|  |
| --- |
| Kauno Technologijos Universitetas |
| Algoritmų sudarymas ir analizė |
| Laboratorinis darbas Nr. 1, 3 variantas |

|  |
| --- |
| Atliko: Tautvydas Petkus IFF-1  Kaunas 2013 |

**Užduotis**

Rikiavimo uždavinys. Palyginkite tris rūšiavimo algoritmus, kai rūšiavimas atliekamas masyve ir dinaminiame sąraše. Algoritmai:

a) Rūšiavimas „suliejimu“.

b) Greitasis rūšiavimas „Quick sort“.

c) Rūšiavimo algoritmas „Bucket sort“.

**Realizavimas**

Laboratorinio darbo programos realizuotos python programavimo kalbą naudojant Aptana Studio 3 programinę aplinką

Laboratorinio darbo metu rūšiuojama atsitiktinė sveikų skaičių imtis nuo -2.500.000.000 iki 2.500.000.000

**Rūšiavimo algoritmų tyrimas**

**Rūšiavimas „suliejimu“**

Šis rikiavimo metodas remiasi „Skaldyk ir valdyk“ paradigma. Metodas – rekursinis. Algoritmo viekimo tvarka:

\*Visa duomenų imtis, turinti n elementų, padalinama į dvi imtis, kuriuose yra n/2 elementų.

\*Šios dvi imtys rikiuojamos rekursyviai naudojant rikiavimą suliejimu.

\*Sulieti dvi surikiuotas imtis tam, kad gauti surikiuotą atsakymą.

Imtis rekursyviai dalinama, ir tada suliejant šios imtys yra rikiuojamos. Teoriškai šio algoritmo sudėtingumas yra:

n – rikiuojamų elementų kiekis.

**Rūšiavimas „suliejimu“ masyve**

Dešinėje pažymėta, kiek kartų vykdomos eilutės. Skaičiavimo algoritme skaičiuosime tik tas eilutes, kurios yra susijusios su algoritmu.

**L1\_1\_1.py**

import os

import struct

import time

const = 0;

def rusiuoti(f, w, offset):

global const 1

if offset[0] == offset[1]: 1

const += 1 1

return offset 1

if offset[0] + 1 == offset[1]: 1

of1\_f = of1\_t = offset[0] 1

of2\_f = of2\_t = offset[1] 1

const += 2 1

else: 1

of1\_f = offset[0] 1

of1\_t = (offset[0] + offset[1]) / 2 1

const += 2 1

if of1\_t + 1 > offset[1]: 1

of2\_f = offset[1] 1

const += 1 1

else: 1

of2\_f = of1\_t + 1 1

const += 1 1

of2\_t = offset[1] 1

const += 1 1

return sujungti(f, w, rusiuoti(f, w, [of1\_f, of1\_t]), rusiuoti(f, w, [of2\_f, of2\_t])) 2Trusiuoti

def sujungti(f, w, left, right):

global const 1

test\_list = [] 1

begin = left[0] 1

i = j = 0 1

const += 3 1

while i < (left[1] - left[0] + 1) and j < (right[1] - right[0] + 1): n

var\_i = imti(f, (left[0]\*4)+i\*4) Timti

var\_j = imti(f, (right[0]\*4)+j\*4) Timti

const += 2 1

if var\_i < var\_j: 1

var\_bin = imti\_b(f, (left[0]\*4)+i\*4) Timti\_b

deti(w, begin\*4, var\_bin) Tdeti

begin += 1 1

i += 1 1

const += 4 1

else: 1

var\_bin = imti\_b(f, (right[0]\*4)+j\*4) Timti\_b

deti(w, begin\*4, var\_bin) Tdeti

begin += 1 1

j += 1 1

const += 4 1

for tt in xrange(left[0] + i, left[1] + 1): n

var = imti\_b(f, tt\*4) Timti\_b

deti(w, begin\*4, var) Tdeti\_b

begin += 1 1

const += 3 1

for tt in xrange(right[0] + j, right[1] + 1): n

var = imti\_b(f, tt\*4) Timti\_b

deti(w, begin\*4, var) Tdeti

begin += 1 1

const += 3 1

for tt in xrange(left[0], right[1] + 1): n

var = imti\_b(w, tt\*4) Timti\_b

deti(f, tt\*4, var) Tdeti

const += 2 1

return [left[0], right[1]] 1

def rik(failo\_pav, rez\_failo\_pav="Laikinas"):

global const 1

f = open(failo\_pav, 'r+b') 1

open(rez\_failo\_pav, 'w').close() 1

w = open(rez\_failo\_pav, 'r+b') 1

w.truncate() 1

length = os.path.getsize(failo\_pav) 1

#failo\_isvedimas(f, length)

time1 = time.time() 1

rusiuoti(f, w, [0, length/4 - 1]) Trusiuoti

time2 = time.time() 1

#print "-----ISRIKIUOTAS-----"

skirtumas = time2 - time1 1

#failo\_isvedimas(f, length)

