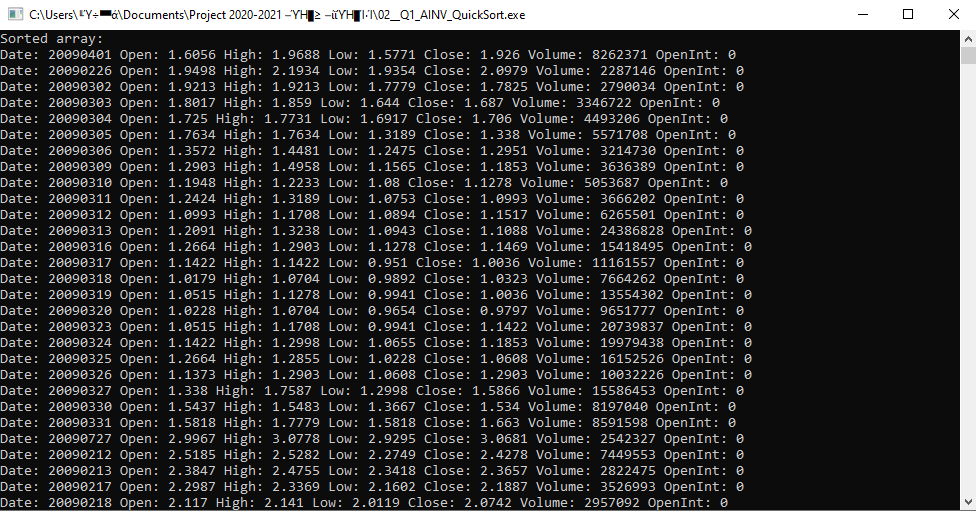
*cout << "Sorted array: \n";*

*PrintArray(Stocks, AINV\_SIZE); //Από το InputFile.h*

*return 0;*

*}*

Αν τρέξουμε τον κώδικα



* Για το αρχείο *ale.us.txt* ο κώδικας είναι:

*#include <iostream>*

*#include"InputFile.h"*

*using namespace std;*

*/\* Αυτή η συνάρτηση ορίζει το τελευταίο στοιχείο ως pivot, τοποθετεί το στοιχείο που έχει ορίσει ως pivot στην σωστή θέση στον ταξινιμημένο πίνακα. Επίσης τοποθετεί όλα τα στοιχεία, τα οποία είναι μικρότερα από το pivot στα αριστερά και όλα τα μεγαλύτερα στα δεξιά.\*/*

*int partition (struct metoxh arr[], int low, int high)*

*{*

*int pivot = arr[high].Open; // pivot*

*int i = (low - 1); /\* Είναι το Index του μικρότερου στοιχείου που έχει βρεθεί μέχρι στιγμής και δείχνει την σωστή θέση του pivot\*/*

*for (int j = low; j <= high - 1; j++)*

*{*

*/\* εάν το στοιχείο που εξετάζουμε είναι μικρότερο του pivot\*/*

*if (arr[j].Open < pivot)*

*{*

*i++; /\* αυξάνουμε το index του μικρότερου στοιχείου\*/*

*swapStructs(arr[i], arr[j]);*

*}*

*}*

*swapStructs(arr[i + 1], arr[high]);*

*return (i + 1);*

*}*

*/\* Η βασική συνάρτηση που υλοποιεί τον QuickSort*

*arr[] --> ο πίνακας που πρέπει να ταξινομηθεί,*

*low --> Starting index,*

*high --> Ending index \*/*

*void quickSort(struct metoxh arr[], int low, int high)*

*{*

*if (low < high)*

*{*

*/\* το pi είναι το index της διαμέρισης, arr[p] είναι τώρα στην σωστή θέση\*/*

*int pi = partition(arr, low, high);*

*/\** *Ξεχωριστή ταξινόμηση στοιχείων πριν τον χωρισμό του πίνακα και μετά.\*/*

*quickSort(arr, low, pi - 1);*

*quickSort(arr, pi + 1, high);*

*}*

*}*

*int main()*

*{*

*ifstream infile("ale.us.txt");*

*struct metoxh Stocks[ALE\_SIZE];*

*ReadValuesfromFile(infile, Stocks);*

*quickSort(Stocks, 0, ALE\_SIZE - 1);*

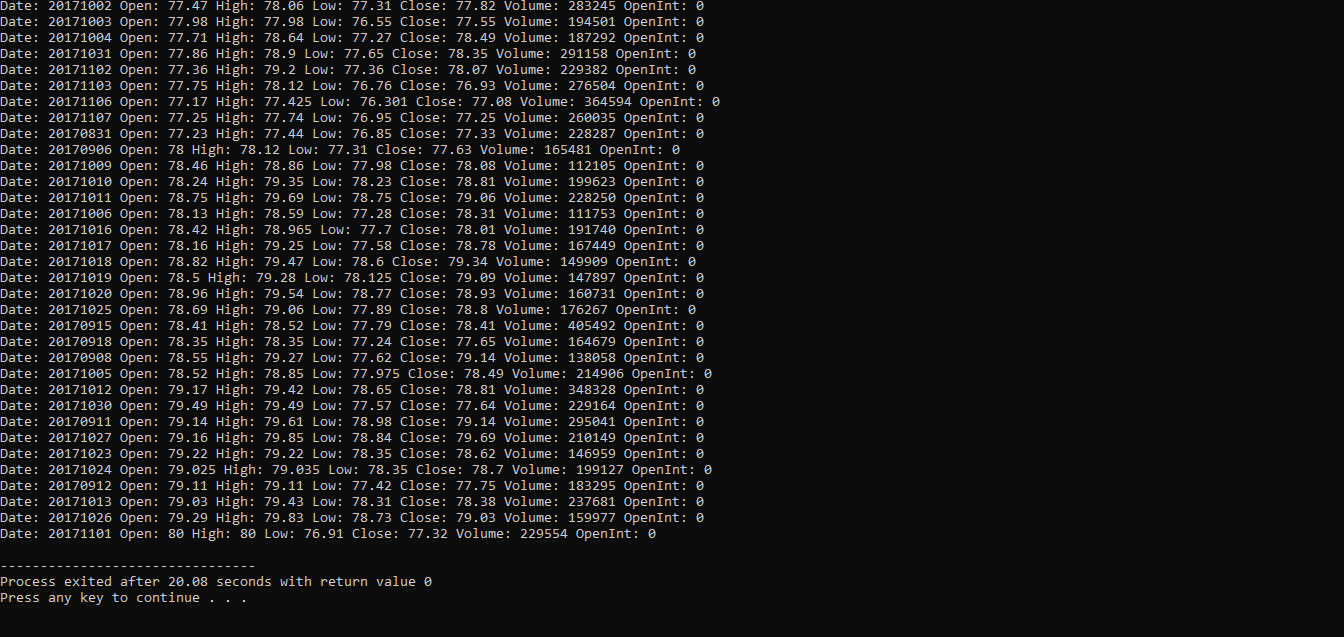
*cout << "Sorted array: \n";*

*PrintArray(Stocks, ALE\_SIZE); // Από το InputFile.h*

*return 0;*

*}*

Αν τρέξουμε τον κώδικα



**Στην ουσία με τον αλγόριθμο quicksort επιλέγουμε μια τιμή, που ονομάζεται pivot, η οποία βρίσκεται στον πίνακα που θέλουμε να ταξινομήσουμε (ο τρόπος επιλογής εξαρτάται από τις τιμές των στοιχείων που υπάρχουν στον πίνακα, συνήθως επιλέγουμε μια ενδιάμεση τιμή).**

**Στην συνέχεια χωρίζουμε τον πίνακα και αναδιατάσσουμε τα στοιχεία του, καθορίζοντας ένα σημείο διαίρεσης, έτσι ώστε όλα τα στοιχεία με τιμές μικρότερες από το pivot να έρχονται πριν από τη διαίρεση, ενώ όλα τα στοιχεία με τιμές μεγαλύτερες από το pivot να έρχονται μετά από αυτό ενώ τα στοιχεία που είναι ίσα με τον άξονα μπορούν να πάνε και προς τις δύο κατευθύνσεις.**

**Τέλος εφαρμόζουμε αναδρομικά την quicksort στην υποπεριοχή μέχρι το σημείο διαίρεσης και στην υποπεριοχή μετά από αυτό, αποκλείοντας ενδεχομένως και από τις δύο περιοχές το στοιχείο που ισούται με το pivot στο σημείο διαίρεσης**

**Οι διαφορές μεταξύ των δυο αλγορίθμων είναι οι εξής:**

* Ο διαχωρισμός μιας σειράς στοιχείων στον αλγόριθμο QuickSort είναι σε οποιαδήποτε αναλογία, όχι απαραίτητα διαιρεμένος σε μισό σε αντίθεση με τον MergeSort.
* Η χειρότερη περίπλοκη περίπτωση του QuickSort είναι η O(n2) καθώς υπάρχει ανάγκη πολλών συγκρίσεων στη χειρότερη κατάσταση ενώ, στον MergeSort ,η χειρότερη περίπτωση και η μέση περίπτωση έχουν τις ίδιες πολυπλοκότητες O(n logn).
* Η MergeSort δεν έχει τεθεί σε εφαρμογή επειδή απαιτεί επιπλέον χώρο μνήμης για την αποθήκευση των βοηθητικών συστοιχιών ενώ ο QuickSort είναι σε ισχύ, καθώς δεν απαιτεί επιπλέον χώρο αποθήκευσης.
* Ο MergeSort μπορεί να λειτουργήσει καλά σε οποιονδήποτε τύπο συνόλων δεδομένων ανεξάρτητα από το μέγεθός του (είτε μεγάλο είτε μικρό) ενώ ο QuickSort δεν μπορεί να λειτουργήσει καλά με μεγάλα σύνολα δεδομένων.
* Η QuickSort είναι μέθοδος εσωτερικής ταξινόμησης όπου τα δεδομένα ταξινομούνται στην κύρια μνήμη ενώ η MergeSort είναι μια εξωτερική μέθοδος ταξινόμησης στην οποία τα δεδομένα που πρόκειται να ταξινομηθούν δεν μπορούν να προσαρμοστούν στη μνήμη και χρειάζονται βοηθητική μνήμη για ταξινόμηση.
* Η MergeSort είναι σταθερή καθώς δύο στοιχεία με ίση τιμή εμφανίζονται με την ίδια σειρά στην ταξινομημένη έξοδο όπως ήταν στην παράταξη χωρίς ταξινόμηση ενώ το Quick sort είναι ασταθές σε αυτό το σενάριο. Αλλά μπορεί να γίνει σταθερό χρησιμοποιώντας κάποιες αλλαγές στον κώδικα.

**ΕΡΩΤΗΜΑ 2Ο**

***Ταξινόμηση κατά αύξουσα σειρά των ημερομηνιών (πεδίο Date) με βάση τις τιμές κλεισίματος (πεδίο Close) των μετοχών με χρήση του αλγορίθμου HEAP SORT***

* Για το αρχείο *agn.us.txt* ο κώδικας είναι:

*#include<cstdlib>*

*#include<iostream>*

*#include<fstream>*

*#include"InputFile.h"*

*using namespace std;*

*/\* Κάνουμε heapify το subtree, το οποίο έχει ρίζα τον κόμβο i .*

*Ο κόμβος i είναι ένα index στον πίνακα arr[] και το n το μέγεθος του σωρού\*/*

*void heapify(struct metoxh StocksArray[], int n, int i)*

*{*

*int largest = i;* *// Αρχικοποιώ το μεγαλύτερο i ως την ρίζα*

*int l = 2 \* i + 1; // left = 2\*i + 1*

*int r = 2 \* i + 2; // right = 2\*i + 2*

*// Εάν το αριστερό παιδί είναι μεγαλύτερο από την ρίζα*

*if (l < n && StocksArray[l].Close > StocksArray[largest].Close)*

*largest = l;*

*// Εάν το δεξιό παιδί είναι μεγαλύτερο από την ρίζα*

*if (r < n && StocksArray[r].Close > StocksArray[largest].Close)*

*largest = r;*

*// Εάν το μεγαλύτερο στοιχείο δεν είναι η ρίζα*

*if (largest != i) {*

*swap(StocksArray[i], StocksArray[largest]);*

*// Κάνε αναδρομικό heapify στο ήδη αλλαγμένο sub-tree*

*heapify(StocksArray, n, largest);*

*}*

*}*

*// Η βασική συνάρτηση που υλοποιεί το heap sort*

*void heapSort(struct metoxh StocksArray[], int n)*

*{*

*// Φτιάχνω τον σωρό (rearrange array)*

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

*heapify(StocksArray, n, i);*

*// Εξάγονται ένα ένα τα στοιχεία από τον σωρό*

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

*// Μεταφέρουμε την τρέχουσα ρίζα στο τέλος*

*swapStructs(StocksArray[0], StocksArray[i]);*

*// Καλούμε την max heapify συνάρτηση στον ήδη μειωμένο σωρό*

*heapify(StocksArray, i, 0);*

*}*

*}*

*/\* Είναι μια χρήσιμη συνάρτηση για να εκτυπώσεις πίνακα μεγέθους n \*/*

*void printArray(struct metoxh StocksArray[], int n)*

*{*

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

*cout <<"Date: "<< StocksArray[i].Date <<" Open: "<< StocksArray[i].Open <<" High: "<< StocksArray[i].High <<" Low: "<< StocksArray[i].Low <<" Close: "<< StocksArray[i].Close <<" Volume: "<< StocksArray[i].Volume <<" OpenInt: "<< StocksArray[i].OpenInt<<endl;*

*}*

*// Driver code*

*int main()*

*{*

*ifstream infile("agn.us.txt");*

*struct metoxh Stocks[AGN\_SIZE];*

*ReadValuesfromFile(infile, Stocks);*

*int n = sizeof(Stocks) / sizeof(Stocks[0]);*

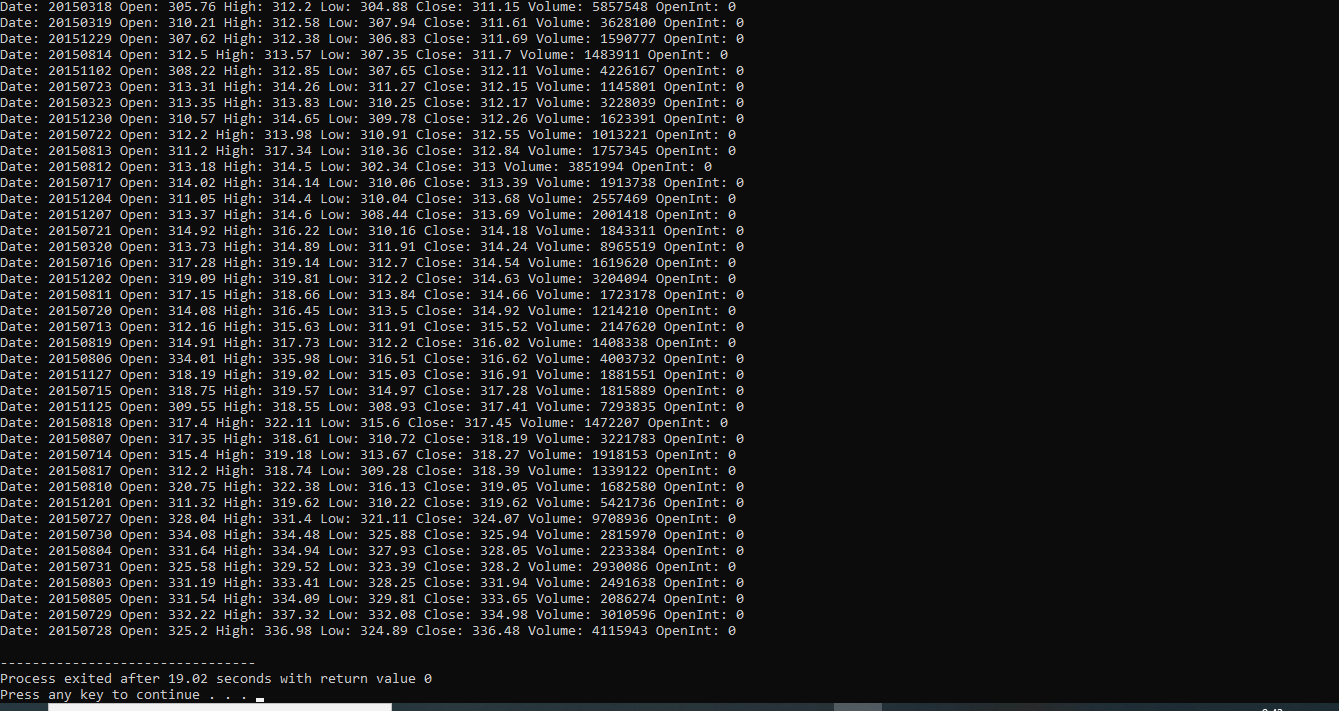
*heapSort(Stocks, n);*

*cout << "Sorted array is \n";*

*printArray(Stocks, n);*

*}*

Αν τρέξουμε τον κώδικα



* Για το αρχείο *ainv.us.txt* ο κώδικας είναι:

*#include<cstdlib>*

*#include<iostream>*

*#include<fstream>*

*#include"InputFile.h"*

*using namespace std;*

*/\* Κάνουμε heapify το subtree, το οποίο έχει ρίζα τον κόμβο i*

*Ο κόμβος i είναι ένα index στον πίνακα arr[] και το n το μέγεθος του σωρού\*/*

*void heapify(struct metoxh StocksArray[], int n, int i)*

*{*

*int largest = i; // Αρχικοποιώ το μεγαλύτερο i ως την ρίζα*

*int l = 2 \* i + 1; // left = 2\*i + 1*

*int r = 2 \* i + 2; // right = 2\*i + 2*

*// Εάν το αριστερό παιδί είναι μεγαλύτερο από την ρίζα*

*if (l < n && StocksArray[l].Close > StocksArray[largest].Close)*

*largest = l;*

*// Εάν το δεξιό παιδί είναι μεγαλύτερο από την ρίζα*

*if (r < n && StocksArray[r].Close > StocksArray[largest].Close)*

*largest = r;*

*// Εάν το μεγαλύτερο στοιχείο δεν είναι η ρίζα*

*if (largest != i) {*

*swap(StocksArray[i], StocksArray[largest]);*

*// Κάνε αναδρομικό heapify στο ήδη αλλαγμένο sub-tree*

*heapify(StocksArray, n, largest);*

*}*

*}*

*// Η βασική συνάρτηση που υλοποιεί το heap sort*

*void heapSort(struct metoxh StocksArray[], int n)*

*{*

*// Φτιάχνω τον σωρό (rearrange array)*

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

*heapify(StocksArray, n, i);*

*// Εξάγονται ένα ένα τα στοιχεία από τον σωρό*

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

*// Μεταφέρουμε την τρέχουσα ρίζα στο τέλος*

*swapStructs(StocksArray[0], StocksArray[i]);*

*// Καλούμε την max heapify συνάρτηση στον ήδη μειωμένο σωρό*

*heapify(StocksArray, i, 0);*

*}*

*}*

*/\* Είναι μια χρήσιμη συνάρτηση για να εκτυπώσεις πίνακα μεγέθους n \*/*

*void printArray(struct metoxh StocksArray[], int n)*

*{*

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

*cout <<"Date: "<< StocksArray[i].Date <<" Open: "<< StocksArray[i].Open <<" High: "<< StocksArray[i].High <<" Low: "<< StocksArray[i].Low <<" Close: "<< StocksArray[i].Close <<" Volume: "<< StocksArray[i].Volume <<" OpenInt: "<< StocksArray[i].OpenInt<<endl;*

*}*

*// Driver code*

*int main()*

*{*

*ifstream infile("ainv.us.txt");*

*struct metoxh Stocks[AINV\_SIZE];*

*ReadValuesfromFile(infile, Stocks);*

*int n = sizeof(Stocks) / sizeof(Stocks[0]);*

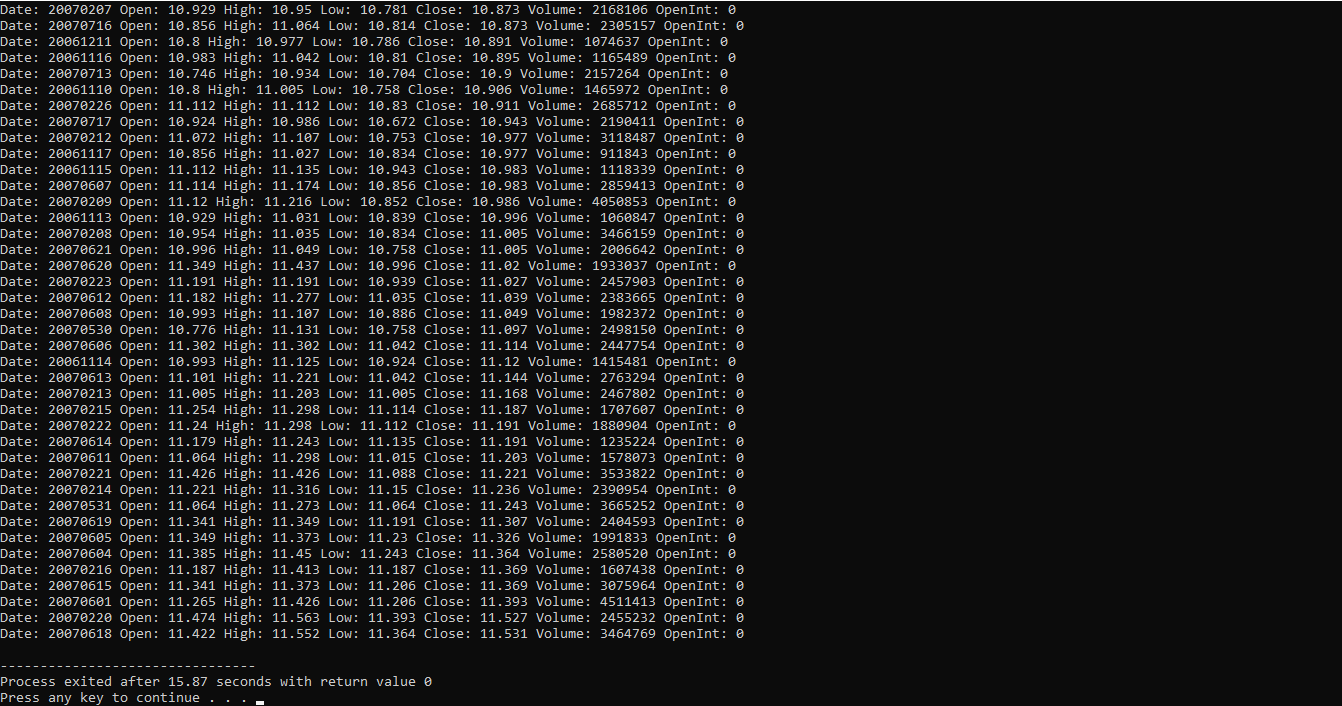
*heapSort(Stocks, n);*

*cout << "Sorted array is \n";*

*printArray(Stocks, n);*

*}*

Αν τρέξουμε τον κώδικα



* Για το αρχείο *ale.us.txt* ο κώδικας είναι:

*#include<cstdlib>*

*#include<iostream>*

*#include<fstream>*

*#include"InputFile.h"*

*using namespace std;*

*/\* Κάνουμε heapify το subtree, το οποίο έχει ρίζα τον κόμβο i*

*Ο κόμβος i είναι ένα index στον πίνακα arr[] και το n το μέγεθος του σωρού\*/*

*void heapify(struct metoxh StocksArray[], int n, int i)*

*{*

*int largest = i; // Αρχικοποιώ το μεγαλύτερο i ως την ρίζα*

*int l = 2 \* i + 1; // left = 2\*i + 1*

*int r = 2 \* i + 2; // right = 2\*i + 2*

*// Εάν το αριστερό παιδί είναι μεγαλύτερο από την ρίζα*

*if (l < n && StocksArray[l].Close > StocksArray[largest].Close)*

*largest = l;*

*// Εάν το δεξιό παιδί είναι μεγαλύτερο από την ρίζα*

*if (r < n && StocksArray[r].Close > StocksArray[largest].Close)*

*largest = r;*

*// Εάν το μεγαλύτερο στοιχείο δεν είναι η ρίζα*

*if (largest != i) {*

*swap(StocksArray[i], StocksArray[largest]);*

*// Κάνε αναδρομικό heapify στο ήδη αλλαγμένο sub-tree*

*heapify(StocksArray, n, largest);*

*}*

*}*

*// Η βασική συνάρτηση που υλοποιεί το heap sort*

*void heapSort(struct metoxh StocksArray[], int n)*

*{*

*// Φτιάχνω τον σωρό (rearrange array)*

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

*heapify(StocksArray, n, i);*

*// Εξάγονται ένα ένα τα στοιχεία από τον σωρό*

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

*// Μεταφέρουμε την τρέχουσα ρίζα στο τέλος*

*swapStructs(StocksArray[0], StocksArray[i]);*

*// Καλούμε την max heapify συνάρτηση στον ήδη μειωμένο σωρό*

*heapify(StocksArray, i, 0);*

*}*

*}*

*/\* Είναι μια χρήσιμη συνάρτηση για να εκτυπώσεις πίνακα μεγέθους n \*/*

*void printArray(struct metoxh StocksArray[], int n)*

*{*

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

*cout <<"Date: "<< StocksArray[i].Date <<" Open: "<< StocksArray[i].Open <<" High: "<< StocksArray[i].High <<" Low: "<< StocksArray[i].Low <<" Close: "<< StocksArray[i].Close <<" Volume: "<< StocksArray[i].Volume <<" OpenInt: "<< StocksArray[i].OpenInt<<endl;*

*}*

*// Driver code*

*int main()*

*{*

*ifstream infile("ale.us.txt");*

*struct metoxh Stocks[ALE\_SIZE];*

*ReadValuesfromFile(infile, Stocks);*

*int n = sizeof(Stocks) / sizeof(Stocks[0]);*

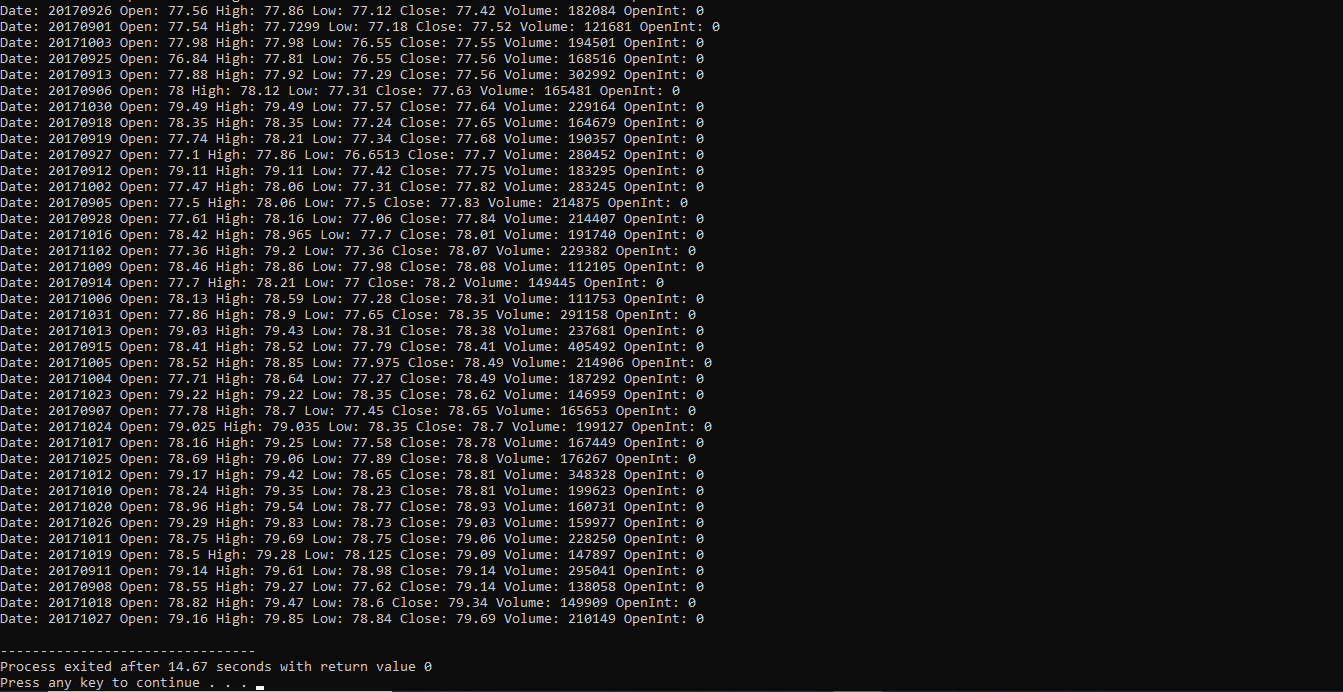
*heapSort(Stocks, n);*

*cout << "Sorted array is \n";*

*printArray(Stocks, n);*

*}*

Αν τρέξουμε τον κώδικα



**Με τον αλγόριθμο Heapsort μετατρέπουμε πρώτα την λίστα σε μέγιστο σωρό. Στη συνέχεια, ο αλγόριθμος ανταλλάσσει επανειλημμένα την πρώτη τιμή της λίστας με την τελευταία τιμή, μειώνοντας το εύρος των τιμών που λαμβάνονται υπόψη στη λειτουργία σωρού κατά ένα. Μετά μεταφέρει τη νέα πρώτη τιμή στη θέση της στο σωρό. Αυτό επαναλαμβάνεται έως ότου το εύρος των εξεταζόμενων τιμών φτάσει σε μήκος μία τιμή.**

**Πιο συγκεκριμένα καλούμε τη συνάρτηση heapify(), και δημιουργούμε έναν σωρό από μια λίστα σε O(n) πράξεις.**

**Στην συνέχεια κάνουμε αντικατάσταση το πρώτο στοιχείο της λίστας με το τελευταίο στοιχείο και μειώνουμε το εύρος της λίστας κατά ένα.**

***Ταξινόμηση κατά αύξουσα σειρά των ημερομηνιών (πεδίο Date) με βάση τις τιμές κλεισίματος (πεδίο Close) των μετοχών με χρήση του αλγορίθμου COUNTING SORT***

* Για το αρχείο *agn.us.txt* ο κώδικας είναι:

*#include <iostream>*

*#include”InputFile.h”*

*using namespace std;*

*void countSort(struct metoxh StocksArray[], int size) {*

*/\* Το μέγεθος του count πρέπει να είναι τουλάχιστον το (max+1). Δυστυχώς όμως δεν μπορούμε δηλώσουμε την μεταβλητή αυτή ως int count(max+1) στην C++ , γιατί δεν υποστηρίζει δυναμική κατανομή μνήμης\*/*

*/\*Έτσι το μέγεθος της θα είναι στατικό\*/*

*struct metoxh output[3500];*

*int count[3500];*

*float max = StocksArray[0].Close;*

*//cout<<”OK\_4\n”;*

*//Βρίσκω το μεγαλύτερο στοιχείο του πίνακα*

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

*if (StocksArray[i].Close > max)*

*max = StocksArray[i].Close;*

*}*

*//cout<<”OK\_5\n”;*

*//Αρχικοποιώ τον count array με μηδενικά παντού*

*for (int i = 0; i <= max; ++i) {*

*count[i] = 0;*

*}*

*// cout<<»OK\_6\n»;*

*// Αποθηκεύουμε το count του κάθε στοιχείου*

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

*count[(int)StocksArray[i].Close]++;*

*}*

*// cout<<»OK\_7\n»;*

*// Αποθηκεύουμε το συσσωρευμένο count του κάθε πίνακα*

*for (int i = 1; i <= max; i++) {*

*count[i] += count[i – 1];*

*}*

*// cout<<»OK\_8\n»;*

*/\* Βρίσκω το index του κάθε στοιχείου του αρχικού πίνακα στον count array, και τοποθετώ τα στοιχεία στον output array\*/*

