**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**INformatikos fakultetas**

**Lygiagretusis programavimas**

INDIVIDUALAUS DARBO ATASKAITA

**Vadovas**

Lek. Dominykas Barisas

Lek. Mindaugas Vasiljevas

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Studentas**

Lukas Gužauskas,

IFF-5/1 gr.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**KAUNAS, 2017-2018**

1. Užduotis

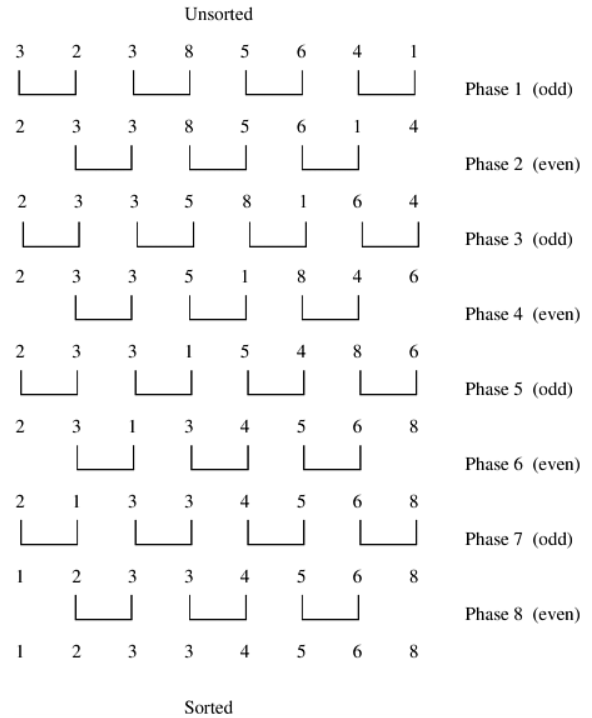
Mano pasirinkta užduotis – Odd-even transposition sort rūšiavimo algoritmas naudojant NetBeans IDE 8.2 JAVA.

1. Užduoties analizė

Rūšiavimas – viena labiausiai tyrinėjamų sričių kompiuterių moksle. Dėl didėjančių duomenų kiekių algoritmų efektyvumas tampa vis svarbesnis. Nauji algoritmai kuriami šiai užduočiai, tačiau netgi ir geriausi nuoseklūs algoritmai užtruks ilgai, jei duomenų daug. Daug greitesni yra lygiagretūs algoritmai. Vieną tokių, pritaikytų būtent lygiagrečiam atlikimui – **Odd-even transposition sort algorithm.**

* *Nuoseklusis rūšiavimas*

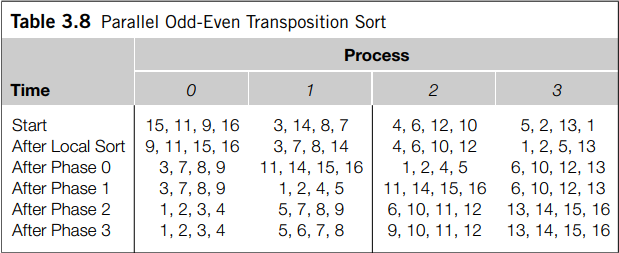
Odd-even transposition sort yra rikiuojama, palyginant gretimus elementus. Algoritmas rūšiuoja n elementus n/2 fazėse, kurių metu reikalinga atlikti n/2 sukeitimo operacijų. Šis algoritmas vykdo perjungimus tarp 2 fazių, vadinamų nelyginė(odd) ir lyginė(even). Nelyginės frazės metu elementai su nelyginiais indeksais yra palyginami su jų kaimynais iš dešinės ir, jei jie yra ne iš eilės, tai poros yra sukeičiamos, t.y. poros yra palyginamai sukeistos. Panašiai lyginė fazė vykdoma kaip nelyginė frazė, bet tik skirtingas poras . The odd-even transpostion sort iliustracija pateikta žemiau:



* *Lygiagretusis rūšiavimas*

Yra nesunku Odd-Even transposition rikiavimus atlikti lygiagrečiai ir toks rikiavimas yra efektyvus. Kiekvienas procesas turi po m elementų, kai:

kur (n – elementų masyvas, p – procesų skaičius).