print "Bendras uztruktas laikas sekundemis: %(time)s" %{'time': skirtumas} 1

const += 13 1

print "Eiluciu kiekis: %(const)s" %{'const': const} 1

def script(parent): #Skirtas testavimo automatizavimui

global const

for x in xrange(1, 18):

const = 0

file\_name = "T%(test)s" %{'test': x}

print file\_name

rik(parent+file\_name)

def imti(f, i):

global const 1

f.seek(i) 1

var = struct.unpack('i', f.read(4))[0] 1

const += 2 1

return var 1

def imti\_b(f, i):

global const 1

f.seek(i) 1

var = f.read(4) 1

const += 2 1

return var 1

def deti(f, i, r):

global const 1

f.seek(i) 1

f.write(r) 1

const += 2 1

def failo\_isvedimas(f, length):

global const 1

for j in xrange(0, length, 4): n

print imti(f, j) 1

const += 1 1

T(n) = 1+1+1+1+1+1+1+Trusiuoti(n)+1+1+1+1+1 = Trusiuoti(n) + 12

Trusiuoti(n) = 1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+Tsujungti(2Trusiuoti(n/2))

Tsujungti(n) = 1+1+1+1+1+n+Timti(n)+Timti(n)+1+1+Timti\_ b(n)+Tdeti(n)+1+1+1+1+Timti\_b(n)+Tdeti(n)+1+1+1+n+ Timti\_b(n)+Tdeti(n)+1+1+n+Timti\_b + Tdeti+1+1+n+Timti\_b+Tdeti+1+1

Timti(n) = 5

Timti\_b(n) = 5

Tdeti(n) = 4

Tsujungti(n) = 5 + n + 5 + 5 + 2 + 5 + 4 + 4 + 5 + 4 + 3 + n + 5 + 4 +2 + n + 5 + 4 + 2 + n + 5 + 4 + 2 = 4n + 75

Trusiuoti(n) = 20 + 2Trusiuoti(n/2) + 4n + 75 = 2Trusiuoti(n/2) + 4n + 95

T(n) = 2Trusiuoti(n/2) +4n + 95

Taikant pradinę teoriją, gauname:

f(n) = 4n + 95

f(n) = n ^ log2(2)

T(n) = O(n \* log2(n))

**Rūšiavimas „suliejimu“ sąraše**

**L1\_1\_2.py**

import os

import struct

import time

const = 0

def rusiuoti(f, w, offset):

global const 1

if offset[0] == offset[1]: 1

const += 1 1

return offset 1

if offset[0] + 1 == offset[1]: 1

of1\_f = of1\_t = offset[0] 1

of2\_f = of2\_t = offset[1] 1

const += 2 1

else: 1

of1\_f = offset[0] 1

of1\_t = (offset[0] + offset[1]) / 2 1

const += 2 1

if of1\_t + 1 > offset[1]: 1

of2\_f = offset[1] 1

const += 1 1

else: 1

of2\_f = of1\_t + 1 1

const += 1 1

of2\_t = offset[1] 1

const += 1 1

return sujungti(f, w, rusiuoti(f, w, [of1\_f, of1\_t]), rusiuoti(f, w, [of2\_f, of2\_t])) Tsujungti(2Trusiuoti)

def sujungti(f, w, left, right):

global const 1

test\_list = [] 1

begin = left[0] 1

i = j = 0 1

const += 3 1

while i < (left[1] - left[0] + 1) and j < (right[1] - right[0] + 1): n

var\_i = imti\_list(f, (left[0]\*4)+i\*4) Timti\_list

var\_j = imti\_list(f, (right[0]\*4)+j\*4) Timti\_list

const += 2 1

if var\_i < var\_j: 1

var\_bin = imti\_list\_b(f, (left[0]\*4)+i\*4) Timti\_list\_b

deti\_list(w, begin\*4, var\_bin) Tdeti\_list

begin += 1 1

i += 1 1

const += 4 1

else: 1

var\_bin = imti\_list\_b(f, (right[0]\*4)+j\*4) Timti\_list\_b

deti\_list(w, begin\*4, var\_bin) Tdeti\_list

begin += 1 1

j += 1 1

const += 4 1

for tt in xrange(left[0] + i, left[1] + 1): n

var = imti\_list\_b(f, tt\*4) Timti\_list\_b

deti\_list(w, begin\*4, var) Tdeti\_list

begin += 1 1

const += 3 1

for tt in xrange(right[0] + j, right[1] + 1): n

var = imti\_list\_b(f, tt\*4) Timti\_list\_b

deti\_list(w, begin\*4, var) Tdeti\_list

begin += 1 1

const += 3 1

for tt in xrange(left[0], right[1] + 1): n

var = imti\_list\_b(w, tt\*4) Timti\_list\_b

deti\_list(f, tt\*4, var) Tdeti\_list

const += 2 1

return [left[0], right[1]] 1

def rik(failo\_pav, rez\_failo\_pav="Laikinas"):

global const 1

f = open(failo\_pav, 'r+b') 1

open(rez\_failo\_pav, 'w').close() 1

w = open(rez\_failo\_pav, 'r+b') 1

w.truncate() 1

length = os.path.getsize(failo\_pav) 1

#failo\_isvedimas\_list(f)

time1 = time.time() 1

rusiuoti(f, w, [0, length/4 - 1]) Trusiuoti

time2 = time.time() 1

skirtumas = time2 - time1 1

#print "-----ISRIKIUOTAS-----"