*for (int i = size – 1; i >= 0; i--) {*

*StructAssignTo(StocksArray[i], output[count[(int)StocksArray[i].Close] – 1]); //output[count[StocksArray[i]] – 1] = StocksArray[i];*

*count[(int)StocksArray[i].Close]--;*

*}*

*// cout<<»OK\_9\n»;*

*/\* Αντιγράφουμε τα ταξινομημένα στοιχεία στον αρχικό(original) πίνακα \*/*

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

*StocksArray[i] = output[i];*

*}*

*// cout<<”OK\_10\n”;*

*}*

*// Συνάρτηση για να εκτυπώσεις έναν πίνακα*

*/\*void printArray(int array[], int size) {*

*for (int i = 0; i < size; i++)*

*cout << array[i] << “ “;*

*cout << endl;*

*}\*/*

*// Driver code*

*int main() {*

*ifstream infile(“agn.us.txt”);*

*// cout<<”OK\_1\n”;*

*struct metoxh Stocks[AGN\_SIZE];*

*// cout<<”OK\_2\n”;*

*ReadValuesfromFile(infile, Stocks);*

*// cout<<”OK\_3\n”;*

*countSort(Stocks, AGN\_SIZE);*

*// cout<<”OK\_\n”;*

*cout << “Sorted array is \n”;*

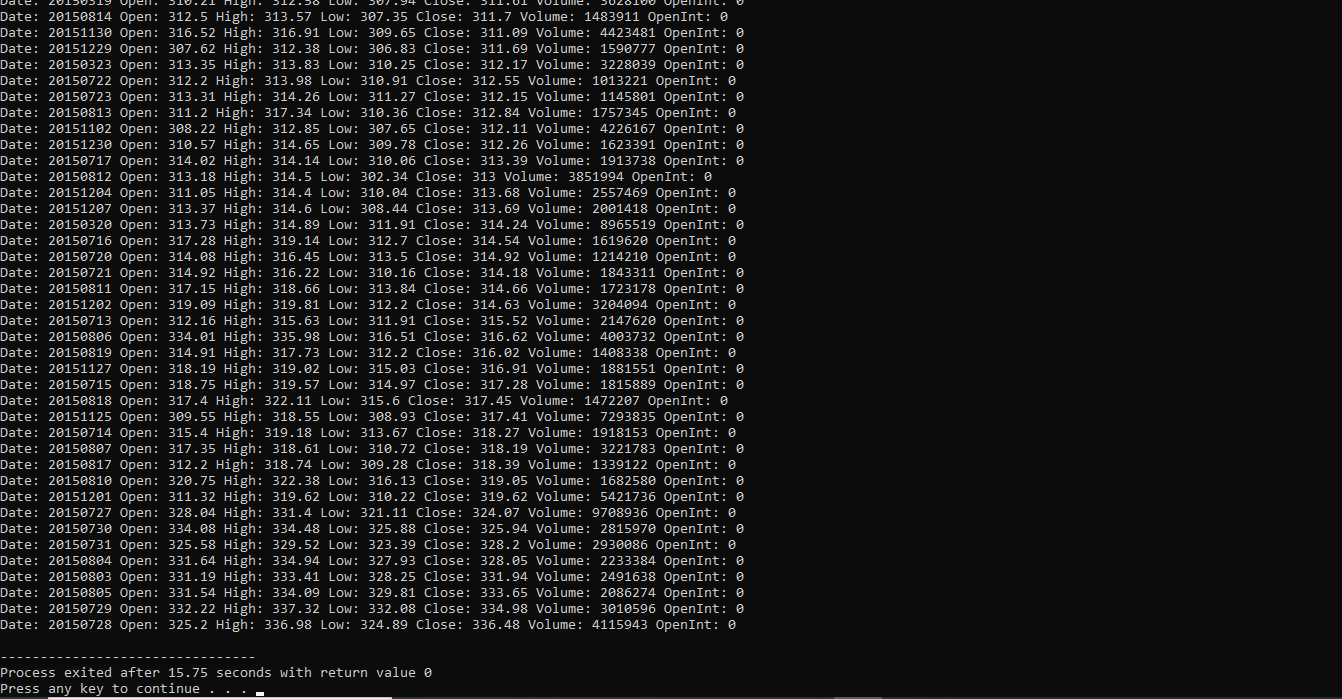
*// cout<<”OK\_\n”;*

*PrintArray(Stocks, AGN\_SIZE);*

*// cout<<»OK\_\n»;*

*}*

Αν τρέξουμε τον κώδικα



* Για το αρχείο *ainv.us.txt* ο κώδικας είναι:

*// Counting sort in C++ programming*

*#include <iostream>*

*#include”InputFile.h”*

*using namespace std;*

*void countSort(struct metoxh StocksArray[], int size) {*

*/\* Το μέγεθος του count πρέπει να είναι τουλάχιστον το (max+1). Δυστυχώς όμως δεν μπορούμε δηλώσουμε την μεταβλητή αυτή ως int count(max+1) στην C++ , γιατί δεν υποστηρίζει δυναμική κατανομή μνήμης\*/*

*/\*Έτσι το μέγεθος της θα είναι στατικό\*/*

*struct metoxh output[3500];*

*int count[3500];*

*float max = StocksArray[0].Close;*

*//cout<<”OK\_4\n”;*

*//Βρίσκω το μεγαλύτερο στοιχείο του πίνακα*

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

*if (StocksArray[i].Close > max)*

*max = StocksArray[i].Close;*

*}*

*//cout<<”OK\_5\n”;*

*//Αρχικοποιώ τον count array με μηδενικά παντού*

*for (int i = 0; i <= max; ++i) {*

*count[i] = 0;*

*}*

*// cout<<»OK\_6\n»;*

*// Αποθηκεύουμε το count του κάθε στοιχείου*

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

*count[(int)StocksArray[i].Close]++;*

*}*

*// cout<<»OK\_7\n»;*

*// Αποθηκεύουμε το συσσωρευμένο count του κάθε πίνακα*

*for (int i = 1; i <= max; i++) {*

*count[i] += count[i – 1];*

*}*

*// cout<<»OK\_8\n»;*

*/\* Βρίσκω το index του κάθε στοιχείου του αρχικού πίνακα στον count array, και τοποθετώ τα στοιχεία στον output array\*/*

*for (int i = size – 1; i >= 0; i--) {*

*StructAssignTo(StocksArray[i], output[count[(int)StocksArray[i].Close] – 1]); //output[count[StocksArray[i]] – 1] = StocksArray[i];*

*count[(int)StocksArray[i].Close]--;*

*}*

*// cout<<»OK\_9\n»;*

*/\* Αντιγράφουμε τα ταξινομημένα στοιχεία στον αρχικό(original) πίνακα\*/*

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

*StocksArray[i] = output[i];*

*}*

*// cout<<”OK\_10\n”;*

*}*

*// Συνάρτηση για να εκτυπώσεις έναν πίνακα*

*/\*void printArray(int array[], int size) {*

*for (int i = 0; i < size; i++)*

*cout << array[i] << “ “;*

*cout << endl;*

*}\*/*

*// Driver code*

*int main() {*

*ifstream infile(“ainv.us.txt”);*

*// cout<<”OK\_1\n”;*

*struct metoxh Stocks[AINV\_SIZE];*

*// cout<<”OK\_2\n”;*

*ReadValuesfromFile(infile, Stocks);*

*// cout<<”OK\_3\n”;*

*countSort(Stocks, AINV\_SIZE);*

*// cout<<”OK\_\n”;*

*cout << “Sorted array is \n”;*

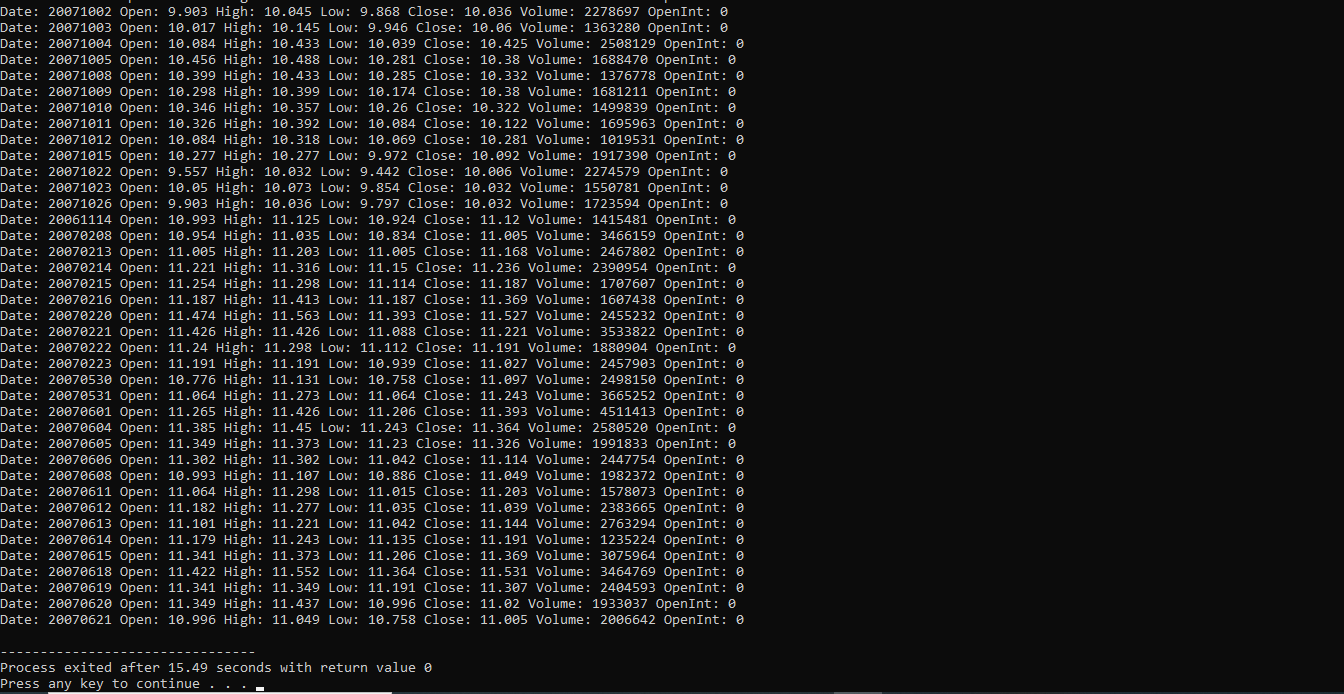
*// cout<<”OK\_\n”;*

*PrintArray(Stocks, AINV\_SIZE);*

*// cout<<»OK\_\n»;*

*}*

Αν τρέξω τον κώδικα



* Για το αρχείο *ale.us.txt* ο κώδικας είναι:

*// Counting sort in C++ programming*

*#include <iostream>*

*#include”InputFile.h”*

*using namespace std;*

*void countSort(struct metoxh StocksArray[], int size) {*

*/\* Το μέγεθος του count πρέπει να είναι τουλάχιστον το (max+1). Δυστυχώς όμως δεν μπορούμε δηλώσουμε την μεταβλητή αυτή ως int count(max+1) στην C++ , γιατί δεν υποστηρίζει δυναμική κατανομή μνήμης\*/*

*/\*Έτσι το μέγεθος της θα είναι στατικό\*/*

*struct metoxh output[3500];*

*int count[3500];*

*float max = StocksArray[0].Close;*

*//cout<<”OK\_4\n”;*

*//Βρίσκω το μεγαλύτερο στοιχείο του πίνακα*

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

*if (StocksArray[i].Close > max)*

*max = StocksArray[i].Close;*

*}*

*//cout<<”OK\_5\n”;*

*//Αρχικοποιώ τον count array με μηδενικά παντού*

*for (int i = 0; i <= max; ++i) {*

*count[i] = 0;*

*}*

*// cout<<»OK\_6\n»;*

*// Αποθηκεύουμε το count του κάθε στοιχείου*

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

*count[(int)StocksArray[i].Close]++;*

*}*

*// cout<<»OK\_7\n»;*

*// Αποθηκεύουμε το συσσωρευμένο count του κάθε πίνακα*

*for (int i = 1; i <= max; i++) {*

*count[i] += count[i – 1];*

*}*

*// cout<<»OK\_8\n»;*

*/\* Βρίσκω το index του κάθε στοιχείου του αρχικού πίνακα στον count array, και τοποθετώ τα στοιχεία στον output array\*/*

*for (int i = size – 1; i >= 0; i--) {*

*StructAssignTo(StocksArray[i], output[count[(int)StocksArray[i].Close] – 1]); //output[count[StocksArray[i]] – 1] = StocksArray[i];*

*count[(int)StocksArray[i].Close]--;*

*}*

*// cout<<»OK\_9\n»;*

*/\* Αντιγράφουμε τα ταξινομημένα στοιχεία στον αρχικό(original) πίνακα\*/*

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

*StocksArray[i] = output[i];*

*}*

*// cout<<”OK\_10\n”;*

*}*

*// Συνάρτηση για να εκτυπώσεις έναν πίνακα*

*/\*void printArray(int array[], int size) {*

*for (int i = 0; i < size; i++)*

*cout << array[i] << “ “;*

*cout << endl;*

*}\*/*

*// Driver code*

*int main() {*

*ifstream infile(“ale.us.txt”);*

*// cout<<”OK\_1\n”;*

*struct metoxh Stocks[ALE\_SIZE];*

*// cout<<”OK\_2\n”;*

*ReadValuesfromFile(infile, Stocks);*

*// cout<<”OK\_3\n”;*

*countSort(Stocks, ALE\_SIZE);*

*// cout<<”OK\_\n”;*

*cout << “Sorted array is \n”;*

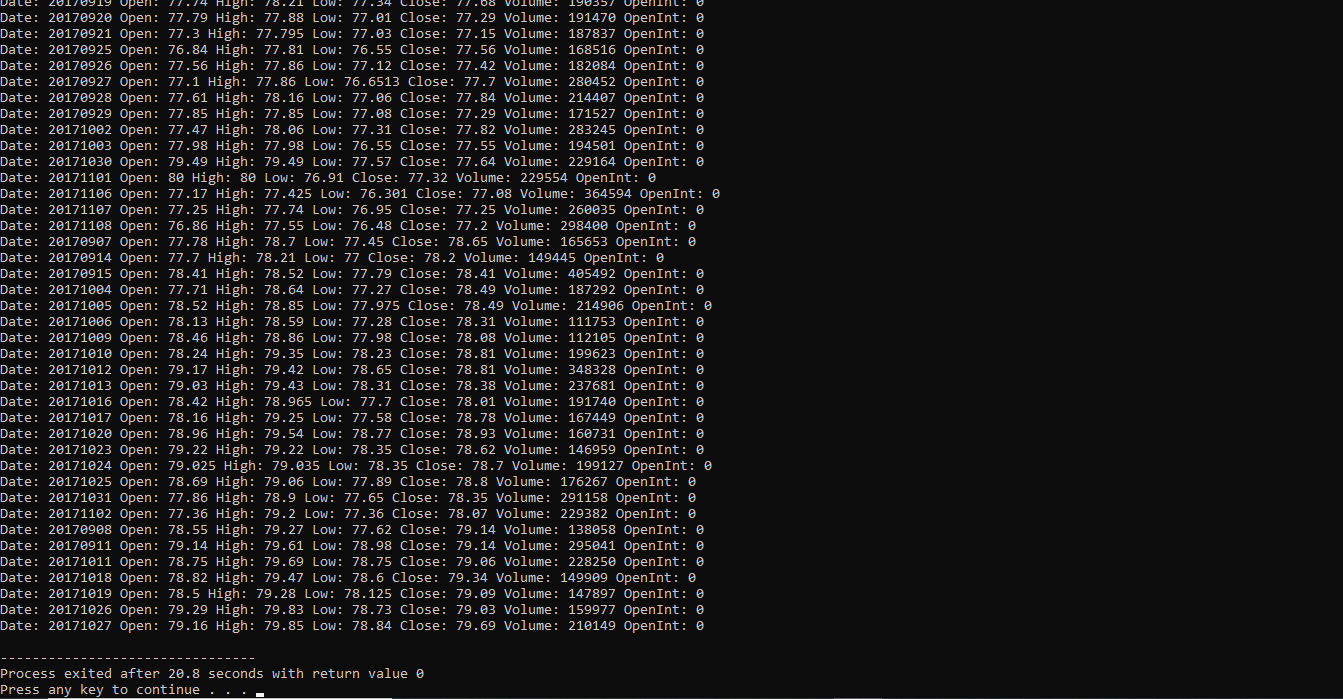
*// cout<<”OK\_\n”;*

*PrintArray(Stocks, ALE\_SIZE);*

*// cout<<»OK\_\n»;*

*}*

Αν τρέξω τον κώδικα



**Στην ουσία ο counting sort είναι ένας αλγόριθμος για την ταξινόμηση μιας συλλογής αντικειμένων σύμφωνα με κλειδιά που είναι ακέραιοι αριθμοί, δηλαδή ένας αλγόριθμος ταξινόμησης ακέραιων αριθμών. Λειτουργεί μετρώντας τον αριθμό των αντικειμένων που έχουν ξεχωριστή τιμή κλειδιού και χρησιμοποιώντας αριθμητική σε αυτές τις μετρήσεις για να καθορίσει τις θέσεις κάθε τιμής κλειδιού στην ακολουθία εξόδου. Ο χρόνος εκτέλεσης είναι γραμμικός ανάλογα με τον αριθμό των στοιχείων και τη διαφορά μεταξύ της μέγιστης και της ελάχιστης τιμής του κλειδιού, οπότε είναι κατάλληλο για άμεση χρήση μόνο σε περιπτώσεις όπου η διακύμανση των κλειδιών δεν είναι σημαντικά μεγαλύτερη από τον αριθμό των στοιχείων.**