Tuomet Local Sort funkcija kiekviename procese surūšiuoja to proceso elementus. Toliau, pvz. jei turime 4 procesus, tai rikiavimas atliekamas per 4 etapus (pvz., kai p = 10, tuomet yra 10 etapų). Lyginių (0, 2, ..., p-2) etapų metu kiekvienas procesas su lyginiu numeriu keičia savo elementus su kaimyninio proceso elementais iš dešinės, o nelyginių (1, 3, ..., p-1) etapų metu atitinkamai keičia elementus procesai su nelyginiu numeriu. Nelyginio etapo metu pradinis ir galiniai procesai elementų keitimo nevykdo. Elementų keitimas vykdomas tokia tvarka, kad procesas išlaiko 4 mažesnius elementus, o kaimynas iš dešinės 4 didesnius elementus. Tokiu būdu paskutinio etapo metu gauname didėjančią surūšiuotą skaičių aibę.

1. Sprendimo metodas

Mano algoritme yra dvi pagrindinės funkcijos – vietinio rikiavimo ir dalijamojo keitimo funkcijos.

Vietinio rikiavimo funkcija iššaukiama per gijų generavimo ciklą. AA masyvas yra duomenų elementai, nthread yra gijų skaičius.

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

OE[i] = new OddEven(AA[l], i, nlocal);

OE[i].start();

}

Kiekviena sukurta gija OddEven klasėje iškviečia run() funkciją. Indekso i reikšmė tikrinama kiekvienoje gijoje, ar %2 lygus 0. Jei ši sąlyga tenkinama, tai yra lyginis (Even sort) rikiavimas, jei ne, tai – nelyginis (Odd sort) . Toliau Even sort žingsnyje, jei A.get(i-1) yra daugiau už A.get(i), vertės poaibyje nėra išsidėstę didėjimo tvarka, tai reikšmės yra sukeičiamos. Taip pat, Odd sort žingsnyje, jei A.get(i) yra daugiau už A.get(i+1), vertės poaibyje nėra mažėjančios, vertės yra sukeičiamos.

int splitNo = id \* nlocal;

for (phase = 0; phase < nlocal; phase++) {

if (phase % 2 == 0) {

for (int i = 1 + splitNo; i < nlocal + splitNo; i += 2)

if (A.get(i-1) > A.get(i))

swap1(i);

}

else {

for (int i = 1 + splitNo; i < nlocal-1 + splitNo; i +=2)

if (A.get(i) > A.get(i+1))

swap2(i);

}

}

Paskutinės iteracijos pabaigoje turėsime tik dalinai surikiuotą seką. Tai yra tik vietinis rikiavimas didėjančią seką.

Dalijamojo keitimo funkcija iššaukiama per dvigubą ciklą. Išorinis ciklas kartoja funkcijos kvietimą daug kartų, kurių skaičius yra *phase* . Ciklas dirba nuo , nthread yra gijų skaičius, o *phase* – elementų keitimo tarp gijų etapas. Vidinis ciklas dirba nuo . AA masyvas yra duomenų elementai, nlocal – duomenų elementai padalinti kiekvienam procesui.

for (phase = 0; phase < nthreads; phase++){

for(int i = 0; i < (nthreads / 2); i++){

SE[i] = new SplitExchange(AA[l], i, nlocal, phase, nthreads);

SE[i].start();