#failo\_isvedimas\_list(f)

print "Bendras uztruktas laikas sekundemis: %(time)s" %{'time': skirtumas} 1

const += 13 1

print "Eiluciu kiekis: %(const)s" %{'const': const} 1

def script(parent): #skirtas testavimu automatizavimui

global const

for x in xrange(1, 18):

const = 0

file\_name = "T%(test)s" %{'test': x}

print file\_name

rik(parent+file\_name)

def pereiti(f, j):

global const 1

i = f.tell() 1

const += 1 1

if i < j: 1

for x in xrange(i+4, j+4, 4): n

f.seek(x) 1

const += 1 1

if i > j: 1

for x in xrange(i-4, j-4, -4): n

f.seek(x) 1

const += 1 1

if i == j: 1

pass 1

const += 1 1

def imti\_list(f, j):

global const 1

pereiti(f, j) Tpereiti

val = f.read(4) 1

const += 2 1

if val == '': 1

const += 1 1

return '' 1

else: 1

const += 1 1

return struct.unpack('i', val)[0] 1

def imti\_list\_b(f, j):

global const 1

pereiti(f, j) Tpereiti

const += 2 1

return f.read(4) 1

def deti\_list(f, j, r):

global const 1

pereiti(f, j) Tpereiti

f.write(r); 1

const += 2 1

def failo\_isvedimas\_list(f):

global const 1

value = 0 1

f.seek(0) 1

offset = 0 1

const += 3 1

while value != '': n

value = imti\_list(f, offset) Timti\_list

const += 1 1

if value == '': 1

const += 1 1

break 1

print (value) 1

offset += 4 1

const += 2 1

T(n) = 1+1+1+1+1+1+1+Trusiuoti(n)+1+1+1+1+1 = Trusiuoti(n) + 12

Trusiuoti(n) = 1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+Tsujungti(2Trusiuoti(n/2))

Tsujungti(n) = 1+1+1+1+1+n+Timti\_list(n)+Timti\_list(n)+1+1+Timti\_list\_b(n)+Tdeti\_list(n)+1+1+1+1+Timti\_list\_b(n)+Tdeti\_list(n)+1+1+1+n+ Timti\_list\_b(n)+Tdeti\_list(n)+1+1+n+Timti\_list\_b + Tdeti\_list+1+1+n+Timti\_list\_b+Tdeti\_list+1+1

Tpereiti(n) = 1+1+1+1+n+1+1+1+n+1+1+1+1+1= 2n + 12

Timti\_list(n) = 1+Tpereiti(n)+1+1+1+1+1+1+1+1= 2n + 12 + 9 = 2n + 21

Timti\_list\_b(n) = 1+Tpereiti(n)+1+1 = 2n + 12 + 3 = 2n + 15

Tdeti\_list(n) = 1+Tpereiti(n)+1+1 = 2n + 12 + 3 = 2n + 15

Tsujungti(n) = 5 + n + 2n + 21 + 2n + 21 + 2 + 2n + 15 + 2n + 15 + 4 + 2n + 15 + 2n + 15 + 3 + n + 2n + 15 + 2n + 15 +2 + n + 2n + 15 + 2n + 15 + 2 + n + 2n + 15 + 2n + 15 + 2 = 28n + 212

Trusiuoti(n) = 20 + 2Trusiuoti(n/2) + 28n + 212 = 2Trusiuoti(n/2) + 28n + 232

T(n) = 2Trusiuoti(n/2) + 28n + 244

Taikant pradinę teoriją, gauname:

f(n) = 28n + 232

f(n) = n ^ log2(2)

T(n) = O(n \* log2(n))

**Algoritmo ekspermentinis tyrimas**

Algoritmas testuojamas rikiuojant sveikuosius skaičius. Buvo sugeneruota daug atsitiktinių skaičių, nurodant failo dydį.  
Rikiuojant buvo skaičiuojama rikiavimo trukmė, palyginamas rikiavimas masyve ir sąraše. Duomenų imtis 200 iki 100.000.