**Οι διαφορές μεταξύ των δυο αλγορίθμων είναι οι εξής:**

* Ο αλγόριθμος heap sort έχει στην χειρότερη περίπτωση χρονική πολυπλοκότητα Ο(NlogN) ενώ ο counting sort Θ(Ν+Κ).
* Στην καλύτερη περίπτωση ο heap sort έχει χρονική πολυπλοκότητα Ω(ΝlogN) ενώ ο counting sort O(N+K). Πιο συγκεκριμένα στον counting sort η αρχικοποίηση του πίνακα μετρήσεων και ο δεύτερος βρόχος for που εκτελεί ένα άθροισμα προθέματος στον πίνακα μετρήσεων, επαναλαμβάνονται το πολύ Κ+1 φορές και συνεπώς χρειάζονται O(Κ) χρόνο. Οι άλλοι δύο βρόχοι for, καθώς και η αρχικοποίηση του πίνακα εξόδου, χρειάζονται χρόνο O(Ν). Επομένως, ο χρόνος για ολόκληρο τον αλγόριθμο είναι το άθροισμα των χρόνων για αυτά τα βήματα, O(Ν + Κ).
* Ο heap sort είναι ο καλύτερος αλγόριθμος σε απόδοση. Αυτό σημαίνει ότι κανένας άλλος αλγόριθμος ταξινόμησης δεν μπορεί να αποδώσει καλύτερα σε σύγκριση. Πιο συγκεκριμένα αποδίδει εξίσου καλά στην καλύτερη, τη μέση και τη χειρότερη περίπτωση.
* Ο counting sort είναι σχετικά σταθερός αλγόριθμος σε σχέση με τον heapsort. Πιο συγκεκριμένα μια σταθερή ταξινόμηση διατηρεί τη σχετική σειρά των στοιχείων που έχουν το ίδιο κλειδί, δηλαδή με τον τρόπο που υπάρχουν στον αρχικό πίνακα. Ο Heapsort είναι ασταθής ταξινόμηση. Μπορεί να αναδιατάξει τη σχετική σειρά
* Ο counting sort δεν είναι κατάλληλος για την ταξινόμηση μεγάλων συνόλων δεδομένων και τιμών συμβολοσειράς.

**ΕΡΩΤΗΜΑ 3Ο**

***Εύρεση συνολικού όγκου συναλλαγών (πεδίο Volume) στις μετοχές του αρχείου για συγκεκριμένη ημερομηνία (Date) με την βόηθεια της δυαδικής αναζήτησης***

* Για το αρχείο *agn.us.txt* ο κώδικας είναι:

*#include <iostream>*

*#include <cstdlib>*

*#include <fstream>*

*#include "F:\==KONSTANTINOS==\CEID 4О еналгмо\долес дедолемым\PROJECTS\_2020\_2021\MY PROGRAMMS\InputFile.h"*

*int BinarySearch(struct metoxh array[], int first, int middle, int last, long int WantedDate){*

*first = 0; //Ορίζουμε το πρώτο μας index*

*last = TEST\_SIZE - 1;* *//Ορίζουμε το τελευταίο μας index*

*middle = (first+last)/2; /\*Ορίζουμε το μεσαίο index που θα καθορίζει σε ποιο μισό του πίνακα θα πρέπει να ψάξουμε \*/*

*bool found = false;*

*while (first <= last) {*

*if(array[middle].Date < WantedDate)* */\*Εάν η ημερομηνία που δείχνει* *το middle index είναι μικρότερη από αυτή που ψάχνουμε\*/*

*{*

*first = middle + 1; /\*θα ορίσουμε ξανά νέο first γιατί ψάχνουμε τώρα στο δεύτερο μισό του πίνακα αφού το στοιχείο μας είναι μεγαλύτερο από αυτό που δείχνει το middle index \*/*

*middle = (first + last)/2; // όμοια με το first παραπάνω*

*//cout<<"Right"<<endl;*

*}*

*else if (array[middle].Date == WantedDate) /\*Εάν το στοιχείο που ψάχνουμε είναι ίσο με αυτό που δείχνει το middle index τότε το βρήκαμε\*/*

*{*

*cout << "H hmeromhnia poy pliktrologisate breuhke sth thesh " << middle;*

*cout<< "\nVolume of this date is:" << array[middle].Volume;*

*found = true;*

*return middle;*

*}*

*else /\*Εάν η ημερομηνία που δείχνει το middle index είναι μεγαλύτερη από αυτή που ψάχνουμε\*/*

*{*

*last = middle - 1; //ορίζω ξανά νέο last*

*middle = (first + last)/2; // ορίζω ξανά νέο middle*

*//cout<<"Left"<<endl;*

*}*

*}*

*if (!found){*

*cout << "Date Not Found!!!"<<endl;*

*}*

*return -1;*

*}*

*int main()*

*{*

*struct metoxh binarysearcharray[AGN\_SIZE];*

*ifstream fin("agn.us.txt");*

*ReadValuesfromFile(fin, binarysearcharray);*

*//mergeSortdifferent(binarysearcharray, 0, TEST\_SIZE - 1);*

*int first = 0; /\*Ορίζουμε το πρώτο μας index στο αρχείο ,του οποίου θέλουμε να ψάξουμε μια ημερομηνία\*/*

*int last = AGN\_SIZE-1; //Ορίζουμε το τελευταίο μας index*

*int middle = (first + last)/2; /\*Ορίζουμε το μεσαίο index που θα καθορίζει σε ποιο μισό του πίνακα θα πρέπει να ψάξουμε\*/*

*long int fulldate;*

*fulldate = getDatefromKeyboard(); /\* εισάγουμε την ημερομηνία που θέλουμε να βρούμε\*/*

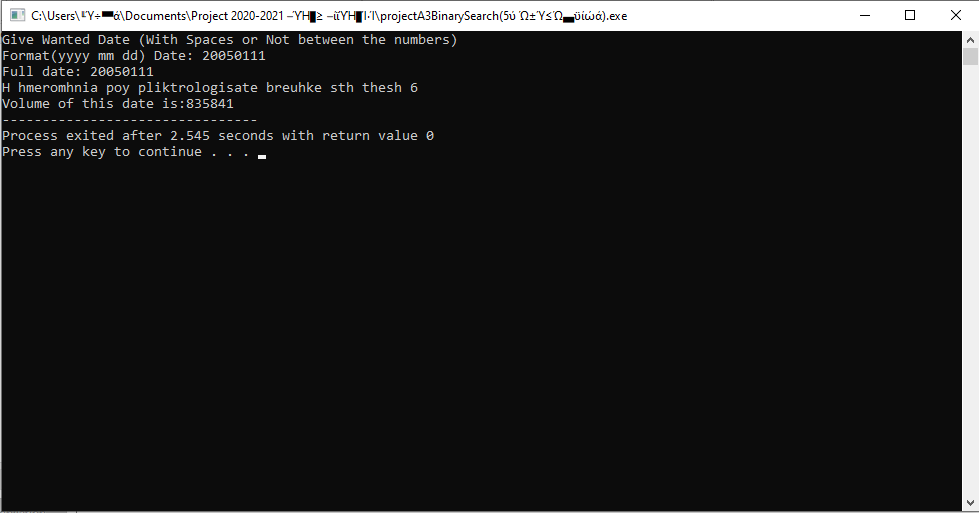
*cout<<"Full date: "<<fulldate<<endl;*

*BinarySearch(binarysearcharray , first, middle, last, fulldate); /\*Καλούμε την συνάρτηση για να γίνει η αναζήτηση\*/*

*return 0;*

*}*

Αν τρέξουμε τον κώδικα θα έχουμε τα εξής αποτελέσματα:



**Η δυαδική αναζήτηση λειτουργεί σε ταξινομημένους πίνακες. Aρχίζει με τη σύγκριση ενός στοιχείου στο μέσο του πίνακα με την τιμή-στόχο. Εάν η τιμή-στόχος ταιριάζει με το στοιχείο, επιστρέφεται η θέση του στον πίνακα. Εάν η τιμή-στόχος είναι μικρότερη από το στοιχείο, η αναζήτηση συνεχίζεται στο αριστερό μισό του πίνακα. Εάν η τιμή-στόχος είναι μεγαλύτερη από το στοιχείο, η αναζήτηση συνεχίζεται στο δεξιό μισό του πίνακα. Με τον τρόπο αυτό, ο αλγόριθμος εξαλείφει το μισό στο οποίο δεν μπορεί να βρίσκεται η τιμή-στόχος σε κάθε επανάληψη. Αυτή η διαδικασία συνεχίζεται και για το μισό που θα βρίσκεται το στοιχείο που ψάχνουμε.**

***Εύρεση συνολικού όγκου συναλλαγών (πεδίο Volume) στις μετοχές του αρχείου για συγκεκριμένη ημερομηνία (Date) με την βόηθεια της αναζήτησης με παρεμβολή***

* Για το αρχείο *agn.us.txt* ο κώδικας είναι:

*#include <iostream>*

*#include "InputFile.h"*

*#include <fstream>*

*using namespace std;*

*bool found = false;*

*/\** *Εάν το x υπάρχει στο arr[0. . n-1], τότε επιστρέφει το index του αλλιώς επιστρέφει -1\*/*

*int interpolationSearch(struct metoxh arr[], int lo, int hi, long int date) /\* Ορίζουμε την σύναρτηση που θα κάνει την αναζήτηση με παρεμβολή\*/*

*{*

*int pos;*

*if (lo <= hi && date >= arr[lo].Date && date <= arr[hi].Date) {*

*/\* Διερεύνουμε την θέση με γνώμονα την ομοιόμορφη κατανομή.\*/*

*pos = lo + ((((double)(hi - lo) \*(date - arr[lo].Date))/ (arr[hi].Date - arr[lo].Date)));*

*//cout<<"Current Position: "<<pos<<endl;*

*// Ο στόχος βρέθηκε*

*if (arr[pos].Date == date){*

*cout << "H hmeromhnia poy pliktrologisate breuhke sth thesh " << pos;*

*cout<<"\nVolume of this date:"<< arr[pos].Volume;*

*found = true;*

*//cout<<"\n\n\t"<<found<<endl;*

*return pos;*

*}*

*/\** *Εάν το x είναι μεγαλύτερο τότε είναι στο δεξιό sub array\*/*

*else if (arr[pos].Date < date){*

*//cout<<"Right"<<endl;*

*interpolationSearch(arr, pos+1, hi, date);*

*}*

*/\* Εάν το x είναι μικρότερο τότε είναι στο αριστερό sub array\*/*

*else //(arr[pos].Date > date)*

*{*

*//cout<<"Left"<<endl;*

*interpolationSearch(arr, lo, pos-1, date);*

*}*

*}*

*}*

*// Driver Code*

*int main()*

*{*

*/\* Το Array των στοιχείων στο οποίο θα γίνει η αναζήτηση \*/*

*struct metoxh InterPolationArray[AGN\_SIZE];*

*int left = 0;*

*int right = AGN\_SIZE-1;*

*//getDatefromKeyboard();*

*ifstream fin("agn.us.txt");*

*ReadValuesfromFile(fin, InterPolationArray); //διαβάζω τα στοιχεία του αρχείου*

*long int fulldate;*

*fulldate = getDatefromKeyboard(); /\* εισάγω από το πληκτρολόγιο την ημερομηνία που ψάχνουμε\*/*

*cout<<"Full date: "<<fulldate<<endl;*

*interpolationSearch(InterPolationArray, left, right, fulldate); /\*καλούμε την συνάρτηση της αναζήτησης με παρεμβολή\*/*

*if (!found){*

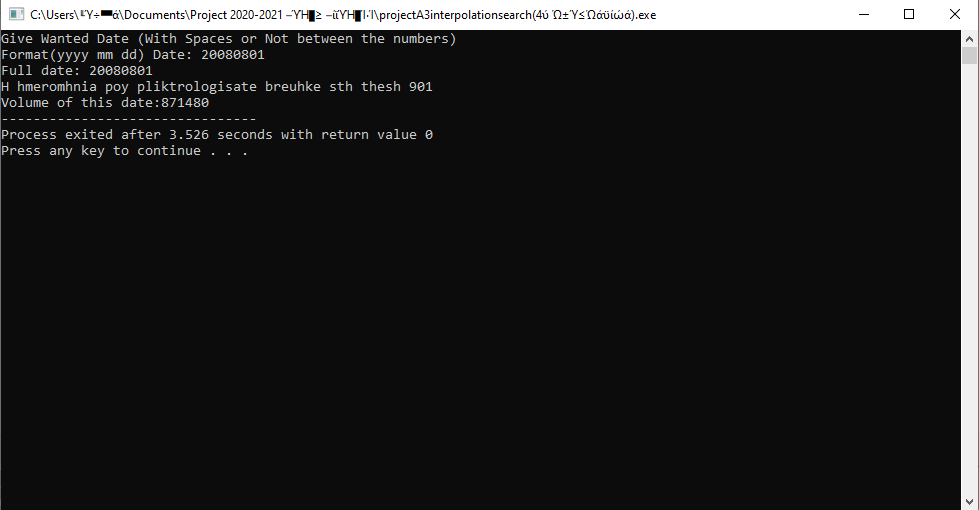
*cout << "\nDate Not Found!!!"<<endl;*

*}*

*return 0;*

*}*

Αν τρέξουμε τον κώδικα θα έχουμε τα εξής αποτελέσματα:



**Η αναζήτηση παρεμβολής είναι ένας αλγόριθμος για την αναζήτηση ενός κλειδιού σε έναν πίνακα που έχει ταξινομηθεί με βάση τις αριθμητικές τιμές που έχουν εκχωρηθεί στα κλειδιά (τιμές κλειδιών). Σε κάθε βήμα ο αλγόριθμος υπολογίζει πού στον εναπομείναντα χώρο αναζήτησης μπορεί να είναι το αναζητούμενο στοιχείο, με βάση τις τιμές των κλειδιών στα όρια του χώρου αναζήτησης και την τιμή του αναζητούμενου κλειδιού, συνήθως μέσω γραμμικής παρεμβολής. Στη συνέχεια, η τιμή κλειδιού που βρέθηκε σε αυτή την εκτιμώμενη θέση συγκρίνεται με την αναζητούμενη τιμή κλειδιού. Εάν δεν είναι ίσο, τότε ανάλογα με τη σύγκριση, ο εναπομένον χώρος αναζήτησης περιορίζεται στο τμήμα πριν ή μετά την εκτιμώμενη θέση.**

Κατά μέσο όρο, η αναζήτηση παρεμβολής κάνει περίπου log(log(n)) συγκρίσεις. Στη χειρότερη περίπτωση μπορεί να κάνει έως και O(n) συγκρίσεις. Ενώ η δυαδική αναζήτηση κατά μέσο όρο κάνει Ο(logn) όπως και στην χειρότερη περίπτωση. Οπότε καταλαβαίνουμε ότι η αναζήτηση με παρεμβολή είναι πιο γρήγορη ,με αποτέλεσμα και πιο αποδοτική.

