Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут”

Кафедра АСОІУ

**ЗВІТ**

про виконання комп’ютерного практикуму № 2

з дисципліни

“Теорія алгоритмів”

|  |  |  |
| --- | --- | --- |
| Прийняв: |  | Виконав: |
| Халус Олена Андріївна |  | студент 2-го курсу  гр. ІП-51 ФІОТ  Зарічковий Олександр Анатолійович |

Київ – 2017

**ЗМІСТ:**

[1 Код програми 3](#_Toc476750393)

[2 Графіки Складностей алгоритмів 23](#_Toc476750394)

[3 Висновок 24](#_Toc476750395)

# Код програми

# -\*- coding: utf-8 -\*-

"""

Лабораторна робота №2 з курсу "Теорія алгоритмів"

Спеціальність: Інформаційні управляючі системи та технології

МЕТА:

Дослідити поведінку алгоритмів сортування методами включення та злиття.

ОПИС РОБОТИ:

В даній роботі досліджуються два методи сортування - метод сортування

включенням (insertion sort) та метод сортування злиття (merge sort).

Відомо, що час роботи алгоритму сортування включенням складає О(n^2), тоді

як час для алгоритму сортування злиттям - O(n\*lg(n)). Це означає, що

метод злиття працює швидше ніж метод включення для достатньо великих

розмірностей вхідних даних. Для вхідних даних малої розмірності це

може бути не так. Адже, в асимптотичних позначеннях нехтують константами

та меншими членами функцій, які, насправді, можуть мати значний вплив при

невеликих розмірностях входу алгоритму.

В роботі пропонується визначити розмір вхідних даних, при яких метод

сортування включенням працює швидше, за метод сортування злиттям.

На основі даних (для яких входів краще використовувати метод включення,

аніж метод злиття) можна створити, так званий, гібридний метод сортування.

Ідея його полягає в тому, щоб використовувати як основний метод

злиття, проте, коли при рекурсивних викликах процедури сортування для

масивів меншої розмірності ця розмірність досягає певної наперед визначеної

величини, то для таких підзадач використовувати метод сортування включенням.

ЗАВДАННЯ:

1) Реалізувати алгоритм сортування методом злиття. Для цього написати

функцію merge на основі наведеного псевдокоду (див. коментарі до функції).

2) Знайти розмірність вхідних даних, при яких сортування методом включення

працює швидше за метод злиття. Для цього провести експерименти, підібравши

значення параметрів у функції compare\_ins\_and\_merge. Відповідь на

поставлене питання можна знайти шляхом аналізу графіку порівнянь роботи

двох алгоритмів (див. виклик фукнції plot\_data у кінці функції

compare\_ins\_and\_merge).

Вказати значення розмірності вхідних даних, після яких метод злиття починає

працювати швидше за метод включення.

ВІДПОВІДЬ:

3.1) (На додаткові 2 бали) Реалізувати гібридний алгоритм (функція

hybrid\_sort), який використовує як базовий метод злиття, але при певних

розмірностях підзадач переходить до використання методу включення (див.

функцію insertion\_sort).

Вказати значення розмірності підмасиву, для якого у гібридному методі буде

використовуватись метод включення.

ВІДПОВІДЬ:

3.2) Порівняти час роботи алгоритмів за методом злиття та гібридним методом.

Для цього використати функцію compare\_merge\_and\_hybrid. Зробити висновок

щодо отриманих результатів.

ДОКУМЕНТАЦІЯ:

Python v2.7.3 documentation:

- http://docs.python.org/2/

- http://oim.asu.kpi.ua/python/docs (дзеркало, доступне з кафедральних комп'ютерів)

Книга "Dive Into Python":

- http://www.diveintopython.net/

- http://oim.asu.kpi.ua/python/diveintopython/html/ (дзеркало, доступне з кафедральних комп'ютерів)

"""

import random

from plot\_data import plot\_data

from copy import deepcopy

import time

def generate\_data(n, gen\_type="random"):

"""

Функція генерації масивів для подальшого сортування.