Atlikus tyrimą – gauti tokie rezultatai:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Elementų sk. | Masyvas | | Sąrašas | |
| Trukmė (s) | Operacijų sk. | Trukmė (s) | Operacijų sk. |
| 200 | 0.0280029773712 | 30480 | 0.046942949295 | 136834 |
| 400 | 0.0377688407898 | 68968 | 0.138601064682 | 447930 |
| 600 | 0.0606980323792 | 110389 | 0.27435708046 | 905297 |
| 800 | 0.0848579406738 | 153902 | 0.463610172272 | 1533798 |
| 1.000 | 0.108127832413 | 197436 | 0.703898906708 | 2346424 |
| 1.200 | 0.134846925735 | 244737 | 0.982892990112 | 3315345 |
| 1.400 | 0.161478042603 | 292258 | 1.29967403412 | 4461120 |
| 1.600 | 0.186622142792 | 339798 | 1.65995502472 | 5710916 |
| 1.800 | 0.214425086975 | 387430 | 2.09401583672 | 7260512 |
| 2.000 | 0.240694046021 | 435230 | 2.50607204437 | 8734050 |
| 3.000 | 0.383118152618 | 692010 | 5.43825507164 | 18988868 |
| 5.000 | 0.683442831039 | 1226696 | 14.874557972 | 52323344 |
| 10.000 | 1.47425580025 | 2651839 | 57.7681789398 | 204461345 |
| 20.000 | 3.17960715294 | 5703938 | 228.491272926 | 809978924 |
| 30.000 | 5.03977489471 | 8886143 | 507.873284101 | 1814532217 |
| 50.000 | 8.82009005547 | 15590694 | 1403.47742891 | 5024689734 |
| 100.000 | 18.7796959877 | 33181753 | 5600.19350195 | 20036545219 |

**Laiko trukmė pagal elementų kiekį**

Mėlyna – masyvas. Orandžinė - sąrašas

## **Greitasis rūšiavimas „Quick Sort“**

Šis rikiavimo metodas remiasi „Skaldyk ir valdyk“ paradigma. Metodas – rekursinis. Algoritmo viekimo tvarka:

\*Pertvarkomas masyvas į du kitus masyvus taip, kad tam tikras pasirinktas elementas būtų didesnis arba lygus už iš vieno masyvų visus elementus, o už kito masyvo elementus jis būtų mažesnis arba lygus. Elementas įvardintas kaip tam tikras elementas yra paskaičiuojamas pačioje pertvarkymo pusėje

\*Surikiuojami tie du masyvai rekursyviniais kvietimais į rikiavimo funkciją

\*Kadangi masyvai jau yra surikiuoti ir vienoje vietoje, taigi nėra reikalo juos sujungti. Visas masyvas surikiuotas

Teoriškai šio algoritmo sudėtingumas yra:

n – rikiuojamų elementų kiekis.

**Rūšiavimas „QuickSort“ masyve**

Dešinėje pažymėta, kiek kartų vykdomos eilutės. Skaičiavimo algoritme skaičiuosime tik tas eilutes, kurios yra susijusios su algoritmu.

**L1\_2\_1.py**

import os

import struct

import time

const = 0

def dalinti ( kaire, desine, f):

global const 1

t = indeksas = kaire 1

var\_kaire = imti(f, kaire) Timti

const += 2 1

for j in xrange( kaire + 4, desine + 4, 4): n

var\_j = imti(f, j) Timti

const += 1 1

if var\_j < var\_kaire: 1

indeksas += 4 1

sukeisti(f, indeksas, j) Tsukeisti

const += 2 1

sukeisti(f, kaire, indeksas) Tsukeisti

const += 1 1

return indeksas 1

def rusiuoti( kaire, desine, failas):

global const 1

if kaire < desine : 1

indeksas = dalinti( kaire, desine, failas) Tdalinti

rusiuoti(kaire, indeksas-4, failas) Trusiuoti

rusiuoti(indeksas+4, desine, failas) Trusiuoti

const += 3 1

def rik(failo\_pav):

global const 1

f = open(failo\_pav, 'r+b') 1

length = os.path.getsize(failo\_pav) 1

*#failo\_isvedimas(f, length)*

time1 = time.time() 1

rusiuoti (0, length-4 ,f) Trusiuoti

time2 = time.time() 1

skirtumas = time2 - time1 1

*#print "-----ISRIKIUOTAS-----"*

*#failo\_isvedimas(f, length)*

print "Bendras uztruktas laikas sekundemis: %(time)s" %{'time': skirtumas} 1

const += 10 1

print "Eiluciu kiekis: %(const)s" %{'const': const} 1

def script(parent): #naudojamas testų automatizavimui

global const

for x in xrange(1, 18):

const = 0

file\_name = "T%(test)s" %{'test': x}

print file\_name

rik(parent+file\_name)

def imti(f, i):

global const 1

f.seek(i) 1

var = struct.unpack('i', f.read(4))[0] 1

const += 3 1

return var 1

def imti\_b(f, i):

global const 1

f.seek(i) 1

var = f.read(4) 1

const += 3 1

return var 1

def deti(f, i, r):

global const 1

f.seek(i) 1

f.write(r) 1

const += 2 1

def sukeisti(f, i, j):

global const 1

f.seek(i) 1

var\_a = f.read(4) 1

f.seek(j) 1

var\_b = f.read(4) 1

f.seek(i) 1

f.write(var\_b) 1

f.seek(j) 1

f.write(var\_a) 1

const += 8 1

def failo\_isvedimas(f, length):

global const 1

for j in xrange(0, length, 4): n

print imti(f, j) Timti

const += 1 1

T(n) = 9 + 2T(n/2) + n + 30 = 2T(n/2) + n + 39

Trusiuoti = 2 + n + 27 + 2Trusiuoti + 1 = 2Trusiuoti(n/2) + n + 30

Tdalinti(n) = 2 + Timti + 1 + n + Timti + 3 + Tsukeisti + 1 + Tsukeisti + 2 = 2 + 5 + 1 + n + 5 + 3 + 10 + 1 + 10 + 2 = n + 27

