МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ

КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

ЗВІТ

про виконання лабораторної роботи №1

з дисципліни «Проектування та аналіз алгоритмів»

Варіант №7

Виконала: студентка групи КА-06

Вергелюк Олександр Андрійович

Прийняв: Селін Ю.М.

Київ – 2021

**1. Завдання**

1. Розробити та реалізувати такі алгоритми сортування: сортування обміном із фіксацією наявності пересувань, Шелла та підрахунком.

2. Згенерувати масив розмірності 1000, 10000, 100000 елементів:

a. відсортований за зростанням;

b. з випадковими елементами (кількість генерувань = 1000);

c. відсортований за спаданням.

3. Відсортувати одержані масиви за зростанням, отримавши такі змінні:

a. кількість порівнянь;

b. кількість обмінів.

4. Порахувати для загального випадку:

a. Трудомісткість алгоритму

b. Оцініть час роботи алгоритмів.

**2. Лістинг програми**

import random  
  
  
class AnalysisArray:  
 arr = list()  
  
 def \_\_init\_\_(self):  
 self.arr = list()  
  
 def generate\_up(self, n):  
 self.arr = list(range(1, n+1))  
  
 def generate\_rand(self, n):  
 self.arr.clear()  
 for i in range(n):  
 self.arr.append(random.randint(1, n))  
  
 def generate\_down(self, n):  
 self.arr = list(range(n, 0, -1))  
  
 def bubble\_sort(self):  
 total\_swaps = 0  
 total\_compares = 0  
 for i in range(len(self.arr)-1):  
 swaps = 0  
 for j in range(len(self.arr)-i-1):  
 total\_compares += 1  
 if self.arr[j+1] < self.arr[j]:  
 self.arr[j+1], self.arr[j] = self.arr[j], self.arr[j+1]  
 swaps += 1  
 total\_swaps += 1  
 if not swaps:  
 break  
  
 return total\_compares, total\_swaps  
  
 def shell\_sort(self):  
 total\_swaps = 0  
 total\_compares = 0  
 d = len(self.arr) // 2  
 while d:  
 for i in range(d):  
 for j in range(i+d, len(self.arr), d):  
 k = j  
 total\_compares += 1  
 while k > d-1 and self.arr[k] < self.arr[k-d]:  
 self.arr[k], self.arr[k-d] = self.arr[k-d], self.arr[k]  
 total\_swaps += 1  
 k -= d  
 d = d // 2  
  
 return total\_compares, total\_swaps  
  
 def count\_sort(self):  
 counts = [0] \* (len(self.arr) + 1)  
 for el in self.arr:  
 counts[el] += 1  
  
 new\_arr = []  
  
 for i in range(len(counts)):  
 new\_arr.extend([i]\*counts[i])  
  
 self.arr = new\_arr  
  
 def bubble\_test(self):  
 print("-"\*10 + "Bubble sort test" + "-"\*10)  
 for n in [1000, 10000, 100000]:  
 print("Best result for", n, "element: ", end="")  
 self.generate\_up(n)  
 answ = self.bubble\_sort()  
 print("compares: ", answ[0], ", swaps: ", answ[1], sep="")  
  
 print("Average result for", n, "element: ", end="")  
 self.generate\_rand(n)  
 answ = self.bubble\_sort()  
 print("compares: ", answ[0], ", swaps: ", answ[1], sep="")  
  
 print("Worst result for", n, "element: ", end="")  
 self.generate\_down(n)  
 answ = self.bubble\_sort()  
 print("compares: ", answ[0], ", swaps: ", answ[1], sep="", end="\n\n")  
  
 def shell\_test(self):  
 print("-" \* 10 + "Shell's sort test" + "-" \* 10)  
 for n in [1000, 10000, 100000]:  
 print("Best result for", n, "element: ", end="")  
 self.generate\_up(n)  
 answ = self.shell\_sort()  
 print("compares: ", answ[0], ", swaps: ", answ[1], sep="")  
  
 print("Average result for", n, "element: ", end="")  
 self.generate\_rand(n)  
 answ = self.shell\_sort()  
 print("compares: ", answ[0], ", swaps: ", answ[1], sep="")  
  
 print("Worst result for", n, "element: ", end="")  
 self.generate\_down(n)  
 answ = self.shell\_sort()  
 print("compares: ", answ[0], ", swaps: ", answ[1], sep="", end="\n\n")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 test = AnalysisArray()  
 test.bubble\_test()  
 test.shell\_test()

**3. Результати експерименту**

3.1 Сортування обміном з фіксацією наявності пересувань.