}

Kiekviena sukurta gija SplitExchange iškviečia run() funkciją. Pirma, patikrinama ar procesas nėra paskutinis ir turi kaimyną iš dešinės: kai . Jei ši sąlyga tenkinama, tai viduje yra 2 iteracijos. Pirmas ciklas dirba nuo . Vykdomas elementų keitimas ir rikiavimas tarp proceso elementų ir proceso kaimyno iš dešinės, surikiuoti elementai surašomi į darbinį masyvą *wspace[]*.

int splitNo = id \* 2 + phase % 2;

if ((nthreads - splitNo) > 1) {

int i, j, k;

int[] wspace = new int[nlocal\*2];

splitNo = splitNo \* nlocal;

for (i = j = k = 0; k<nlocal\*2; k++) {

if (i >= nlocal || j < nlocal && A.get(i + splitNo)> A.get(j + splitNo + nlocal)){

wspace[k] = A.get(j + splitNo + nlocal);

j++;

}

else {

wspace[k] = A.get(i + splitNo);

i++;

}

}

for (i=0; i<nlocal\*2; i++)

A.set(i + splitNo, wspace[i]);

}

Paskutinės iteracijos metu išrikiuoti darbinio masyvo *wspace[]* elementai sudedami į pagrindinį A masyvą. Tokiu būdu A masyvo elementai gaunami surikiuoti pagal didėjančią seką.

1. Programos aprašymas ir pagrindinė dalys

Programavimui buvo naudota NetBeans IDE 8.2, JAVA kalbos.

Atminties priskyrimas, pradinių verčių sukūrimas (naudojama paprasta random() funkcija);

BufferedWriter wr = new BufferedWriter(new FileWriter(FileName));

// rasyti i faila

BufferedReader br = new BufferedReader(new FileReader(FileName));

// skaityti is failo

int n = 100; //data

int nthreads = 32; //threads

List<Integer>[] AA = new List[10];

Random random = new Random();

int siz = n;

//rasyti duomenu i faila

for (int i = 0; i < 10; i++){

for (int j = 0; j < siz; j++) {

int randomInt = random.nextInt(1000);

wr.write(randomInt + " ");

}

siz = siz \* 2;

wr.newLine();

}

//skaityti duomenu is failo

ReadData(br, AA, n);

Čia prasideda rūšiavimas ir pradedame skaičiuoti laiką:

// Startuoti laika

long startTime1 = System.currentTimeMillis();

int nlocal = AA[l].size() / nthreads;

//Vietinio rikiavimo funkcija

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

OE[i] = new OddEven1(AA[l], i, nlocal);

OE[i].start();

}

System.out.println(AA[l].size());

for(int i = 0; i < nthreads; i++)

OE[i].join();

System.out.println("Duomenu rinkinys nr: " + l);

System.out.println("\nafter local sort: ");

for (int i = 0; i < AA[l].size(); i++)

System.out.print(AA[l].get(i) + " ");

//Dalijamojo keitimo funkcija

int phase;

for (phase = 0; phase < nthreads; phase++){

for(int i = 0; i < (nthreads / 2); i++){

SE[i] = new SplitExchange(AA[l], i, nlocal, phase, nthreads);

SE[i].start();

}

for(int i = 0; i < (nthreads / 2); i++){

SE[i].join();

}

System.out.println("\nafter phase: " + phase);

for (int i = 0; i < AA[l].size(); i++)

System.out.print(AA[l].get(i) + " ");

}

System.out.println("\nafter all:");

for (int i = 0; i < AA[l].size(); i++)

System.out.print(AA[l].get(i) + " ");

//Stop laika

long totalTime1 = System.currentTimeMillis() - startTime1;

Rūšiavimo metodas naudojantis JAVA išplėsta Thread klase:

1. OddEven gijų klasės metodą:

@Override

public void run(){

int splitNo = id \* nlocal;

for (phase = 0; phase < nlocal; phase++){

if (phase % 2 == 0) {

for (int i = 1 + splitNo; i < nlocal + splitNo; i += 2)

if (A.get(i-1) > A.get(i))

swap1(i);