Timti(n) = 5

Timti\_b(n) = 5

Tdeti(n) = 4

Tsukeisti(n) = 10

Kadangi naudojama rekursija, tai T(n) = O(n²)

**Rūšiavimas „QuickSort“ sąraše**

**L1\_2\_2.py**

**import os**

import struct

import time

const = 0

def dalinti ( kaire, desine, f):

global const 1

t = indeksas = kaire 1

var\_kaire = imti\_list(f, kaire) Timti\_list

const += 2 1

for j in xrange( kaire + 4, desine + 4, 4): n

var\_j = imti\_list(f, j) Timti\_list

const += 1 1

if var\_j < var\_kaire: 1

indeksas += 4 1

sukeisti\_list(f, indeksas, j) Tsukeisti\_list

const += 2 1

sukeisti\_list(f, kaire, indeksas) Tsukeisti\_list

const += 1 1

return indeksas 1

def rusiuoti( kaire, desine, failas):

global const 1

if kaire < desine : 1

indeksas = dalinti( kaire, desine, failas) Tdalinti

rusiuoti(kaire, indeksas-4, failas) Trusiuoti

rusiuoti(indeksas+4, desine, failas) Trusiuoti

const += 3 1

def rik(failo\_pav):

global const 1

f = open(failo\_pav, 'r+b') 1

length = os.path.getsize(failo\_pav) 1

*#failo\_isvedimas\_list(f)*

time1 = time.time() 1

rusiuoti (0, length-4 ,f) Trusiuoti

time2 = time.time() 1

skirtumas = time2 - time1 1

*#print "-----ISRIKIUOTAS-----"*

*#failo\_isvedimas\_list(f)*

print "Bendras uztruktas laikas sekundemis: %(time)s" %{'time': skirtumas} 1

const += 10 1

print "Eiluciu kiekis: %(const)s" %{'const': const} 1

def script(parent): #naudojamas testų automatizavimui

global const

for x in xrange(1, 18):

const = 0

file\_name = "T%(test)s" %{'test': x}

print file\_name

rik(parent+file\_name)

def pereiti(f, j):

global const 1

i = f.tell() 1

const += 1 1

if i < j: 1

for x in xrange(i+4, j+4, 4): n

f.seek(x) 1

const += 1 1

if i > j: 1

for x in xrange(i-4, j-4, -4): n

f.seek(x) 1

const += 1 1

if i == j: 1

pass 1

const += 1 1

def imti\_list(f, j):

global const 1

pereiti(f, j) Tpereiti

val = f.read(4) 1

const += 2 1

if val == '': 1

const += 1 1

return '' 1

else: 1

const += 1 1

return struct.unpack('i', val)[0] 1

def imti\_list\_b(f, j):

global const 1

pereiti(f, j) Tpereiti

const += 2 1

return f.read(4) 1

def deti\_list(f, j, r):

global const 1

pereiti(f, j) Tpereiti

f.write(r) 1

const += 2 1

def failo\_isvedimas\_list(f):

global const 1

value = 0 1

f.seek(0) 1

offset = 0 1

const += 3 1

while value != '': 1

value = imti\_list(f, offset) Timti\_list

const += 1 1

if value == '': 1

const += 1 1

break 1

print (value) 1

offset += 4 1

const += 2 1

def sukeisti\_list(f, i, j):

global const 1

var\_a = imti\_list\_b(f, i) Timti\_list\_b

var\_b = imti\_list\_b(f, j) Timti\_list\_b

deti\_list(f, i, var\_b) Tdeti\_list

deti\_list(f, j, var\_a) Tdeti\_list

const += 4 1

T(n) = 2T(n/2) + 21n + 185

Trusiuoti(n) = 3 + 21n + 173 + 2Trusiuoti(n/2) = 2Trusiuoti(n/2) + 21n + 176

Tdalinti(n) = 2 + 2n + 21 + 1 + n + 2n + 21 + 3 + 8n + 61 + 1 + 8n + 61 + 2 = 21n + 173

Tpereiti(n) = 1+1+1+1+n+1+1+1+n+1+1+1+1+1= 2n + 12

Timti\_list(n) = 1+Tpereiti(n)+1+1+1+1+1+1+1+1= 2n + 12 + 9 = 2n + 21

Timti\_list\_b(n) = 1+Tpereiti(n)+1+1 = 2n + 12 + 3 = 2n + 15

Tdeti\_list(n) = 1+Tpereiti(n)+1+1 = 2n + 12 + 3 = 2n + 15

Tsukeisti\_list(n) = 1 + 2n + 15 + 2n + 15 + 2n + 15 + 2n + 15 = 8n + 61

Kadangi naudojama rekursija, tai T(n) = O(n²)