Параметри:

n (int) - кількість елементів масиву

gen\_type (string) - тип згенерованих даних:

"best" - відсортований масив (мінімальний час для сортування)

"worst" - найгірший варіант для сортування

"random" - послідовність елементів генерується випадкова

(значення за замовчуванням)

Повертає:

Масив (list) довжиною n з елементами від 1 до n

"""

if gen\_type=="best":

a = [i+1 for i in range(n)]

return a

elif gen\_type=="worst":

a = [i+1 for i in reversed(range(n))]

return a

else:

a = [i+1 for i in range(n)]

random.shuffle(a)

return a

def insertion\_sort(seq, p, r):

"""

Алгоритм сортування методом включення.

Відсортовує підмасив seq[p..r-1].

Параметри:

seq - послідовність для сортування

p - початковий індекс підмасиву

r - кінцевий індекс підмасиву (границя не включається у підмасив)

"""

for i in xrange(p+1, r):

j = i-1

key = seq[i]

while (seq[j] > key) and (j >= p):

seq[j+1] = seq[j]

j -= 1

seq[j+1] = key

def merge(seq, p, q, r):

"""

Процедура злиття для методу сортування злиттям.

Зливає (з'єднує) дві вже відсортовані частини вхідного масиву seq -

ліва частина seq[p..q-1] та права seq[q..r-1]. По закінченню роботи

підмасив seq[p..r-1] містить елементи у відсортованому порядку.

Параметри:

seq - послідовність для сортування

p - початковий індекс підмасиву

q - індекс кінця першої половини масиву для злиття

r - індекс кінця другої половини масиву для злиття

ЗАВДАННЯ:

Вашим завданням є написання тіла функції, яка реалізує процедуру

злиття методу сортування злиттям, за наведеним нижче псевдокодом.

ПСЕВДОКОД:

n1 = q – p + 1

n2 = r – q

Створити масиви L[1..n1+1] та R[1..n2+1]

for i = 1 to n1

do L[i] = A[p+i-1]

for j = 1 to n2

do R[j] = A[q+j]

L[n1+1] = infinity

R[n2+1] = infinity

i = 1

j = 1

for k = p to r

do if L[i] <= R[j]

then A[k] = L[i]

i = i + 1

else A[k] = R[j]

j = j + 1

"""

# Тут повинен бути ваш код

L = seq[p:q]

R = seq[q:r]

L.append(float('inf'))

R.append(float('inf'))

i = 0

j = 0

for k in xrange(p, r):

if L[i] < R[j]:

seq[k] = L[i]

i += 1

else:

seq[k] = R[j]

j += 1

# Для перевірки можна вивести відсортований підмасив

#print seq[p:r]

def merge\_sort(seq, p, r):

"""

Алгоритм сортування злиттям.

Відсортовує підмасив seq[p..r-1].

Параметри:

seq - послідовність для сортування

p - початковий індекс підмасиву

r - кінцевий індекс підмасиву (границя не включається у підмасив)

"""

if r-p<=1:

return

q = (r+p) / 2

merge\_sort(seq, p, q)

merge\_sort(seq, q, r)

merge(seq, p, q, r)

def hybrid\_sort(seq, p, r):

"""

Гібридний алгоритм сортування, який об'єднує методи сортування злиттям та

включенням. Ідея полягає у використанні методу злиття, але при досягненні

певного розміру задачі використовувати метод включення (для задач малої

розмірності). Відсортовує підмасив seq[p..r-1].

Параметри:

seq - послідовність для сортування

p - початковий індекс підмасиву

r - кінцевий індекс підмасиву (границя не включається у підмасив)

"""

SIZE\_OF\_BASE = 80

if r - p <= SIZE\_OF\_BASE:

insertion\_sort(seq, p, r)

return

q = (r+p) / 2

hybrid\_sort(seq, p, q)

hybrid\_sort(seq, q, r)

merge(seq, p, q, r)

def test(f, data):

"""

Функція тестування різних алгоритмів сортування в рамках даної лабораторної

роботи. Підраховує час виконання одного сортування для масиву фіксованої

довжини.