}

else {

for (int i = 1 + splitNo; i < nlocal-1 + splitNo; i +=2)

if (A.get(i) > A.get(i+1))

swap2(i);

}

}

}

2. SplitExchange gijų klasės metodą:

@Override

public void run(){

int splitNo = id \* 2 + phase % 2;

if ((nthreads - splitNo) > 1) {

int i, j, k;

int[] wspace = new int[nlocal\*2];

splitNo = splitNo \* nlocal;

for (i = j = k = 0; k<nlocal\*2; k++) {

if (i >= nlocal || j < nlocal && A.get(i + splitNo)> A.get(j + splitNo + nlocal)){

wspace[k] = A.get(j + splitNo + nlocal);

j++;

}

else {

wspace[k] = A.get(i + splitNo);

i++;

}

}

for (i=0; i<nlocal\*2; i++)

A.set(i + splitNo, wspace[i]);

}

}

1. Vykdymo laiko kitimo tyrimas

|  |  |
| --- | --- |
| Duomenų rinkinys nr: 1, n = 100. | |
| Procesų skaičius | T(5 bandymų vidurkis) |
| 1 | 38 |
| 2 | 90 |
| 4 | 71 |
| 8 | 107 |
| 16 | 443 |
| 32 | 654 |

|  |  |
| --- | --- |
| Duomenų rinkinys nr: 2, n = 200. | |
| Procesų skaičius | T(5 bandymų vidurkis) |
| 1 | 40 |
| 2 | 31 |
| 4 | 44 |
| 8 | 90 |
| 16 | 407 |
| 32 | 992 |

|  |  |
| --- | --- |
| Duomenų rinkinys nr: 3, n = 400. | |
| Procesų skaičius | T(5 bandymų vidurkis) |
| 1 | 75.4 |
| 2 | 68 |
| 4 | 87 |
| 8 | 208 |
| 16 | 825 |
| 32 | 1880 |

|  |  |
| --- | --- |
| Duomenų rinkinys nr: 4, n = 800. | |
| Procesų skaičius | T(5 bandymų vidurkis) |
| 1 | 111 |
| 2 | 188 |
| 4 | 272 |
| 8 | 701 |
| 16 | 1015 |
| 32 | 1390 |

|  |  |
| --- | --- |
| Duomenų rinkinys nr: 5, n = 1600. | |
| Procesų skaičius | T(5 bandymų vidurkis) |
| 1 | 221 |
| 2 | 486 |
| 4 | 528 |
| 8 | 878 |
| 16 | 1117 |
| 32 | 1945 |

|  |  |
| --- | --- |
| Duomenų rinkinys nr: 6, n = 3200. | |
| Procesų skaičius | T(5 bandymų vidurkis) |
| 1 | 496 |
| 2 | 835 |
| 4 | 992 |
| 8 | 1040 |
| 16 | 1780 |
| 32 | 3396 |

|  |  |
| --- | --- |
| Duomenų rinkinys nr: 7, n = 6400. | |
| Procesų skaičius | T(5 bandymų vidurkis) |
| 1 | 1471 |
| 2 | 1440 |
| 4 | 1121 |
| 8 | 1945 |
| 16 | 3652 |
| 32 | 6010 |