**Algoritmo ekspermentinis tyrimas**

Algoritmas testuojamas rikiuojant sveikuosius skaičius. Buvo sugeneruota daug atsitiktinių skaičių, nurodant failo dydį.  
Rikiuojant buvo skaičiuojama rikiavimo trukmė, palyginamas rikiavimas masyve ir sąraše. Duomenų imtis 200 iki 100.000.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Elementų sk. | Masyvas | | Sąrašas | |
| Trukmė (s) | Operacijų sk. | Trukmė (s) | Operacijų sk. |
| 200 | 0.00722980499268 | 16607 | 0.0264909267426 | 77174 |
| 400 | 0.0161151885986 | 33653 | 0.0766530036926 | 233878 |
| 600 | 0.0260949134827 | 55203 | 0.140232086182 | 458394 |
| 800 | 0.0358850955963 | 78335 | 0.235728979111 | 783338 |
| 1.000 | 0.0518169403076 | 111904 | 0.370328903198 | 1236110 |
| 1.200 | 0.0570778846741 | 126472 | 0.498943805695 | 1686500 |
| 1.400 | 0.0687489509583 | 153458 | 0.6734790802 | 2295624 |
| 1.600 | 0.0854139328003 | 190399 | 0.849053144455 | 2892716 |
| 1.800 | 0.0842127799988 | 194545 | 1.06589412689 | 3697035 |
| 2.000 | 0.11750292778 | 258097 | 1.31132292747 | 4491840 |
| 3.000 | 0.18168592453 | 406869 | 2.7841091156 | 9545275 |
| 5.000 | 0.313282012939 | 698795 | 7.66326284409 | 26701233 |
| 10.000 | 0.653005123138 | 1475820 | 29.0159919262 | 103059070 |
| 20.000 | 1.46271204948 | 3178045 | 114.94175601 | 407102394 |
| 30.000 | 2.37878799438 | 5067712 | 254.048069 | 909007000 |
| 50.000 | 4.39882588387 | 9419899 | 703.124616861 | 2516228387 |
| 100.000 | 8.94521903992 | 19373869 | 2794.91764808 | 10016471453 |

**Laiko trukmė pagal elementų kiekį**

Mėlyna – masyvas. Orandžinė - sąrašas

**Rūšiavimo algoritmas „Bucket sort“**

Šis rikiavimo metodas remiasi „Skaldyk ir valdyk“ paradigma. Metodas – rekursinis. Algoritmo viekimo tvarka:

\*Duomenys yra išskaldomi pagal reikšmes į atskiras kategorijas

\*Įterpiant reikšmę į kategoriją, ši reikšmė yra iš karto įterpiama taip, jog seka būtų nuosekli

\*Kai visos reikšmės yra išbarstytos į kategorijas, pradedamas suliejimas

\*Suliejimo metu nuosekliai imamos reikšmės nuo žemiausios kategorijos ir palaipsniui imamos vis kitos reikšmės iki pačios aukščiausios reikšmės

\*Taip ištraukus reikšmes turime surikiuotą sąrašą

Teoriškai šio algoritmo sudėtingumas yra:

n – rikiuojamų elementų kiekis. K – kategorijų skaičius

**Rūšiavimas „Bucket sort“ masyve**

Dešinėje pažymėta, kiek kartų vykdomos eilutės. Skaičiavimo algoritme skaičiuosime tik tas eilutes, kurios yra susijusios su algoritmu.

**L1\_3\_1.py**

import os

import struct

import time

const = 0

def bucketSort(f, length, bucket\_files, result\_length):

global const 1

for j in xrange(0, length, 4): n

variable = imti(f, j) Timti

range = variable / 1000000000 1

const += 2 1

if bucket\_files.has\_key(range): 1

sort\_insertion(bucket\_files[range], result\_length[range], imti(f, j), imti\_b(f, j)) Tsort\_intertion

result\_length[range] += 4 1

const += 2 1

pos = 0 1

dict\_list = sorted(bucket\_files) 1

const += 2 1

for i in xrange(0, len(dict\_list)): n

for j in xrange(0, result\_length[dict\_list[i]], 4): n

val = imti\_b(bucket\_files[dict\_list[i]], j) Timti\_b

deti(f, pos, val) Tdeti

pos += 4 1

const += 3 1

def rik(failo\_pav, rez\_failo\_pav="Bucket"):

global const 1

f = open(failo\_pav, 'r+b') 1

bucket\_files = {} 1

result\_length = {} 1

for i in xrange(-10, 10): 21

open(rez\_failo\_pav+"\_"+str(i), 'w').close() 1

w = open(rez\_failo\_pav+"\_"+str(i), 'r+b') 1

w.truncate() 1

new\_category = {i: w} 1

new\_length = {i: 0} 1

bucket\_files.update(new\_category) 1

result\_length.update(new\_length) 1

const += 7 1

length = os.path.getsize(failo\_pav) 1

*#failo\_isvedimas(f, length)*

time1 = time.time() 1

bucketSort(f, length, bucket\_files, result\_length) TbucketSort

time2 = time.time() 1

skirtumas = time2 - time1 1

*#print "-----ISRIKIUOTAS-----"*

*#failo\_isvedimas(f, length)*

print "Bendras uztruktas laikas sekundemis: %(time)s" %{'time': skirtumas} 1

const += 12 1

print "Eiluciu kiekis: %(const)s" %{'const': const} 1

def script(parent): #naudojamas testų automatizavimui

global const

for x in xrange(1, 18):