Η δυαδική αναζήτηση πηγαίνει στο μεσαίο στοιχείο για έλεγχο ανεξάρτητα από το κλειδί αναζήτησης. Από την άλλη πλευρά, η αναζήτηση παρεμβολής μπορεί να μεταβεί σε διαφορετικές θέσεις ανάλογα με το κλειδί αναζήτησης. Εάν η τιμή του κλειδιού αναζήτησης είναι κοντά στο τελευταίο στοιχείο, η αναζήτηση παρεμβολής είναι πιθανό να ξεκινήσει την αναζήτηση προς την πλευρά του τέλους.

**ΕΡΩΤΗΜΑ 4Ο**

***Εύρεση συνολικού όγκου συναλλαγών (πεδίο Volume) στις μετοχές του αρχείου για συγκεκριμένη ημερομηνία (Date) με την βόηθεια της δυαδικής αναζήτησης με παρεμβολή***

* Για το αρχείο *agn.us.txt* ο κώδικας είναι:

*#include <iostream>*

*#include "InputFile.h"*

*#include <math.h>*

*using namespace std;*

*bool found = false;*

*/\** *Εάν το x υπάρχει στο arr[0. . n-1], τότε επιστρέφει το index του αλλιώς επιστρέφει -1\*/*

*int BinaryInterpolationSearch(struct metoxh arr[], int left, int right, long int date) /\* Ορίζουμε την σύναρτηση που θα κάνει την δυαδική αναζήτηση με παρεμβολή\*/*

*{*

*int size = right-left+1;*

*int root = sqrt(sqrt(size));*

*int next = ((size \* ((double)date-arr[left].Date))/(arr[right].Date-arr[left].Date))+1;*

*cout<<"Size: "<<size<<"\nNext: "<<next<<"\nRoot: "<<root<<endl;*

*while(date != arr[next].Date){*

*int i=0;*

*size = right - left +1;*

*//cout<<"New Size: "<<size<<endl;*

*//cout<<"OK\_01"<<endl;*

*// if(size<=3){*

*// //DirectSearch();*

*// }*

*if (date>=arr[next].Date){*

*while(date>arr[next-(i\*root)+1].Date){*

*cout<<"ARR INDEX: "<<next-(i\*root)+1<<"\nOK\_"<<i<<"\nARR: "<<arr[next-(i\*root)+1].Date<<endl;*

*i++;*

*}*

*right = next + (i\* root);*

*cout<<"OK\_Right: "<<right<<endl;*

*left = next +((i-1) \*root);*

*cout<<"OK\_Left: "<<left<<endl;*

*root = sqrt(right-left);*

*}*

*else if(date<arr[next].Date){*

*while(date<arr[next-(i\*root)+1].Date){*

*cout<<"OK\_"<<i<<"\nARR: "<<arr[next-(i\*root)+1].Date<<endl;*

*i++;*

*}*

*left = next - (i\* root);*

*cout<<"OK\_Left: "<<left<<endl;*

*right = next -((i-1) \*root);*

*cout<<"OK\_Right: "<<right<<endl;*

*root = sqrt(right-left);*

*}*

*next = left+((right-left+1)\*(((double)date-arr[left].Date)/(arr[right].Date-arr[left].Date)))-1;*

*cout<<"NEXT= "<<next<<endl;*

*}*

*if(date==arr[next].Date){*

*//Success*

*cout << "H hmeromhnia poy pliktrologisate breuhke sth thesh " << next;*

*cout<<"\nVolume of this date:"<< arr[next].Volume;*

*return next;*

*}*

*else{*

*cout<<"Date not Found"<<endl;*

*}*

*return -1;*

*}*

*// Driver Code*

*int main()*

*{*

*/\* Το Array των στοιχείων στο οποίο θα γίνει η αναζήτηση \*/*

*struct metoxh BinaryInterPolationArray[AGN\_SIZE];*

*int left =0;*

*int right = AGN\_SIZE-1;*

*//getDatefromKeyboard();*

*ifstream fin("agn.us.txt");*

*ReadValuesfromFile(fin, BinaryInterPolationArray);*

*long int fulldate;*

*fulldate = getDatefromKeyboard(); /\*εισάγουμε την ημερομηνία που ψάχνουμε\*/*

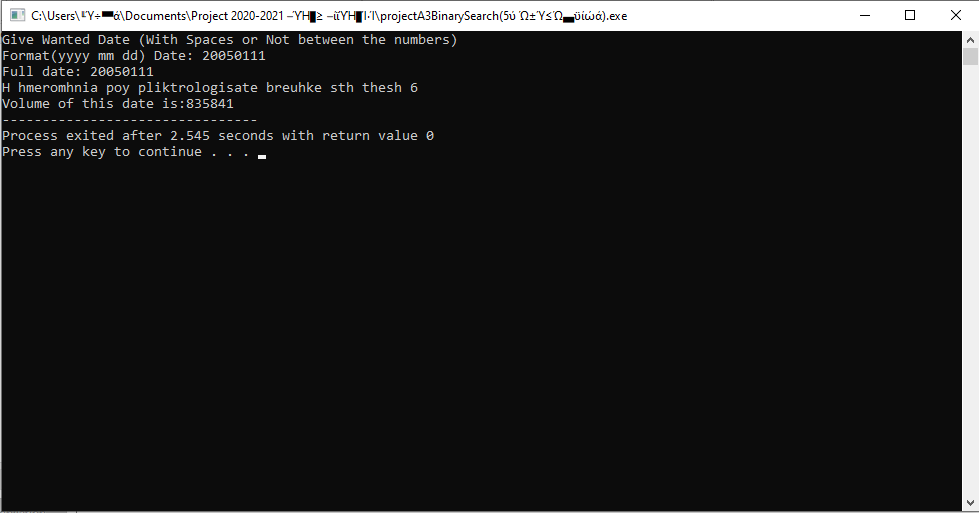
*cout<<"Full date: "<<fulldate<<endl;*

*BinaryInterpolationSearch(BinaryInterPolationArray, left, right, fulldate); /\*καλούμε την συνάρτηση με την οποία θα γίνει η δυαδική αναζήτηση με παρεμβολή\*/*

*return 0;*

*}*

Αν τρέξουμε τον κώδικα θα έχουμε τα εξής αποτελέσματα:



**Τα πειραματικά αποτελέσματα φαίνονται στον παρακάτω πίνακα:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| # Εκτέλεση | 01\_Mergesort | 01\_Quicksort | 02\_Heapsort | 02\_Countingsort | 03\_Binary SearchAvg | 03\_Interpolation SearchAVG | 04\_Binary Interpolation SearchAvg |
| ***1*** | 0,187 | 0,182 | 0,187 | 0,188 | 0,265 | 0,171 | 0,171 |
| ***2*** | 0,156 | 0,168 | 0,156 | 0,169 | 0,281 | 0,172 | 0,157 |
| ***3*** | 0,172 | 0,166 | 0,172 | 0,166 | 0,266 | 0,156 | 0,156 |
| ***4*** | 0,156 | 0,166 | 0,156 | 0,168 | 0,172 | 0,172 | 0,156 |
| ***5*** | 0,157 | 0,167 | 0,172 | 0,163 | 0,156 | 0,157 | 0,156 |
| ***6*** | 0,171 | 0,167 | 0,156 | 0,163 | 0,156 | 0,156 | 0,157 |
| ***7*** | 0,157 | 0,166 | 0,157 | 0,159 | 0,157 | 0,156 | 0,156 |
| ***8*** | 0,156 | 0,167 | 0,172 | 0,159 | 0,156 | 0,156 | 0,156 |
| ***9*** | 0,172 | 0,167 | 0,156 | 0,159 | 0,172 | 0,172 | 0,172 |
| ***10*** | 0,156 | 0,17 | 0,156 | 0,16 | 0,156 | 0,156 | 0,156 |
| ***11*** | 0,156 | 0,166 | 0,172 | 0,16 | 0,156 | 0,157 | 0,156 |
| ***12*** | 0,157 | 0,167 | 0,156 | 0,16 | 0,156 | 0,156 | 0,157 |
| ***13*** | 0,171 | 0,167 | 0,156 | 0,162 | 0,157 | 0,156 | 0,156 |
| ***14*** | 0,157 | 0,167 | 0,157 | 0,161 | 0,156 | 0,156 | 0,156 |
| ***15*** | 0,156 | 0,167 | 0,172 | 0,159 | 0,156 | 0,157 | 0,172 |
| ***16*** | 0,172 | 0,177 | 0,156 | 0,164 | 0,156 | 0,172 | 0,156 |
| ***17*** | 0,156 | 0,167 | 0,156 | 0,173 | 0,157 | 0,156 | 0,157 |
| ***18*** | 0,156 | 0,166 | 0,156 | 0,169 | 0,156 | 0,156 | 0,156 |
| ***19*** | 0,172 | 0,168 | 0,157 | 0,169 | 0,156 | 0,156 | 0,156 |
| ***20*** | 0,156 | 0,166 | 0,171 | 0,16 | 0,156 | 0,157 | 0,156 |
| ***21*** | 0,157 | 0,167 | 0,157 | 0,17 | 0,157 | 0,156 | 0,157 |
| ***22*** | 0,172 | 0,168 | 0,156 | 0,205 | 0,156 | 0,156 | 0,156 |
| ***23*** | 0,156 | 0,169 | 0,156 | 0,163 | 0,156 | 0,156 | 0,156 |
| ***24*** | 0,156 | 0,167 | 0,172 | 0,162 | 0,156 | 0,172 | 0,156 |
| ***25*** | 0,172 | 0,167 | 0,156 | 0,163 | 0,157 | 0,156 | 0,157 |
| ***26*** | 0,156 | 0,166 | 0,157 | 0,161 | 0,156 | 0,157 | 0,171 |
| ***27*** | 0,156 | 0,167 | 0,156 | 0,162 | 0,156 | 0,156 | 0,157 |
| ***28*** | 0,172 | 0,168 | 0,172 | 0,161 | 0,156 | 0,156 | 0,156 |
| ***29*** | 0,157 | 0,153 | 0,156 | 0,162 | 0,172 | 0,156 | 0,156 |
| ***30*** | 0,156 | 0,172 | 0,156 | 0,161 | 0,157 | 0,157 | 0,156 |
| ***31*** | 0,172 | 0,172 | 0,156 | 0,16 | 0,156 | 0,172 | 0,157 |
| ***32*** | 0,156 | 0,156 | 0,157 | 0,161 | 0,156 | 0,156 | 0,156 |
| ***33*** | 0,156 | 0,172 | 0,172 | 0,162 | 0,156 | 0,156 | 0,156 |
| ***34*** | 0,172 | 0,172 | 0,156 | 0,159 | 0,157 | 0,156 | 0,156 |
| ***35*** | 0,156 | 0,156 | 0,156 | 0,163 | 0,156 | 0,157 | 0,157 |
| ***36*** | 0,157 | 0,172 | 0,172 | 0,161 | 0,156 | 0,156 | 0,156 |
| ***37*** | 0,171 | 0,156 | 0,156 | 0,161 | 0,156 | 0,156 | 0,156 |
| ***38*** | 0,157 | 0,172 | 0,156 | 0,16 | 0,157 | 0,156 | 0,156 |
| ***39*** | 0,156 | 0,172 | 0,157 | 0,161 | 0,156 | 0,172 | 0,157 |
| ***40*** | 0,172 | 0,156 | 0,172 | 0,161 | 0,172 | 0,156 | 0,156 |
| ***41*** | 0,156 | 0,172 | 0,161 | 0,161 | 0,156 | 0,157 | 0,156 |
| ***42*** | 0,156 | 0,172 | 0,16 | 0,162 | 0,156 | 0,156 | 0,172 |
| ***43*** | 0,172 | 0,156 | 0,163 | 0,161 | 0,156 | 0,156 | 0,156 |
| ***44*** | 0,156 | 0,172 | 0,16 | 0,16 | 0,157 | 0,156 | 0,157 |
| ***45*** | 0,157 | 0,157 | 0,159 | 0,162 | 0,156 | 0,157 | 0,156 |
| ***46*** | 0,172 | 0,172 | 0,162 | 0,161 | 0,156 | 0,156 | 0,156 |
| ***47*** | 0,156 | 0,172 | 0,161 | 0,164 | 0,156 | 0,172 | 0,156 |
| ***48*** | 0,156 | 0,156 | 0,162 | 0,166 | 0,157 | 0,156 | 0,157 |
| ***49*** | 0,172 | 0,172 | 0,161 | 0,164 | 0,156 | 0,156 | 0,156 |
| ***50*** | 0,156 | 0,156 | 0,16 | 0,163 | 0,156 | 0,157 | 0,156 |
| ***51*** | 0,156 | 0,172 | 0,161 | 0,161 | 0,156 | 0,156 | 0,156 |
| ***52*** | 0,172 | 0,172 | 0,161 | 0,162 | 0,157 | 0,156 | 0,157 |
| ***53*** | 0,157 | 0,156 | 0,161 | 0,161 | 0,156 | 0,156 | 0,156 |
| ***54*** | 0,156 | 0,172 | 0,161 | 0,162 | 0,156 | 0,157 | 0,172 |
| ***55*** | 0,172 | 0,156 | 0,16 | 0,16 | 0,156 | 0,171 | 0,156 |
| ***56*** | 0,156 | 0,172 | 0,149 | 0,161 | 0,157 | 0,157 | 0,156 |
| ***57*** | 0,156 | 0,172 | 0,156 | 0,161 | 0,156 | 0,156 | 0,156 |
| ***58*** | 0,172 | 0,156 | 0,172 | 0,161 | 0,156 | 0,156 | 0,157 |
| ***59*** | 0,156 | 0,172 | 0,157 | 0,16 | 0,156 | 0,156 | 0,156 |
| ***60*** | 0,157 | 0,172 | 0,156 | 0,161 | 0,157 | 0,157 | 0,156 |
| ***61*** | 0,171 | 0,156 | 0,172 | 0,162 | 0,172 | 0,156 | 0,156 |
| ***62*** | 0,157 | 0,172 | 0,156 | 0,161 | 0,156 | 0,156 | 0,172 |
| ***63*** | 0,156 | 0,172 | 0,156 | 0,161 | 0,156 | 0,172 | 0,157 |
| ***64*** | 0,172 | 0,156 | 0,156 | 0,161 | 0,156 | 0,156 | 0,156 |
| ***65*** | 0,156 | 0,172 | 0,172 | 0,16 | 0,157 | 0,157 | 0,156 |
| ***66*** | 0,156 | 0,156 | 0,157 | 0,162 | 0,156 | 0,156 | 0,156 |
| ***67*** | 0,172 | 0,172 | 0,156 | 0,16 | 0,156 | 0,156 | 0,157 |
| ***68*** | 0,156 | 0,172 | 0,156 | 0,162 | 0,156 | 0,156 | 0,156 |
| ***69*** | 0,157 | 0,156 | 0,172 | 0,161 | 0,157 | 0,157 | 0,156 |
| ***70*** | 0,172 | 0,172 | 0,156 | 0,162 | 0,156 | 0,156 | 0,156 |
| ***71*** | 0,156 | 0,156 | 0,156 | 0,16 | 0,156 | 0,172 | 0,157 |
| ***72*** | 0,156 | 0,172 | 0,157 | 0,162 | 0,156 | 0,156 | 0,171 |
| ***73*** | 0,172 | 0,172 | 0,156 | 0,162 | 0,157 | 0,156 | 0,157 |
| ***74*** | 0,156 | 0,156 | 0,172 | 0,16 | 0,156 | 0,156 | 0,156 |
| ***75*** | 0,156 | 0,172 | 0,156 | 0,161 | 0,156 | 0,157 | 0,156 |
| ***76*** | 0,172 | 0,157 | 0,156 | 0,161 | 0,156 | 0,156 | 0,156 |
| ***77*** | 0,148 | 0,171 | 0,157 | 0,161 | 0,157 | 0,156 | 0,157 |
| ***78*** | 0,172 | 0,172 | 0,171 | 0,162 | 0,156 | 0,156 | 0,156 |
| ***79*** | 0,156 | 0,157 | 0,157 | 0,159 | 0,156 | 0,157 | 0,156 |
| ***80*** | 0,172 | 0,171 | 0,156 | 0,161 | 0,156 | 0,156 | 0,156 |
| ***81*** | 0,156 | 0,172 | 0,156 | 0,161 | 0,157 | 0,156 | 0,157 |
| ***82*** | 0,172 | 0,172 | 0,172 | 0,161 | 0,156 | 0,172 | 0,156 |
| ***83*** | 0,156 | 0,156 | 0,156 | 0,162 | 0,156 | 0,156 | 0,156 |
| ***84*** | 0,172 | 0,172 | 0,157 | 0,161 | 0,172 | 0,157 | 0,156 |
| ***85*** | 0,156 | 0,157 | 0,156 | 0,161 | 0,156 | 0,156 | 0,157 |
| ***86*** | 0,172 | 0,171 | 0,172 | 0,161 | 0,156 | 0,156 | 0,172 |
| ***87*** | 0,165 | 0,172 | 0,156 | 0,162 | 0,157 | 0,156 | 0,156 |
| ***88*** | 0,156 | 0,157 | 0,156 | 0,16 | 0,156 | 0,157 | 0,156 |
| ***89*** | 0,172 | 0,171 | 0,156 | 0,161 | 0,156 | 0,171 | 0,156 |
| ***90*** | 0,156 | 0,157 | 0,172 | 0,161 | 0,156 | 0,157 | 0,157 |
| ***91*** | 0,157 | 0,172 | 0,157 | 0,162 | 0,157 | 0,156 | 0,156 |
| ***92*** | 0,172 | 0,171 | 0,156 | 0,16 | 0,156 | 0,156 | 0,156 |
| ***93*** | 0,156 | 0,157 | 0,156 | 0,159 | 0,156 | 0,156 | 0,156 |
| ***94*** | 0,172 | 0,172 | 0,156 | 0,162 | 0,156 | 0,157 | 0,157 |
| ***95*** | 0,156 | 0,156 | 0,172 | 0,165 | 0,157 | 0,156 | 0,156 |
| ***96*** | 0,172 | 0,172 | 0,156 | 0,165 | 0,172 | 0,156 | 0,156 |
| ***97*** | 0,156 | 0,172 | 0,157 | 0,168 | 0,156 | 0,156 | 0,156 |
| ***98*** | 0,156 | 0,156 | 0,156 | 0,166 | 0,156 | 0,172 | 0,157 |
| ***99*** | 0,172 | 0,172 | 0,172 | 0,16 | 0,156 | 0,157 | 0,156 |
| ***100*** | 0,156 | 0,172 | 0,156 | 0,162 | 0,157 | 0,156 | 0,156 |