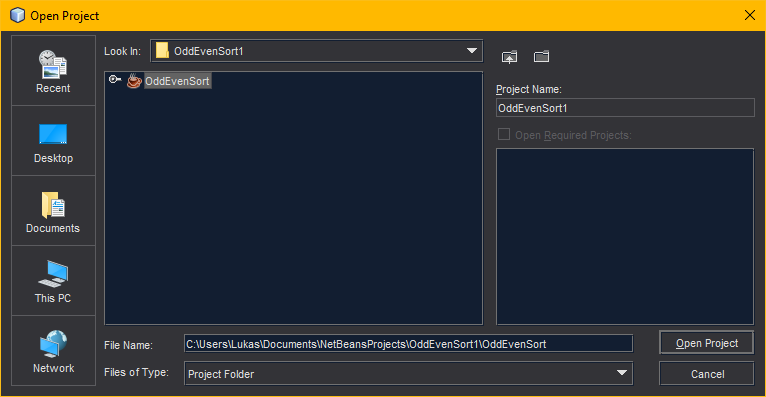
|  |  |
| --- | --- |
| Duomenų rinkinys nr: 8, n = 12800. | |
| Procesų skaičius | T(5 bandymų vidurkis) |
| 1 | 3401 |
| 2 | 2340 |
| 4 | 2313 |
| 8 | 3135 |
| 16 | 7086 |
| 32 | 11662 |

|  |  |
| --- | --- |
| Duomenų rinkinys nr: 9, n = 25600. | |
| Procesų skaičius | T(5 bandymų vidurkis) |
| 1 | 14949 |
| 2 | 7750 |
| 4 | 5424 |
| 8 | 7900 |
| 16 | 13835 |
| 32 | 21713 |

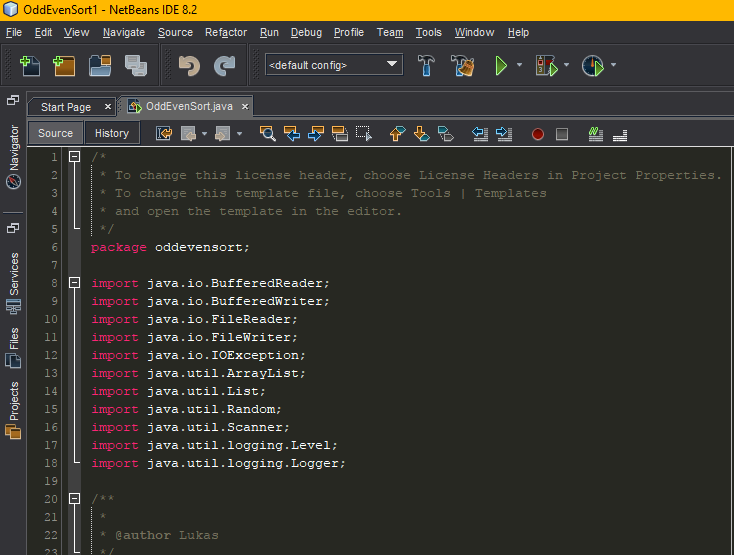
|  |  |
| --- | --- |
| Duomenų rinkinys nr: 10, n = 51200. | |
| Procesų skaičius | T(5 bandymų vidurkis) |
| 1 | 42489 |
| 2 | 23303 |
| 4 | 15510 |
| 8 | 17597 |
| 16 | 27844 |
| 32 | 45780 |

1. Testavimas ir programos instaliavimo bei vykdymo instrukcija

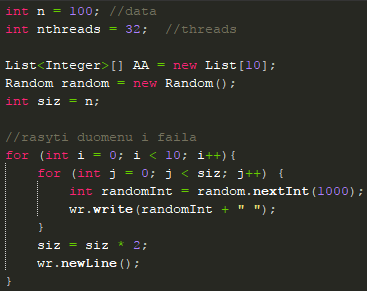
Atidaryti programinės įrangos NetBeans priemonę, kuri naudojama kurti programas JAVA kalba. NetBeans priemonėje atidaryti projektą.