Параметри:

f - функція сортування, яка сама на вхід приймає три параметри:

seq - масив для сортування

p - початковий індекс підмасиву

r - кінцевий індекс підмасиву

data - список масивів для сортування. Масив data містить декілька

тестових масивів і процедура test викликає тестовий алгоритм

для кожного з тествих підмасивів

Повертає:

час роботи алгоритму f на одному екземплярі (у секундах, тип float)

"""

repeats = len(data)

start = time.clock()

for i in xrange(repeats):

# тут можна вивести поточну вхідну послідовність для сортування

#print data[i]

f(data[i], 0, len(data[i]))

# тут можна вивести поточну відсортовану послідовність

#print data[i], '\n'

end = time.clock()

return (end-start)/repeats

def compare\_ins\_and\_merge():

"""

Процедура порівняння двох методів сортування: включенням та злиттям.

Порівння алгоритмів ґрунтуєься на дослідженні часу їх роботи (в сек). Для

цього використовується функція test.

Тестування проводиться на задачах різної розмірності: від n\_begin до n\_end

з кроком n\_step (значення цих параметрів встановлюються в середині процедури)

Для кожної розмірності генерується repeats екземплярів задачі. При чому

обидва алгоритми запускаються на одних і тих самих екземплярах задачі.

"""

# параметри для проведення експерименту

repeats = 100 # кількість запусків для однієї розмірності

n\_begin = 1 # початкова розмірність задачі

n\_end = 200 # кінцева розмірність задачі

n\_step = 2 # крок розмірності

types = ["random"]

data\_plot = {'random': {'insertion':{}, 'merge':{}}}

data\_plot\_2 = {'ratio': {'insertion/merge':{}}}

for n in xrange(n\_begin,n\_end+1,n\_step):

print "\nDATA SIZE: ", n

for gen\_type in types:

# згенерувати тестові набори даних розмірності n в кількості repeats

data = [generate\_data(n) for i in xrange(repeats)]

t\_insertion = test(insertion\_sort , deepcopy(data))

print "Insertion time for size", n, ":", t\_insertion

data\_plot[gen\_type]['insertion'][n] = t\_insertion

t\_merge = test(merge\_sort, deepcopy(data))

print "Merge time for size", n, ":", t\_merge

data\_plot[gen\_type]['merge'][n] = t\_merge

print "Ratio insertion/merge:", t\_insertion/t\_merge

data\_plot\_2['ratio']['insertion/merge'][n] = t\_insertion/t\_merge

# побудувати графіки швидкості роботи алгоритмів

plot\_data(data\_plot, logarithmic=False, oneplot=True, data\_2=data\_plot\_2)

def compare\_merge\_and\_hybrid():

"""

Процедура порівняння двох методів сортування: злиттям та гібридного,

який ґрунтується на методах включення та злиття.

Детальніше - див. функцію compare\_ins\_and\_merge()

"""

# параметри для проведення експерименту

repeats = 10 # кількість запусків для однієї розмірності

n\_begin = 100 # початкова розмірність задачі

n\_end = 5000 # кінцева розмірність задачі

n\_step = 100 # крок розмірності

types = ["random"]

data\_plot = {'random': {'merge':{}, 'hybrid':{}}}

data\_plot\_2 = {'ratio': {'merge/hybrid':{}}}

for n in xrange(n\_begin,n\_end+1,n\_step):

print "\nDATA SIZE: ", n

for gen\_type in types:

data = [generate\_data(n) for i in xrange(repeats)]

t\_merge = test(merge\_sort, deepcopy(data))

print "Merge time for size", n, ":", t\_merge

data\_plot[gen\_type]['merge'][n] = t\_merge

t\_hybrid = test(hybrid\_sort, deepcopy(data))

print "Hybrid time for size", n, ":", t\_hybrid

data\_plot[gen\_type]['hybrid'][n] = t\_hybrid

print "Ratio merge/hybrid:", t\_merge/t\_hybrid

data\_plot\_2['ratio']['merge/hybrid'][n] = t\_merge/t\_hybrid

# побудувати графіки швидкості роботи алгоритмів

plot\_data(data\_plot, logarithmic=False, oneplot=True, data\_2=data\_plot\_2)

# --------------------------------------------

def mergeSort\_impr(seq, p, r):

SIZE\_OF\_BASE = 80

for i in xrange(p, r, SIZE\_OF\_BASE):

insertion\_sort(seq, i, min(i + SIZE\_OF\_BASE, r))

blockSize = SIZE\_OF\_BASE

while blockSize <= r - p:

for k in xrange(p + blockSize, r, 2 \* blockSize):

merge(seq, k-blockSize, k, min(k + blockSize, r))

blockSize \*= 2

#print "SORTED: " + str(seq)

def compare\_merge\_impr\_and\_hybrid():

"""

Процедура порівняння двох методів сортування: злиттям та гібридного,

який ґрунтується на методах включення та злиття.