const = 0

file\_name = "T%(test)s" %{'test': x}

print file\_name

rik(parent+file\_name)

def imti(f, i):

global const 1

f.seek(i) 1

var = struct.unpack('i', f.read(4))[0] 1

const += 3 1

return var 1

def imti\_b(f, i):

global const 1

f.seek(i) 1

var = f.read(4) 1

const += 3 1

return var 1

def deti(f, i, r):

global const 1

f.seek(i) 1

f.write(r) 1

const += 2 1

def failo\_isvedimas(f, length):

global const 1

for j in xrange(0, length, 4): n

print imti(f, j) 1

const += 1 1

def sort\_insertion(f, length, value, value\_b):

global const 1

if length == 0: 1

deti(f, 0, value\_b) Tdeti

const += 1 1

else: 1

j = 0 1

const += 1 1

while j < length and imti(f, j) < value: n

j += 4 1

const += 1 1

value\_to\_write = value\_b 1

value\_to\_take = imti\_b(f, j) Timti\_b

const += 2 1

for x in xrange(j, length + 4, 4): n

deti(f, x, value\_to\_write) Tdeti

value\_to\_write = value\_to\_take 1

const += 2 1

try: 1

value\_to\_take = imti\_b(f, x+4) Timti\_b

const += 1 1

except: 1

const += 1 1

pass 1

T(n) = 40 + Tbucket\_sort(n) = 5n + 60 + 40 = 5n + 100

Tbucketsort(n) = 1 + n + Timti + 3 + Tsort\_insertion + 5 + 2n + Timti\_b + Tdeti + 2 = 3n + 11 + 5 + 5 + 4 + 2n + 35 = 5n + 60

Timti(n) = 5

Timti\_b(n) = 5

Tdeti(n) = 4

Tsort\_insertion(n) = 2 + Tdeti + 4 + n + 3 + Timti\_b + 1 + n + Tdeti + 3 + Timti\_b + 4 = 2n + 17 + 10 + 8 = 2n + 35

Galutinis T(n) = O(n \* k), k – kategorijų skaičius

**Rūšiavimas „Bucket sort“ sąraše**

**L1\_3\_2.py**

import os

import struct

import time

const = 0

def bucketSort(f, length, bucket\_files, result\_length):

global const 1

for j in xrange(0, length, 4): n

variable = imti\_list(f, j) Timti\_list

range = variable / 1000000000 1

const += 2 1

if bucket\_files.has\_key(range): 1

sort\_insertion(bucket\_files[range], result\_length[range], imti\_list(f, j), imti\_list\_b(f, j)) Tsort\_insertion

result\_length[range] += 4 1

const += 2 1

pos = 0 1

dict\_list = sorted(bucket\_files) 1

const += 2 1

for i in xrange(0, len(dict\_list)): n

for j in xrange(0, result\_length[dict\_list[i]], 4): n

val = imti\_list\_b(bucket\_files[dict\_list[i]], j) Timti\_list\_b

deti\_list(f, pos, val) Tdeti\_list

pos += 4 1

const += 3 1

def rik(failo\_pav, rez\_failo\_pav="Bucket"):

global const 1

f = open(failo\_pav, 'r+b') 1

bucket\_files = {} 1

result\_length = {} 1

for i in xrange(-10, 10): 21

open(rez\_failo\_pav+"\_"+str(i), 'w').close() 1

w = open(rez\_failo\_pav+"\_"+str(i), 'r+b') 1

w.truncate() 1

new\_category = {i: w} 1

new\_length = {i: 0} 1

bucket\_files.update(new\_category) 1

result\_length.update(new\_length) 1

const += 7 1

length = os.path.getsize(failo\_pav) 1

*#failo\_isvedimas\_list(f)*

time1 = time.time() 1

bucketSort(f, length, bucket\_files, result\_length) Tbucket\_sort

time2 = time.time() 1

skirtumas = time2 - time1 1

*#print "-----ISRIKIUOTAS-----"*

*#failo\_isvedimas\_list(f)*

print "Bendras uztruktas laikas sekundemis: %(time)s" %{'time': skirtumas} 1

const += 12 1

print "Eiluciu kiekis: %(const)s" %{'const': const} 1

def script(parent): #naudojamas testu automatizavimui

global const

for x in xrange(1, 18):

const = 0

file\_name = "T%(test)s" %{'test': x}

print file\_name

rik(parent+file\_name)

def pereiti(f, j):

global const 1

i = f.tell() 1

const += 1 1

if i < j: 1

for x in xrange(i+4, j+4, 4): n

f.seek(x) 1

const += 1 1

if i > j: 1

for x in xrange(i-4, j-4, -4): n

f.seek(x) 1

const += 1 1

if i == j: 1

pass 1

const += 1 1

def imti\_list(f, j):