**Τα πειραματικά αποτελέσματα της δυαδικής αναζήτησης και της αναζήτησης με παρεμβολή στην καλύτερη, μέση και χειρότερη περίπτωση φαίνονται παρακάτω:**

* Για την δυαδική αναζήτηση

|  |  |  |
| --- | --- | --- |
| 03\_Binary SearchWorst | 03\_Binary SearchAvg | 03\_Binary SearchBest |
| 0,265 | 0,265 | 0,156 |
| 0,266 | 0,281 | 0,156 |
| 0,218 | 0,266 | 0,156 |
| 0,157 | 0,172 | 0,172 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,156 | 0,157 | 0,156 |
| 0,172 | 0,156 | 0,156 |
| 0,157 | 0,172 | 0,172 |
| 0,156 | 0,156 | 0,156 |
| 0,172 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,156 | 0,157 | 0,156 |
| 0,156 | 0,156 | 0,172 |
| 0,157 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,157 | 0,172 |
| 0,156 | 0,156 | 0,156 |
| 0,157 | 0,156 | 0,204 |
| 0,156 | 0,156 | 0,187 |
| 0,156 | 0,157 | 0,172 |
| 0,156 | 0,156 | 0,156 |
| 0,157 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,172 |
| 0,172 | 0,157 | 0,157 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,157 | 0,156 | 0,172 |
| 0,156 | 0,172 | 0,156 |
| 0,156 | 0,157 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,157 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,157 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,157 | 0,156 | 0,172 |
| 0,171 | 0,156 | 0,218 |
| 0,157 | 0,157 | 0,157 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,172 | 0,156 |
| 0,156 | 0,156 | 0,172 |
| 0,157 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,157 | 0,157 |
| 0,156 | 0,156 | 0,156 |
| 0,157 | 0,156 | 0,156 |
| 0,172 | 0,156 | 0,156 |
| 0,156 | 0,157 | 0,157 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,157 | 0,156 | 0,156 |
| 0,156 | 0,157 | 0,157 |
| 0,172 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,172 |
| 0,156 | 0,157 | 0,156 |
| 0,157 | 0,156 | 0,157 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,157 | 0,156 |
| 0,157 | 0,172 | 0,172 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,156 | 0,156 | 0,156 |
| 0,172 | 0,157 | 0,156 |
| 0,157 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,172 |
| 0,156 | 0,156 | 0,157 |
| 0,156 | 0,157 | 0,156 |
| 0,157 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,156 | 0,157 | 0,156 |
| 0,157 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,172 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,157 | 0,156 |
| 0,157 | 0,156 | 0,157 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,172 | 0,157 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,156 | 0,156 | 0,156 |
| 0,157 | 0,172 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,156 | 0,157 | 0,156 |
| 0,157 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,156 | 0,157 | 0,156 |
| 0,157 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,172 | 0,157 | 0,156 |
| 0,156 | 0,172 | 0,156 |
| 0,157 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,157 | 0,156 |

* Για την αναζήτηση με παρεμβολή

|  |  |  |
| --- | --- | --- |
| 03\_Interpolation SearchWorst | 03\_Interpolation SearchAVG | 03\_Interpolation SearchBest |
| 0,265 | 0,171 | 0,265 |
| 0,234 | 0,172 | 0,25 |
| 0,266 | 0,156 | 0,203 |
| 0,234 | 0,172 | 0,156 |
| 0,157 | 0,157 | 0,157 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,172 |
| 0,156 | 0,156 | 0,156 |
| 0,157 | 0,172 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,172 | 0,157 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,157 | 0,156 | 0,172 |
| 0,156 | 0,157 | 0,156 |
| 0,156 | 0,172 | 0,157 |
| 0,156 | 0,156 | 0,156 |
| 0,157 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,172 | 0,157 | 0,157 |
| 0,156 | 0,156 | 0,172 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,157 | 0,172 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,156 | 0,157 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,157 | 0,156 | 0,172 |
| 0,172 | 0,156 | 0,156 |
| 0,156 | 0,157 | 0,156 |
| 0,156 | 0,172 | 0,157 |
| 0,156 | 0,156 | 0,156 |
| 0,157 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,172 | 0,157 | 0,172 |
| 0,156 | 0,156 | 0,157 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,157 | 0,172 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,156 | 0,157 | 0,156 |
| 0,156 | 0,156 | 0,172 |
| 0,172 | 0,156 | 0,156 |
| 0,157 | 0,156 | 0,156 |
| 0,156 | 0,157 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,156 | 0,172 | 0,156 |
| 0,157 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,172 |
| 0,172 | 0,157 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,157 | 0,157 | 0,156 |
| 0,156 | 0,171 | 0,157 |
| 0,156 | 0,157 | 0,156 |
| 0,156 | 0,156 | 0,172 |
| 0,157 | 0,156 | 0,156 |
| 0,172 | 0,156 | 0,156 |
| 0,156 | 0,157 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,156 | 0,156 | 0,156 |
| 0,157 | 0,172 | 0,172 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,157 | 0,156 |
| 0,172 | 0,156 | 0,157 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,157 | 0,157 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,156 | 0,172 | 0,171 |
| 0,156 | 0,156 | 0,157 |
| 0,172 | 0,156 | 0,156 |
| 0,157 | 0,156 | 0,156 |
| 0,156 | 0,157 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,156 | 0,156 | 0,156 |
| 0,157 | 0,156 | 0,156 |
| 0,156 | 0,157 | 0,172 |
| 0,172 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,156 | 0,172 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,157 | 0,157 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,156 | 0,156 |
| 0,172 | 0,157 | 0,172 |
| 0,157 | 0,171 | 0,156 |
| 0,156 | 0,157 | 0,156 |
| 0,156 | 0,156 | 0,157 |
| 0,156 | 0,156 | 0,156 |
| 0,157 | 0,156 | 0,156 |
| 0,156 | 0,157 | 0,156 |
| 0,156 | 0,156 | 0,172 |
| 0,172 | 0,156 | 0,157 |
| 0,156 | 0,156 | 0,156 |
| 0,156 | 0,172 | 0,156 |
| 0,157 | 0,157 | 0,156 |
| 0,156 | 0,156 | 0,157 |

PART II: “BSTs && HASHING”

Ο κώδικας του δεύτερου μέρους είναι:

A)

*#include <iostream>*

*#include <queue>*

*#include <unordered\_map>*

*#include"InputFile.h"*

*using namespace std;*

*struct node {*

*struct node \*left;*

*struct metoxh data;*

*int height;*

*struct node \*right;*

*};*

*int BinarySearchArr(struct metoxh array[], int first, int middle, int last, long int WantedDate){*

*first = 0;*

*last = AGN\_SIZE - 1;*

*middle = (first+last)/2;*

*bool found = false;*

*while (first <= last) {*

*if(array[middle].Date < WantedDate)*

*{*

*first = middle + 1;*

*middle = (first + last)/2;*

*//cout<<"Right"<<endl;*

*}*

*else if (array[middle].Date == WantedDate)*

*{*

*cout<< "\nVolume of this date is:" << array[middle].Volume<<endl;*

*found = true;*

*return middle;*

*}*

*else*

*{*

*last = middle - 1;*

*middle = (first + last)/2;*

*//cout<<"Left"<<endl;*

*}*

*}*

*if (!found){*

*cout << "Date Not Found!!!"<<endl;*

*}*

*return -1;*

*}*

*class AVL //με την κλάση AVL φτιάχνουμε το δυαδικό δέντρο AVL*

*{*

*public:*

*struct node \* root;*

*AVL(){*

*this->root = NULL;*

*}*

*int calheight(struct node \*p){ /\*η συνάρτηση αυτή βρίσκει το ύψος εωός κόμβου p\*/*

*if(p->left && p->right){*

*if (p->left->height < p->right->height)*

*return p->right->height + 1;*

*else return p->left->height + 1;*

*}*

*else if(p->left && p->right == NULL){*

*return p->left->height + 1;*

*}*

*else if(p->left ==NULL && p->right){*

*return p->right->height + 1;*

*}*

*return 0;*

*}*

*int bf(struct node \*n){ /\* η συνάρτηση αυτή βγάζει ως αποτέλεσμα το αποτέλεσμα της αφαίρεσης του ύψους του δεξιού υποδέντρου από του αριστερού υποδέντρου του κόμβου n \*/*

*if(n->left && n->right){*

*return n->left->height - n->right->height;*

*}*

*else if(n->left && n->right == NULL){*

*return n->left->height;*

*}*

*else if(n->left== NULL && n->right ){*

*return -n->right->height;*

*}*

*}*

*struct node \* llrotation(struct node \*n){ /\*η σύναρτηση αυτή κάνει δεξία περιστροφή στον κόμβο n \*/*

*struct node \*p;*

*struct node \*tp;*

*p = n;*

*tp = p->left;*

*p->left = tp->right;*

*tp->right = p;*

*return tp;*

*}*

*struct node \* rrrotation(struct node \*n){ /\*η σύναρτηση αυτή κάνει αριστερή περιστροφή στον κόμβο n \*/*

*struct node \*p;*

*struct node \*tp;*

*p = n;*

*tp = p->right;*

*p->right = tp->left;*

*tp->left = p;*

*return tp;*

*}*

*struct node \* rlrotation(struct node \*n){ /\* η συνάρτηση αυτή αρχικά κάνει δεξία περιστροφή στον κόμβο tp και μετά μια αριστερή περιστροφή στον κόμβο n\*/*

*struct node \*p;*

*struct node \*tp;*

*struct node \*tp2;*

*p = n;*

*tp = p->right;*

*tp2 =p->right->left;*

*p -> right = tp2->left;*

*tp ->left = tp2->right;*

*tp2 ->left = p;*

*tp2->right = tp;*

*return tp2;*

*}*

*struct node \* lrrotation(struct node \*n){ /\* η συνάρτηση αυτή αρχικά κάνει αριστερή περιστροφή στον κόμβο tp και μετά μια δεξιά περιστροφή στον κόμβο n\*/*

*struct node \*p;*

*struct node \*tp;*

*struct node \*tp2;*

*p = n;*

*tp = p->left;*

*tp2 =p->left->right;*

*p -> left = tp2->right;*

*tp ->right = tp2->left;*

*tp2 ->right = p;*

*tp2->left = tp;*

*return tp2;*

*}*

*struct node\* insert(struct node \*r,struct metoxh data){ /\*αυτή η συνάρτηση εισάγει ένα νέο κόμβο ως εξής: συγκρίνει αυτού με την τιμή της ρίζας, κια εαν είναι μεγαλύτερη κατευθείνεται στο δεξιό υποδέντρο της ρίζας ενώ αν είναι μικρότερη κατευθείνεται στο αριστερό υποδέντρο της . τέλος επαναλαμβάνει την ίδια διαδικασία για κάθε κόμβο που θα συναντήσει μέχρι να βρει ο κόμβος την σωστή θέση να τοποθετηθεί\*/*

*if(r==NULL){*

*struct node \*n;*

*n = new struct node;*

*StructAssignTo(data,n->data);//n->data = data;*

*r = n;*

*r->left = r->right = NULL;*

*r->height = 1;*

*return r;*

*}*

*else{*

*if(data.Date < r->data.Date)*

*r->left = insert(r->left,data);*

*else*

*r->right = insert(r->right,data);*

*}*

*r->height = calheight(r);*

*if(bf(r)==2 && bf(r->left)==1){*

*r = llrotation(r);*

*}*

*else if(bf(r)==-2 && bf(r->right)==-1){*

*r = rrrotation(r);*

*}*

*else if(bf(r)==-2 && bf(r->right)==1){*

*r = rlrotation(r);*

*}*

*else if(bf(r)==2 && bf(r->left)==-1){*

*r = lrrotation(r);*

*}*

*return r;*

*}*

*void printInorder(struct node\* v) /\* κάνει ενδοδιατεταγμένη διαπέραση στο δέντρο AVL\*/*