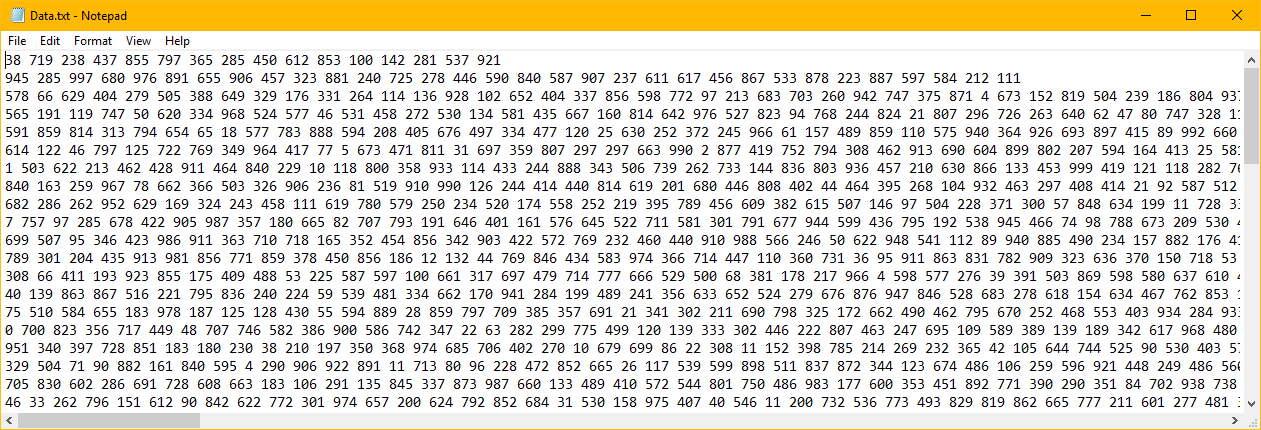
Pasirinkti projektą pavadinimu „OddEvenSort“ ir spausti „Open Project “. Atveriamas OddEvenSort.java programos kodas.



Kad išbandyti programą su skirtingos apimties duomenimis ir su skirtingu procesų skaičiumi, tai apibrėžta *n* kaip duomenų skaičius, *nthreads* – gijų skaičius. Pradinė *n* reikšmė yra 100, o *nthreads* – 1 gijoa. AA masyvas turi indeksą iki 10 ir kiekvienas indeksas turi skirtingą duomenų kiekį. Sukurta Random klasė, iššaukiama per dvigubą ciklą. Išorinis ciklas dirba nuo 0 iki 10. Vidinis ciklas dirbti nuo 0 iki *siz*, *siz* – duomenų dydis. Per iteraciją *siz* dididamas 2 kartus, tai masyvas su indeksu 0 turi 1000 skaitmenų, o su sekančiu indeksu – 2000, ir t.t. Visi sukurti masyvai įrašomi į failą.

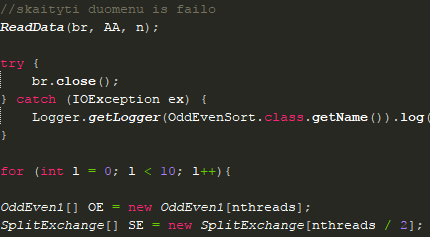


Paskutinės iteracijos pabaigoje galima atidaryti duomenų tekstą.

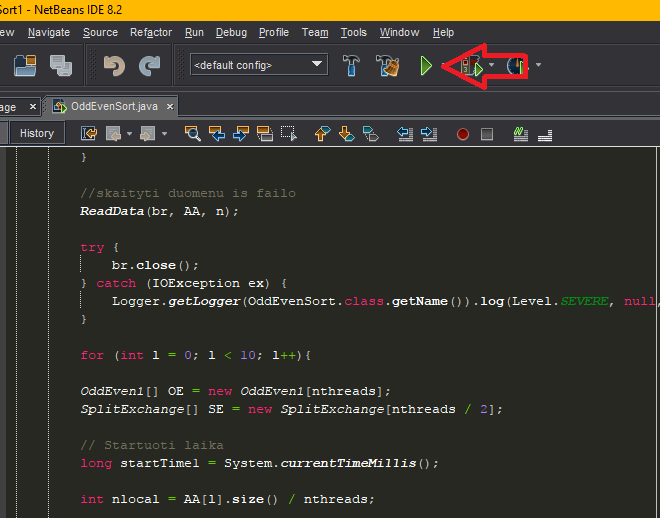


Kiekvienoje teksto eilutėje matyti skirtingo masyvo neišrikiuoti duomenys.