global const 1

pereiti(f, j) Tpereiti

val = f.read(4) 1

const += 2 1

if val == '': 1

const += 1 1

return '' 1

else: 1

const += 1 1

return struct.unpack('i', val)[0] 1

def imti\_list\_b(f, j):

global const 1

pereiti(f, j) Tpereiti

const += 2 1

return f.read(4) 1

def deti\_list(f, j, r):

global const 1

pereiti(f, j) Tpereiti

f.write(r) 1

const += 2 1

def failo\_isvedimas\_list(f):

global const 1

value = 0 1

f.seek(0) 1

offset = 0 1

const += 3 1

while value != '': 1

value = imti\_list(f, offset) Timti\_list

const += 1 1

if value == '': 1

const += 1 1

break 1

print (value) 1

offset += 4 1

const += 2 1

def sort\_insertion(f, length, value, value\_b):

global const 1

if length == 0: 1

deti\_list(f, 0, value\_b) Tdeti\_list

const += 1 1

else: 1

j = 0 1

const += 1 1

while j < length and imti\_list(f, j) < value: n

j += 4 1

const += 1 1

value\_to\_write = value\_b 1

value\_to\_take = imti\_list\_b(f, j) Timti\_list\_b

const += 2 1

for x in xrange(j, length + 4, 4): n

deti\_list(f, x, value\_to\_write) Tdeti\_list

value\_to\_write = value\_to\_take 1

const += 2 1

try: 1

value\_to\_take = imti\_list\_b(f, x+4) Timti\_list\_b

const += 1 1

except: 1

const += 1 1

pass 1

T(n) = 40 + Tbucket\_sort(n) = 40 + 139 + 19n = 19n + 179

Tbucketsort(n) = 1 + n + Timti\_list + 3 + Tsort\_insertion + 5 + 2n + Timti\_list\_b + Tdeti\_list + 2 = 1 + n + 2n + 21 + 3 + 10n + 77 + 5 + 2n + 2n + 15 + 2n + 15 + 2 = 19n + 139

Tpereiti(n) = 1+1+1+1+n+1+1+1+n+1+1+1+1+1= 2n + 12

Timti\_list(n) = 1+Tpereiti(n)+1+1+1+1+1+1+1+1= 2n + 12 + 9 = 2n + 21

Timti\_list\_b(n) = 1+Tpereiti(n)+1+1 = 2n + 12 + 3 = 2n + 15

Tdeti\_list(n) = 1+Tpereiti(n)+1+1 = 2n + 12 + 3 = 2n + 15

Tsort\_insertion(n) = 2 + Tdeti\_list + 4 + n + 3 + Timti\_list\_b + 1 + n + Tdeti\_list + 3 + Timti\_list\_b + 4 = 2 + 2n + 15 + 4 + n + 3 + 2n + 15 + 1 + n + 2n + 15 + 3 + 2n + 15 + 4 = 10n + 77

Galutinis T(n) = O(n \* k), k – kategorijų skaičius

**Algoritmo ekspermentinis tyrimas**

Algoritmas testuojamas rikiuojant sveikuosius skaičius. Buvo sugeneruota daug atsitiktinių skaičių, nurodant failo dydį.  
Rikiuojant buvo skaičiuojama rikiavimo trukmė, palyginamas rikiavimas masyve ir sąraše. Duomenų imtis 200 iki 100.000.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Elementų sk. | Masyvas | | Sąrašas | |
| Trukmė (s) | Operacijų sk. | Trukmė (s) | Operacijų sk. |
| 200 | 0.0172638893127 | 34486 | 0.0241098403931 | 56096 |
| 400 | 0.0555970668793 | 118993 | 0.078996181488 | 195951 |
| 600 | 0.11394906044 | 255736 | 0.168732881546 | 422188 |
| 800 | 0.199162006378 | 455735 | 0.30180811882 | 754576 |
| 1.000 | 0.313737154007 | 711695 | 0.465600967407 | 1178508 |
| 1.200 | 0.456677913666 | 1001335 | 0.653183937073 | 1662029 |
| 1.400 | 0.576724052429 | 1329738 | 0.8708589077 | 2209305 |
| 1.600 | 0.760828018188 | 1719055 | 1.10874414444 | 2857383 |
| 1.800 | 0.996230125427 | 2226430 | 1.44347405434 | 3697037 |
| 2.000 | 1.24052000046 | 2681600 | 1.78206801414 | 4455230 |
| 3.000 | 2.6599919796 | 5973089 | 3.89628505707 | 9941081 |
| 5.000 | 7.33256602287 | 16575953 | 10.7687971592 | 27603363 |
| 10.000 | 29.5510778427 | 66196647 | 43.6714830399 | 110270250 |
| 20.000 | 117.724260092 | 263500943 | 174.706534863 | 439015347 |
| 30.000 | 265.602306843 | 593854711 | 394.671413183 | 989608952 |
| 50.000 | 734.482411146 | 1643497575 | 1098.88654399 | 2738745233 |
| 100.000 | 2956.100564 | 6588102715 | 4513.14706802 | 10979540748 |

**Laiko trukmė pagal elementų kiekį**

Mėlyna – masyvas. Orandžinė - sąrašas