Детальніше - див. функцію compare\_ins\_and\_merge()

"""

# параметри для проведення експерименту

repeats = 5 # кількість запусків для однієї розмірності

n\_begin = 1000 # початкова розмірність задачі

n\_end = 10000 # кінцева розмірність задачі

n\_step = 300 # крок розмірності

types = ["random"]

data\_plot = {'random': {'merge\_impr':{}, 'hybrid':{}}}

data\_plot\_2 = {'ratio': {'merge\_impr/hybrid':{}}}

for n in xrange(n\_begin,n\_end+1,n\_step):

print "\nDATA SIZE: ", n

for gen\_type in types:

data = [generate\_data(n) for i in xrange(repeats)]

t\_merge = test(mergeSort\_impr, deepcopy(data))

print "Merge\_impr time for size", n, ":", t\_merge

data\_plot[gen\_type]['merge\_impr'][n] = t\_merge

t\_hybrid = test(hybrid\_sort, deepcopy(data))

print "Hybrid time for size", n, ":", t\_hybrid

data\_plot[gen\_type]['hybrid'][n] = t\_hybrid

print "Ratio merge\_impr/hybrid:", t\_merge/t\_hybrid

data\_plot\_2['ratio']['merge\_impr/hybrid'][n] = t\_merge/t\_hybrid

# побудувати графіки швидкості роботи алгоритмів

plot\_data(data\_plot, logarithmic=False, oneplot=True, data\_2=data\_plot\_2)

# ------------------------------------------------------------------------

compare\_merge\_impr\_and\_hybrid()

"""

Порівняння алгоритмів сортування за методом включенння та методом злиття.

"""

#compare\_ins\_and\_merge()

"""

Порівняння алгоритмів сортування за методом злиття та гібридного методу,

із використанням методів включення та злиття.

"""

#compare\_merge\_and\_hybrid()

# Приклад виконання програми

Приклад виконання програми наведений на рисунку 2.1:

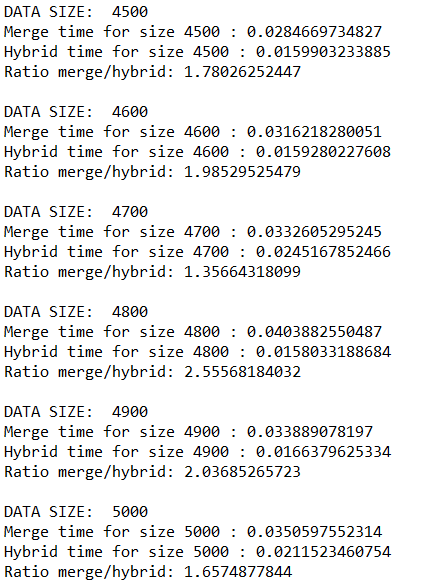


Рисунок 2.1 – Приклад виконання програми

# Графіки Складностей алгоритмів

Графік виконання фунції compare\_ins\_and\_merge наведений на   
рисунку 3.1:

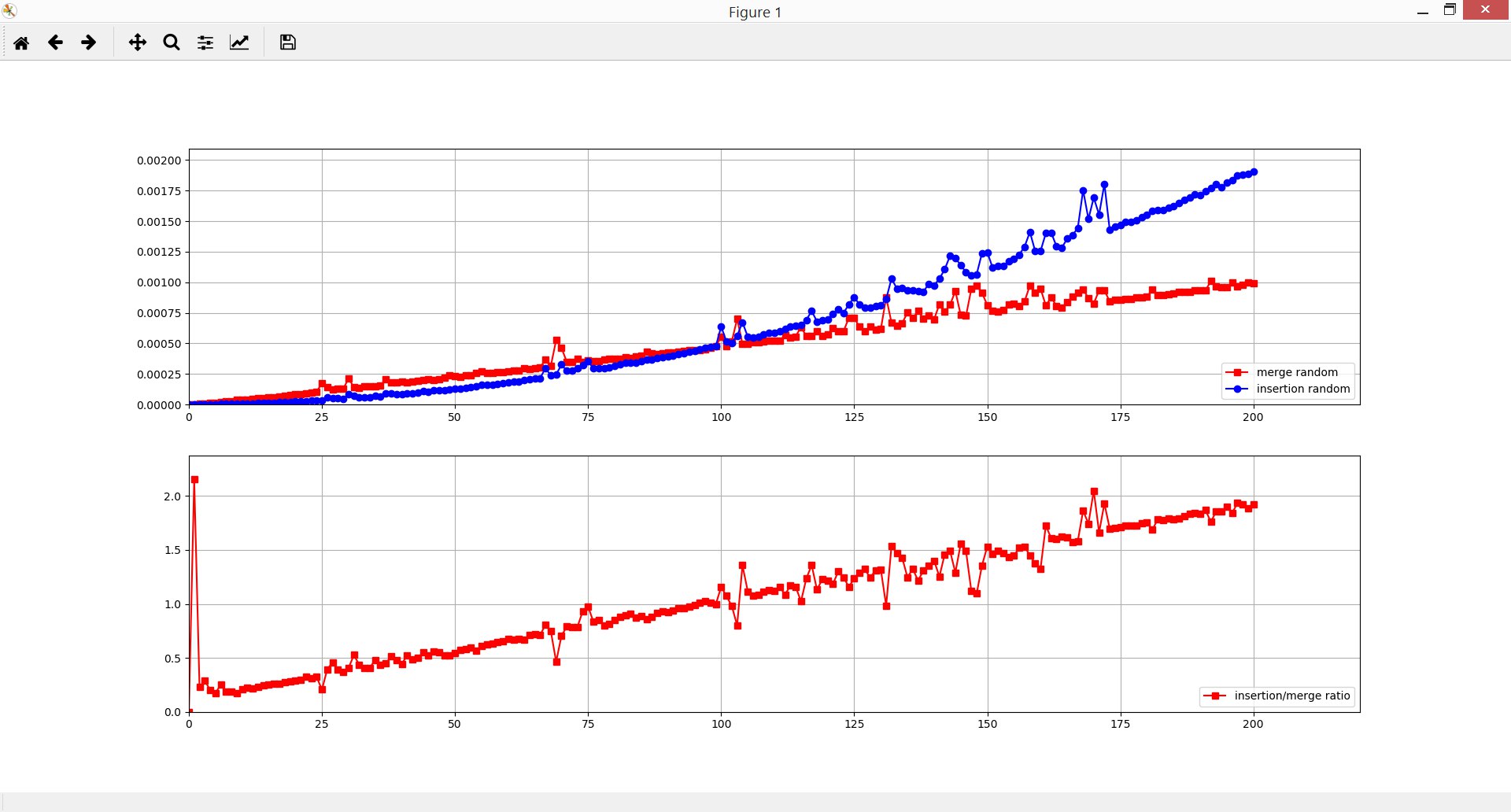


Рисунок 3.1 – Графік порівняння складності Insertion sort i Merge sort

Графік виконання фунції hybrid\_sort наведений на рисунку 3.2:

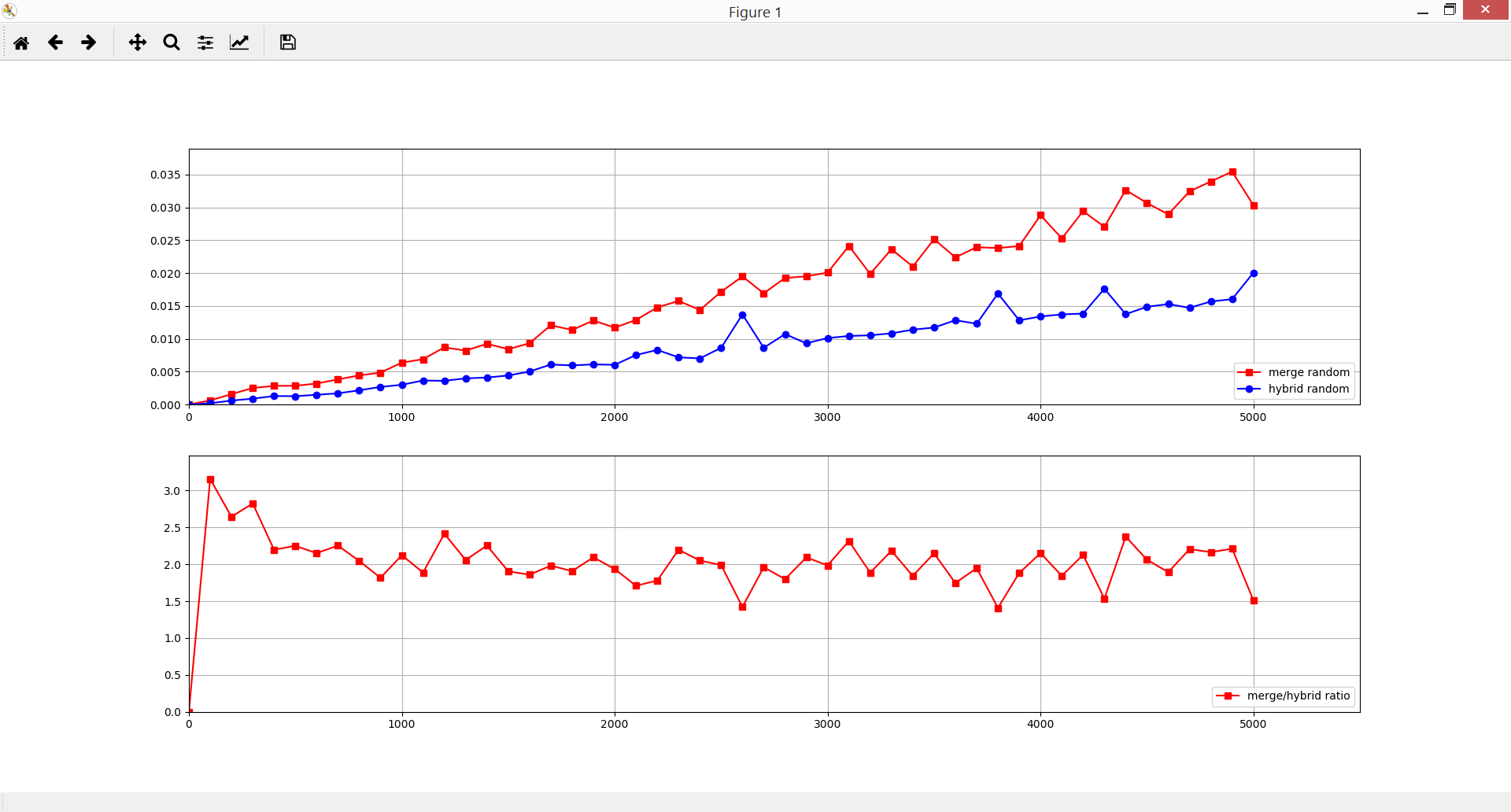


Рисунок 3.2 – Графік порівння складності Megre sort i Hybrid sort

# Висновок

В ході виконання лабораторної роботи було досліджено наступні алгоритми сортування — insertion\_sort, merge\_sort, hybrid\_sort. Вони були перевірені на наборах даних величиною в діапазоні від 1 до 5000 елементів, згенерованих випадково. За графіком роботи алгоритмів видно, що метод сортування злиттям має меншу асимптотичну складність, аніж метод сортування вставками для послідовностей, що складаються більше ніж з 95 елементів. Аналогічно гібридний метод сортування, що являє собою поєднання методів сортування злиттям та вставками, працює від 40% до 200% швидше за метод сортування злиттям. Різниця між алгоритмами найбільш помітна на великих наборах вхідних даних