*{*

*/\*--------------------------------------------------------------------------------------\*/*

*queue <struct node \*> q;*

*struct node \*cur;*

*q.push(v);*

*q.push(NULL);*

*while(!q.empty()){*

*cur = q.front();*

*q.pop();*

*if(cur == NULL && q.size()!=0){*

*cout<<"\n";*

*q.push(NULL);*

*continue;*

*}*

*if(cur!=NULL){*

*cout<<"|Date: "<<cur->data.Date<<", Volume: "<<cur->data.Volume<<"|";*

*if (cur->left!=NULL){*

*q.push(cur->left);*

*}*

*if (cur->right!=NULL){*

*q.push(cur->right);*

*}*

*}*

*}*

*}*

*void printInorder()*

*{*

*if (this->root == NULL)*

*{*

*cout<<"\n"<<"Empty tree"<<"\n";*

*return;*

*}*

*printInorder(this->root->left);*

*cout <<"|Date: "<<this->root->data.Date << ", Volume: "<<this->root->data.Volume<<"|";*

*printInorder(this->root->right);*

*}*

*struct node \* deleteNode(struct node \*p,struct metoxh data){ /\* η συνάρτηση αυτή διαγράφει τον κόμβο p ως εξής: αν ο κόμβος p δεν έχει παιδιά τότε διαγράφεται, αν ο κόμβος p έχει ένα παιδί τότε τον διαγράφουμε και τον αντικαθιστούμε με το παιδί του και εξετάζουμε για ανισσοροπία τους προγόνους του νέου κόμβου , αν ο κόμβος p έχει δυο παιδιά και το δεξί παιδί του δεν έχει αριστερό παιδί τότε διαγράφεται ο κόμβος p και αντικαθίσταται με το δεξί παιδί του και αν ο κόμβος p έχει δύο παιδιά και το δεξί παιδί του έχει αριστερό παιδί τότε αντικαθίσταται ο κόμβος p με τον επόμενο του στην ενδοδιατεταγμένη διαδρομή και εξετάζονται όλοι οι πρόγονοι του νέου κόμβου για ανισσοροπία \*/*

*if(p->left == NULL && p->right == NULL){*

*if(p==this->root)*

*this->root = NULL;*

*delete p;*

*return NULL;*

*}*

*struct node \*t;*

*struct node \*q;*

*if(p->data.Date < data.Date){*

*p->right = deleteNode(p->right,data);*

*}*

*else if(p->data.Date > data.Date){*

*p->left = deleteNode(p->left,data);*

*}*

*else{*

*if(p->left != NULL){*

*q = inpre(p->left);*

*StructAssignTo(q->data,p->data);//p->data = q->data;*

*p->left=deleteNode(p->left,q->data);*

*}*

*else{*

*q = insuc(p->right);*

*StructAssignTo(q->data,p->data);// p->data= q->data;*

*p->right = deleteNode(p->right,q->data);*

*}*

*}*

*if(bf(p)==2 && bf(p->left)==1){ p = llrotation(p); }*

*else if(bf(p)==2 && bf(p->left)==-1){ p = lrrotation(p); }*

*else if(bf(p)==2 && bf(p->left)==0){ p = llrotation(p); }*

*else if(bf(p)==-2 && bf(p->right)==-1){ p = rrrotation(p); }*

*else if(bf(p)==-2 && bf(p->right)==1){ p = rlrotation(p); }*

*else if(bf(p)==-2 && bf(p->right)==0){ p = llrotation(p); }*

*return p;*

*}*

*struct node\* inpre(struct node\* p){ /\**

*while(p->right!=NULL)*

*p = p->right;*

*return p;*

*}*

*struct node\* insuc(struct node\* p){*

*while(p->left!=NULL)*

*p = p->left;*

*return p;*

*}*

*struct node\* BinarySearch(struct node\* n, int WantedDate){ /\* θα έκανε δυαδική αναζήτηση στο δέντρο αλλά δεν τρέχει\*/*

*if (n->data.Date==WantedDate){*

*cout<<"Found, Volume is: "<<n->data.Volume<<endl;*

*return n;*

*}*

*else if(n->data.Date>WantedDate){*

*n=n->right;*

*// cout<<"Not Found, Volume is: "<<n->data.Volume;*

*BinarySearch(n, WantedDate);*

*}*

*else{*

*n=n->left;*

*// cout<<"Not Found, Volume is: "<<n->data.Volume;*

*BinarySearch(n, WantedDate);*

*}*

*return NULL;*

*}*

*~AVL(){*

*}*

*};*

*int main(){*

*AVL avltree;*

*int choice, x;*

*ifstream infile("agn.us.txt");*

*struct metoxh Stocks[AGN\_SIZE];*

*int first = 0;*

*int last = AGN\_SIZE-1;*

*int middle = (first + last)/2;*

*ReadValuesfromFile(infile, Stocks);*

*for (int i = 0; i<AGN\_SIZE; i++){*

*avltree.root=avltree.insert(avltree.root,Stocks[i]);*

*// cout<<avltree.root->data.Date<<endl;*

*}*

*do{*

*cout<<"AVL Tree Display with In Order Traversal\t(1)\nSearch by Date and Print Volume\t\t\t(2)\nSearch by Date and Set new Volume\t\t(3)\nSearch by Date and Delete Record\t\t(4)\n\nExit\t\t\t\t\t\t(0)"<<endl;*

*cin>>choice;*

*system("cls");*

*switch(choice){*

*case 1: /\*κάνει ενδοδιαδεταγμένη διαπέραση στο δέντρο AVL\*/*

*avltree.printInorder();*

*system("PAUSE");*

*system("cls");*

*break;*

*case 2: /\* εισάγει ο χρήστης μια ημερομηνία από την συνάρτηση getDatefromKeyboard() και την αναζητά με δυαδική αναζήτηση σε πίνακα και εξάγει το Volume της ημερομηνίας\*/*

*long int find;*

*find = getDatefromKeyboard();*

*BinarySearchArr(Stocks , first, middle, last, find);*

*system("PAUSE");*

*system("cls");*

*break;*

*case 3: /\* εισάγει ο χρήστης μια ημερομηνία από την συνάρτηση getDatefromKeyboard() και την αναζητά με δυαδική αναζήτηση σε πίνακα, εξάγει το Volume της ημερομηνίας και πληκτρολογεί ο χρήστης νέο Volume\*/*

*long int find2;*

*int t;*

*find2 = getDatefromKeyboard();*

*t = BinarySearchArr(Stocks , first, middle, last, find2);*

*if(t == -1){*

*system("cls");*

*break;*

*}*

*cout<<"Give new Volume: ";*

*int newvol;*

*cin>>newvol;*

*Stocks[t].Volume = newvol;*

*for (int i = 0; i<AGN\_SIZE; i++){*

*avltree.root=avltree.insert(avltree.root,Stocks[i]);*

*// cout<<avltree.root->data.Date<<endl;*

*//cout<<"Tree is Rebuilding"<<endl;*

*//system("cls");*

*}*

*cout<<"Tree Rebuilt"<<endl;*

*system("PAUSE");*

*system("cls");*

*break;*

*case 4: /\* εισάγει ο χρήστης μια ημερομηνία από την συνάρτηση getDatefromKeyboard(), την αναζητά με δυαδική αναζήτηση σε πίνακα και σβήνει την εγγραφή αυτή\*/*

*long int find3;*

*int j;*

*find3 = getDatefromKeyboard();*

*j = BinarySearchArr(Stocks , first, middle, last, find3);*

*if(j == -1){*

*system("cls");*

*break;*

*}*

*Stocks[j].Close = -1;*

*Stocks[j].Date = -1;*

*Stocks[j].High = -1;*

*Stocks[j].Low = -1;*

*Stocks[j].Open = -1;*

*Stocks[j].OpenInt = -1;*

*Stocks[j].Volume = -1;*

*for (int i = 0; i<AGN\_SIZE; i++){*

*if(Stocks[i].Date!=-1){*

*avltree.root=avltree.insert(avltree.root,Stocks[i]);*

*// cout<<avltree.root->data.Date<<endl;*

*//cout<<"Tree is Rebuilding"<<endl;*

*}*

*//system("cls");*

*}*

*cout<<"Tree Rebuilt"<<endl;*

*system("PAUSE");*

*system("cls");*

*break;*

*default:*

*system("cls");*

*break;*

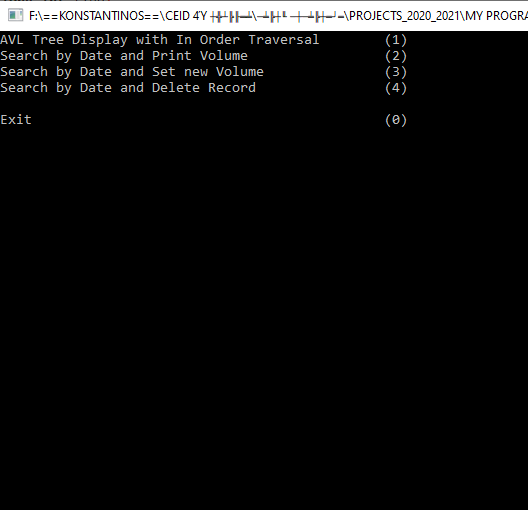
*}*

*}while(choice!=0);*

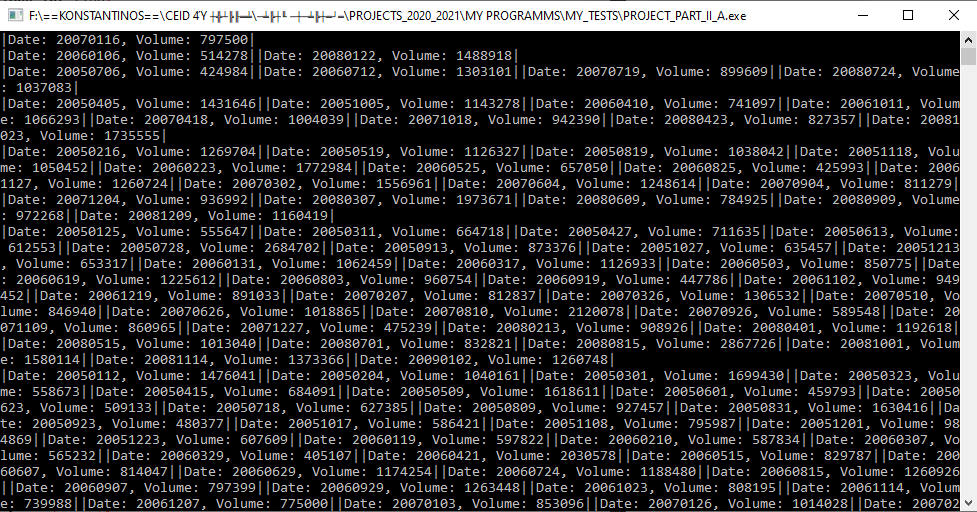
*return 0;*

*}*

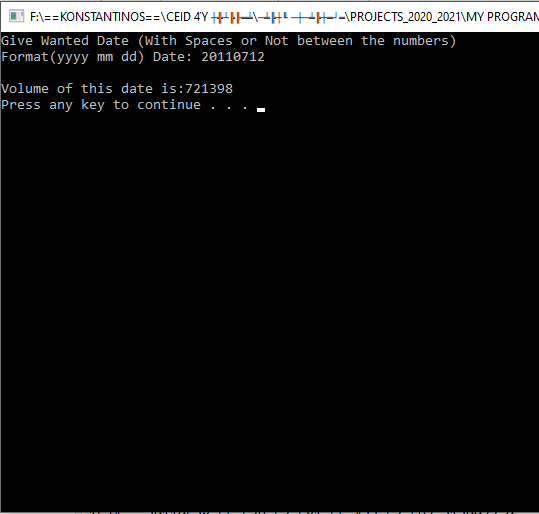
Τα print screen των αποτελεσμάτων είναι :



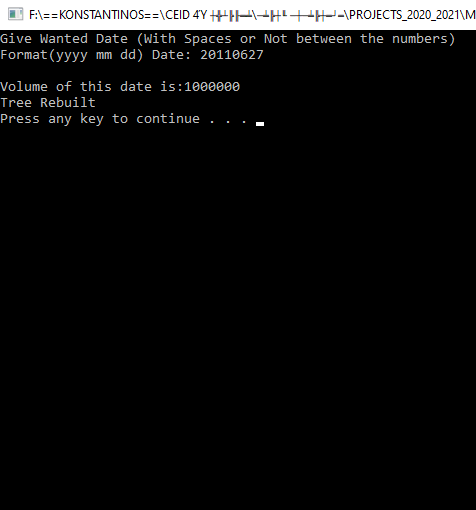
Αν επιλέξει τον αριθμό 1:



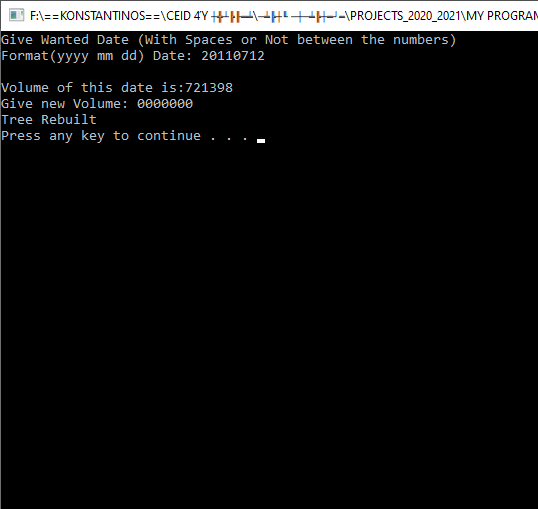
Αν επιλέξει τον αριθμό 2 :



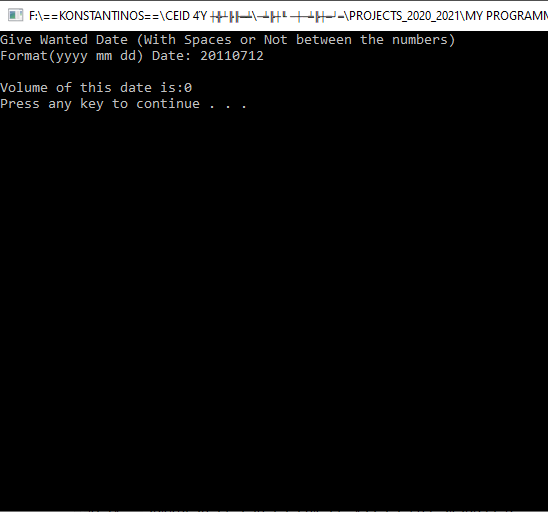
Αν πληκτρολογήσει ο χρήστης τον αριθμό 3 τότε του εμφανίζει το εξής :



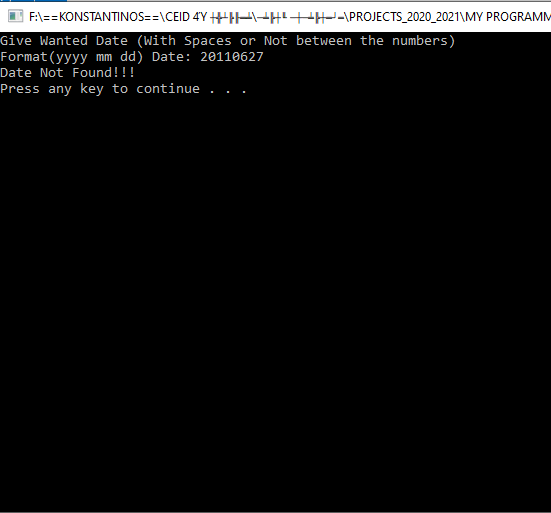
Αν επιλέξει το 4:



Όταν ο χρήστης πληκτρολογεί τον αριθμό 3 και εμείς βάλουμε την ημερομηνία 20110712, βγάζει το Volume της και μας αφήνει να εισάγουμε νέο volume ,αντικαθιστώντας το παλιό του, το οποίο εμείς βάλαμε 0. Στην συνέχεια όταν ο χρήστης θα πληκτρολογήσει την τιμή 2 και εισάγει την ίδια ημερομηνία το volume που θα εμφανιστεί θα είναι όντως το 0.