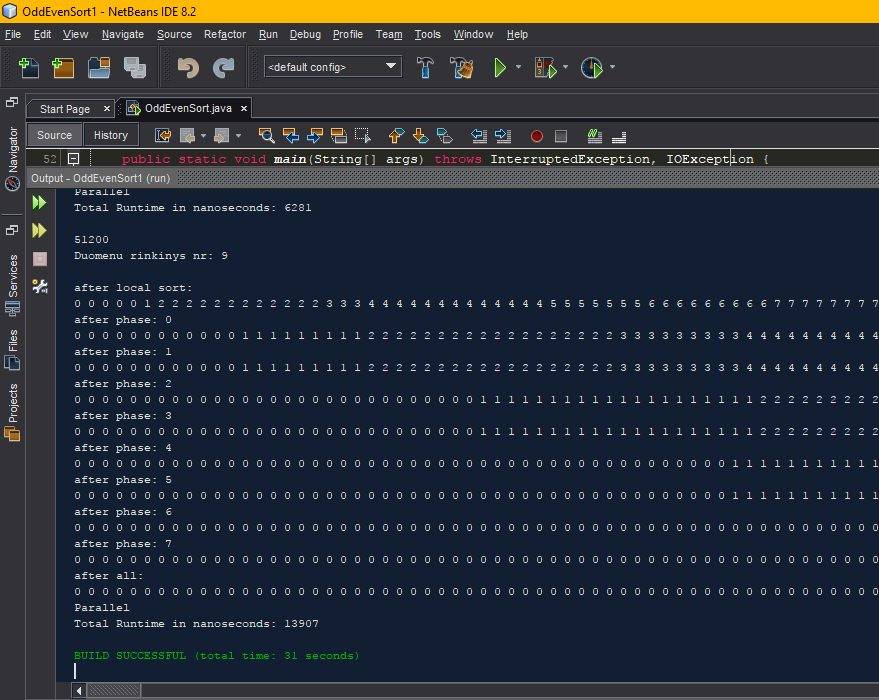
Grįžti atgal į programos kodą, ReadData metodas – tai skaitymas iš failo. Vykdomas metodas, kuris perskaitytus duomenis sudeda į AA masyvus. Praleidžiama iteracija dirbti nuo 0 to 10, su kiekvienu skirtingo indeksu masyvu vykdomos gijų klases.



Ir spausti F6 mygtuką, kad paleisti vykdymo programą,



rezultatus galima matyti:



1. Išvados

Matome, kad JAVA lygiagretusis rikiavimas greičiausiai rikiuoja, kai duomenų daugiau kaip 512000 prie 2-16 gijų, nei prie 1 ir 32 gijų. O kada mažesnis duomenų skaičius (100-400), tai nuo 1 iki 4 gijų darbo laikas buvo tolygus, po 4 gijų darbo laikas pradeda kilti į didėjančiai. Algoritmas dirba efektyviai kai apdorojamas didesnis duomenų skaičius, taip pat gijų skaičius.

1. Literatūra:

[Introduction to Parallel Computing, Vipin Kumar.](https://github.com/vineethshankar/pagerank/blob/master/Introduction%20to%20Parallel%20Computing%2C%20Second%20Edition-Ananth%20Grama%2C%20Anshul%20Gupta%2C%20George%20Karypis%2C%20Vipin%20Kumar.pdf)

[An Introduction to Parallel Programming, Peter Pacheco](http://www.sciencedirect.com/science/book/9780123742605)

[en.wikipedia.org/wiki/Odd%E2%80%93even\_sort](https://en.wikipedia.org/wiki/Odd%E2%80%93even_sort)