Επίσης όταν ο χρήστης πληκτρολογήσει τον αριθμό 4 και εισάγει την ημερομηνία 20110627 τότε θα διαγραφεί η εγγραφή αυτή και στην συνέχεια αν πληκτρολογησει τον αριθμό 2 θα εμφανιστεί το παρακάτω μήνυμα.



Β )

Ο κώδικας είναι

*#include <iostream>*

*#include <queue>*

*#include <unordered\_map>*

*#include"InputFile.h"*

*using namespace std;*

*struct node {*

*struct node \*left;*

*struct metoxh data;*

*int height;*

*struct node \*right;*

*};*

*int BinarySearchArr(struct metoxh array[], int first, int middle, int last, long int WVolume){*

*first = 0;*

*last = AGN\_SIZE - 1;*

*middle = (first+last)/2;*

*bool found = false;*

*while (first <= last) {*

*if(array[middle].Volume < WVolume)*

*{*

*first = middle + 1;*

*middle = (first + last)/2;*

*//cout<<"Right"<<endl;*

*}*

*else if (array[middle].Volume == WVolume)*

*{*

*cout<< "\nVolume of this date is:" << array[middle].Volume<<endl;*

*found = true;*

*return middle;*

*}*

*else*

*{*

*last = middle - 1;*

*middle = (first + last)/2;*

*//cout<<"Left"<<endl;*

*}*

*}*

*if (!found){*

*cout << "Date Not Found!!!"<<endl;*

*}*

*return -1;*

*}*

*void mergeVolume(struct metoxh arr[], int left, int mid, int right) /\* Κάνει merge sort στον πίνακα με τιμές τις Volume για να τις έχουμε σωστά κατανεμημένες \*/*

*{*

*int i, j, k;*

*int n1 = mid - left + 1;*

*int n2 = right - mid;*

*/\* create temp arrays \*/*

*struct metoxh L[n1], R[n2];*

*/\* Copy data to temp arrays L[] and R[], xorizo sti mesi ton pinaka arr \*/*

*//while(g=1){*

*for (i = 0; i < n1; i++)*

*StructAssignTo(arr[left+i], L[i]); //L[i] = arr[l + i];*

*for (j = 0; j < n2; j++)*

*StructAssignTo(arr[mid + 1 + j],R[j]);//R[j] = arr[m + 1 + j];*

*//}*

*/\* Merge the temp arrays back into arr[l..r]\* , ftiaxno tous neous pinakes L kai R \*/*

*i = 0; // Initial index of first subarray*

*j = 0; // Initial index of second subarray*

*k = left; // Initial index of merged subarray*

*while (i < n1 && j < n2) {*

*if (L[i].Volume <= R[j].Volume) {*

*StructAssignTo( L[i], arr[k] ); //arr[k] = L[i]*

*i++;*

*}*

*else {*

*StructAssignTo( R[j],arr[k] );//arr[k] = R[j];*

*j++;*

*}*

*k++;*

*}*

*/\* Copy the remaining elements of L[], if there*

*are any \*/*

*while (i < n1) {*

*StructAssignTo( L[i],arr[k] );//arr[k] = L[i];*

*i++;*

*k++;*

*}*

*/\* Copy the remaining elements of R[], if there*

*are any \*/*

*while (j < n2) {*

*StructAssignTo( R[j],arr[k] );//arr[k] = R[j];*

*j++;*

*k++;*

*}*

*}*

*/\* l is for left index and r is right index of the*

*sub-array of arr to be sorted \*/*

*void mergeSortVolume(struct metoxh met[], int left, int right)*

*{*

*if (left >= right) {*

*return;*

*}*

*// Same as (l+r)/2, but avoids overflow for*

*// large l and h*

*int mid = left + (right - left) / 2;*

*// Sort first and second halves*

*mergeSortVolume(met, left, mid);*

*mergeSortVolume(met, mid + 1, right);*

*mergeVolume(met, left, mid, right);*

*//cout<<"Done"<<endl; //Debug Help*

*}*

*class AVL*

*{*

*public:*

*struct node \* root;*

*AVL(){*

*this->root = NULL;*

*}*

*int calheight(struct node \*p){*

*if(p->left && p->right){*

*if (p->left->height < p->right->height)*

*return p->right->height + 1;*

*else return p->left->height + 1;*

*}*

*else if(p->left && p->right == NULL){*

*return p->left->height + 1;*

*}*

*else if(p->left ==NULL && p->right){*

*return p->right->height + 1;*

*}*

*return 0;*

*}*

*int bf(struct node \*n){*

*if(n->left && n->right){*

*return n->left->height - n->right->height;*

*}*

*else if(n->left && n->right == NULL){*

*return n->left->height;*

*}*

*else if(n->left== NULL && n->right ){*

*return -n->right->height;*

*}*

*}*

*struct node \* llrotation(struct node \*n){*

*struct node \*p;*

*struct node \*tp;*

*p = n;*

*tp = p->left;*

*p->left = tp->right;*

*tp->right = p;*

*return tp;*

*}*

*struct node \* rrrotation(struct node \*n){*

*struct node \*p;*

*struct node \*tp;*

*p = n;*

*tp = p->right;*

*p->right = tp->left;*

*tp->left = p;*

*return tp;*

*}*

*struct node \* rlrotation(struct node \*n){*

*struct node \*p;*

*struct node \*tp;*

*struct node \*tp2;*

*p = n;*

*tp = p->right;*

*tp2 =p->right->left;*

*p -> right = tp2->left;*

*tp ->left = tp2->right;*

*tp2 ->left = p;*

*tp2->right = tp;*

*return tp2;*

*}*

*struct node \* lrrotation(struct node \*n){*

*struct node \*p;*

*struct node \*tp;*

*struct node \*tp2;*

*p = n;*

*tp = p->left;*

*tp2 =p->left->right;*

*p -> left = tp2->right;*

*tp ->right = tp2->left;*

*tp2 ->right = p;*

*tp2->left = tp;*

*return tp2;*

*}*

*struct node\* insert(struct node \*r,struct metoxh data){*

*if(r==NULL){*

*struct node \*n;*

*n = new struct node;*

*StructAssignTo(data,n->data);//n->data = data;*

*r = n;*

*r->left = r->right = NULL;*

*r->height = 1;*

*return r;*

*}*

*else{*

*if(data.Volume < r->data.Volume)*

*r->left = insert(r->left,data);*

*else*

*r->right = insert(r->right,data);*

*}*

*r->height = calheight(r);*

*if(bf(r)==2 && bf(r->left)==1){*

*r = llrotation(r);*

*}*

*else if(bf(r)==-2 && bf(r->right)==-1){*

*r = rrrotation(r);*

*}*

*else if(bf(r)==-2 && bf(r->right)==1){*

*r = rlrotation(r);*

*}*

*else if(bf(r)==2 && bf(r->left)==-1){*

*r = lrrotation(r);*

*}*

*return r;*

*}*

*void printInorder(struct node\* v)*

*{*

*/\*--------------------------------------------------------------------------------------\*/*

*queue <struct node \*> q;*

*struct node \*cur;*

*q.push(v);*

*q.push(NULL);*

*while(!q.empty()){*

*cur = q.front();*

*q.pop();*

*if(cur == NULL && q.size()!=0){*

*cout << "\n";*

*q.push(NULL);*

*continue;*

*}*

*if(cur!=NULL){*

*cout<<" Volume: "<<cur->data.Volume<<"|";*

*if (cur->left!=NULL){*

*q.push(cur->left);*

*}*

*if (cur->right!=NULL){*

*q.push(cur->right);*

*}*

*}*

*}*

*}*

*void printInorder()*

*{*

*if (this->root == NULL)*

*{*

*cout<<"\n"<<"Empty tree"<<"\n";*

*return;*

*}*

*printInorder(this->root->left);*

*cout <<"Volume: "<<this->root->data.Volume<<"|";*

*printInorder(this->root->right);*

*}*

*struct node \* deleteNode(struct node \*p,struct metoxh data){*

*if(p->left == NULL && p->right == NULL){*

*if(p==this->root)*

*this->root = NULL;*

*delete p;*

*return NULL;*

*}*

*struct node \*t;*

*struct node \*q;*

*if(p->data.Volume < data.Volume){*

*p->right = deleteNode(p->right,data);*

*}*

*else if(p->data.Volume > data.Volume){*

*p->left = deleteNode(p->left,data);*

*}*

*else{*

*if(p->left != NULL){*

*q = inpre(p->left);*

*StructAssignTo(q->data,p->data);//p->data = q->data;*

*p->left=deleteNode(p->left,q->data);*

*}*

*else{*

*q = insuc(p->right);*

*StructAssignTo(q->data,p->data);// p->data= q->data;*

*p->right = deleteNode(p->right,q->data);*

*}*

*}*

*if(bf(p)==2 && bf(p->left)==1){ p = llrotation(p); }*

*else if(bf(p)==2 && bf(p->left)==-1){ p = lrrotation(p); }*

*else if(bf(p)==2 && bf(p->left)==0){ p = llrotation(p); }*

*else if(bf(p)==-2 && bf(p->right)==-1){ p = rrrotation(p); }*

*else if(bf(p)==-2 && bf(p->right)==1){ p = rlrotation(p); }*

*else if(bf(p)==-2 && bf(p->right)==0){ p = llrotation(p); }*

*return p;*

*}*

*struct node\* inpre(struct node\* p){*

*while(p->right!=NULL)*

*p = p->right;*

*return p;*

*}*

*struct node\* insuc(struct node\* p){*

*while(p->left!=NULL)*

*p = p->left;*

*return p;*

*}*

*struct node\* BinarySearch(struct node\* n, int WantedDate){*

*if (n->data.Volume==WantedDate){*

*cout<<"Found, Date is: "<<n->data.Date<<endl;*

*return n;*

*}*

*else if(n->data.Volume>WantedDate){*

*n=n->right;*

*//cout<<"Not Found, Volume is: "<<n->data.Volume;*

*BinarySearch(n, WantedDate);*

*}*

*else{*

*n=n->left;*

*//cout<<"Not Found, Volume is: "<<n->data.Volume;*

*BinarySearch(n, WantedDate);*

*}*

*return NULL;*

*}*

*~AVL(){*

*}*

*};*

*int main(){*

*AVL avltree;*

*int choice, x;*

*ifstream infile("agn.us.txt");*

*struct metoxh Stocks[AGN\_SIZE];*

*int first = 0;*

*int last = AGN\_SIZE-1;*

*int middle = (first + last)/2;*

*ReadValuesfromFile(infile, Stocks);*

*mergeSortVolume(Stocks, 0, AGN\_SIZE-1);*

*for (int i = 0; i<AGN\_SIZE; i++){*

*avltree.root=avltree.insert(avltree.root,Stocks[i]);*

*// cout<<avltree.root->data.Date<<endl;*

*}*

*do{*

*cout<<"AVL Tree Display with In Order Traversal\t(1)\nSearch by Volume and Print Volume\t\t(2)\nSearch by Volume and Set new Volume\t\t(3)\nSearch by Volume and Delete Record\t\t(4)\nFind Min Volume\t\t\t\t\t(5)\nFind Max Volume\t\t\t\t\t(6)\n\nExit\t\t\t\t\t\t(0)"<<endl;*

*cin>>choice;*

*system("cls");*

*switch(choice){*

*case 1:*

*avltree.printInorder();*

*system("PAUSE");*

*system("cls");*

*break;*

*case 2: \*/Ο χρήστης πληκτρολογεί μια volume και εξάγει μια volume\*/*

*long int find;*

*cin>>find ;*

*BinarySearchArr(Stocks , first, middle, last, find);*

*system("PAUSE");*

*system("cls");*

*break;*

*case 3: : /\* εισάγει ο χρήστης ένα volume από την συνάρτηση getDatefromKeyboard() και την αναζητά με δυαδική αναζήτηση σε πίνακα και πληκτρολογεί ο χρήστης νέο Volume\*/*

*long int find2;*

*int t;*

*cin>>find2 ;*

*t = BinarySearchArr(Stocks , first, middle, last, find2);*

*if(t == -1){*

*system("cls");*

*break;*

*}*

*cout<<"Give new Volume: ";*

*int newvol;*

*cin>>newvol;*

*Stocks[t].Volume = newvol;*

*for (int i = 0; i<AGN\_SIZE; i++){*

*avltree.root=avltree.insert(avltree.root,Stocks[i]);*

*// cout<<avltree.root->data.Date<<endl;*

*//cout<<"Tree is Rebuilding"<<endl;*

*//system("cls");*

*}*

*cout<<"Tree Rebuilt"<<endl;*

*system("PAUSE");*

*system("cls");*

*break;*

*case 4: \*/ /\* εισάγει ο χρήστης ένα volume από την συνάρτηση getDatefromKeyboard(), την αναζητά με δυαδική αναζήτηση σε πίνακα και διαγράφει την εγγραφή\*/*

*long int find3;*

*int j;*

*cin>>find3;*

*j = BinarySearchArr(Stocks , first, middle, last, find3);*

*if(j == -1){*

*system("cls");*

*break;*

*}*

*Stocks[j].Close = -1;*

*Stocks[j].Date = -1;*

*Stocks[j].High = -1;*

*Stocks[j].Low = -1;*

*Stocks[j].Open = -1;*

*Stocks[j].OpenInt = -1;*

*Stocks[j].Volume = -1;*

*for (int i = 0; i<AGN\_SIZE; i++){*

*if(Stocks[i].Date!=-1){*

*avltree.root=avltree.insert(avltree.root,Stocks[i]);*

*// cout<<avltree.root->data.Date<<endl;*

*//cout<<"Tree is Rebuilding"<<endl;*

*}*

*//system("cls");*

*}*

*cout<<"Tree Rebuilt"<<endl;*

*system("PAUSE");*

*system("cls");*

*break;*

*case 5: /\* εμφανίζει τη μικρότερη τιμή volume του πίνακα που βρίσκεται στην 1η θέση του πίνακα αφού είναι κατανεμημένες κατά αύξουσα σειρά\*/*

*mergeSortVolume(Stocks, 0, AGN\_SIZE-1);*

*cout<<"Date with Min Volume is: "<<Stocks[0].Date<<endl; //Meta apo to sorting sthn prwti thesi toy pinaka tha vrisketai to mikrotero Volume*

*system("PAUSE");*

*system("cls");*

*break;*

*case 6: /\* εμφανίζει τη μεγαλύτερη τιμή volume του πίνακα που βρίσκεται στην τελευταία θέση του πίνακα αφού είναι κατανεμημένες κατά αύξουσα σειρά\*/*

*mergeSortVolume(Stocks, 0, AGN\_SIZE-1);*

*cout<<"Date with Max Volume is: "<<Stocks[AGN\_SIZE-1].Date<<endl; //Meta apo to sorting sthn teleftaia thesi toy pinaka tha vrisketai to megalytero Volume*

*system("PAUSE");*

*system("cls");*

*break;*

*default:*

*system("cls");*

*break;*

*}*

*}while(choice!=0);*

*return 0;*

*}*

Όταν ο χρήστης πατήσει compile στον παραπάνω κώδικα τότε θα του εμφανιστεί το παρακάτω :

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Επίσης, όταν ο χρήστης πληκτρολογήσει τον αριθμό 5 τότε θα του εμφανιστεί το παρακάτω:  