Сортування обміном з фіксацією наявності пересувань містить в собі два цикли: перший від початкового до n-1-го елемента масиву, другий – від початкового до n-(i+1)-го елемента масиву, де і – ітератор першого циклу. У внутрішньому циклі порівнюються два сусідні елементи, і якщо вони неправильно впорядковані, то обмінюються місцями. За кожну ітерацію внутрішнього циклу на своє місце виводиться один елемент. Перший цикл має всього n ітерацій. На кожній ітерації другий цикл матиме кількість проходжень на 1 менше, ніж на попередній ітерації. Отже маємо арифметичну прогресію, перший член якої дорівнює n-1, останній дорівнює 0, всього елементів n, тоді маємо формулу:

З цієї формули видно, що в найгіршому випадку асимптотична складність алгоритму є О(n2). В середньому випадку асимптотична складність наближається до найгіршого випадку і також є О(n2).

А в найкращому випадку завдяки фіксації наявності обмінів, після першого виконання першого циклу індикатор вказує, що не було жодного обміну, отже жоден елемент не помінявся місцем із сусіднім. Відповідно усі елементи попарно правильно впорядковані, тому і вся послідовність відсортована вірно. Очевидно, що в найкращому випадку асимптотична складність виконання алгоритму сортування є О(n). Результати роботи програми наведено у таблиці 1.

Таблиця 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | Метод обміну з фіксацією наявності пересувань | | | | | |
| Кількість порівнянь | | | Кількість копіювань | | |
| Теор. | Експерим. | Відношення | Теор. | Експерим. | Відношення |
|  | Найкращий випадок | | | | | |
| 1000 | 999 | 999 | 1 | 0 | 0 | 1 |
| 10000 | 9999 | 9999 | 1 | 0 | 0 | 1 |
| 100000 | 99999 | 99999 | 1 | 0 | 0 | 1 |
|  | Середній випадок | | | | | |
| 1000 | 499500 | 497670 | 0.996 | 249750 | 247541 | 0.991 |
| 10000 | 49995000 | 49994864 | 0.999 | 24997500 | 25272984 | 0.989 |
| 100000 | 4999950000 | 4989950127 | 0.998 | 2499975000 | 2500656544 | 0.999 |
|  | Найгірший випадок | | | | | |
| 1000 | 499500 | 499500 | 1 | 499500 | 499500 | 1 |
| 10000 | 49995000 | 49995000 | 1 | 49995000 | 49995000 | 1 |
| 100000 | 4999950000 | 4999950000 | 1 | 4999950000 | 4999950000 | 1 |

3.2 Сортування Шелла

Сортування Шелла – алгоритм сортування, що є вдосконаленим варіантом сортування вставками. Ідея методу Шелла полягає в порівнянні елементів, що стоять не тільки поруч, але і на певній відстані один від одного.

При сортуванні Шелла спочатку порівнюються і сортуються між собою значення, що стоять один від одного на деякій відстані *d*. Після цього процедура повторюється для деяких менших значень *d*, а завершується сортування Шелла упорядкуванням елементів при *d* = 1 (тобто звичайним сортуванням вставками). Ефективність сортування Шелла в певних випадках забезпечується тим, що елементи «швидше» встають на свої місця.

Ефективність сортування Шелла залежить від обраних значень *d*. В класичному алгоритмі, запропонованому Шеллом, спочатку *d =*  і кожного разу зменшується вдвічі, доки не стане 1. При такій реалізації в усіх випадках асимптотична складність алгоритму є O(n). Результати роботи програми наведено у таблиці 2.

Таблиця 2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | Метод Шелла | | | | | |
| Кількість порівнянь | | | Кількість копіювань | | |
| Теор. | Експерим. | Відношення | Теор. | Експерим. | Відношення |
|  | Найкращий випадок | | | | | |
| 1000 | 8000 | 8006 | 0.999 | 0 | 0 | 1 |
| 10000 | 120000 | 120005 | 0.999 | 0 | 0 | 1 |
| 100000 | 1500000 | 1500006 | 0.999 | 0 | 0 | 1 |
|  | Середній випадок | | | | | |
| 1000 | 8000 | 8006 | 0.999 | 5000 | 5121 | 0.976 |
| 10000 | 120000 | 120005 | 0.999 | 90000 | 86802 | 0.964 |
| 100000 | 1500000 | 1500006 | 0.999 | 1200000 | 1157995 | 0.965 |
|  | Найгірший випадок | | | | | |
| 1000 | 8000 | 8006 | 0.999 | 4500 | 4468 | 0.993 |
| 10000 | 120000 | 120005 | 0.999 | 65000 | 62280 | 0.958 |
| 100000 | 1500000 | 1500006 | 0.999 | 765000 | 763515 | 0.998 |

3.3 Сортування підрахунком

Цей вид сортувань напряму не порівнює і не робить ніяких обмінів. Алгоритм за один прохід підраховує кількість елементів кожного значення, а потім створює новий масив, в який записує необхідну кількість кожного з елементів, підміняючи таким чином початковий масив. Сортування підрахунком базується на використанні деяких властивостей елементів, тому для нього необхідно, щоб заздалегідь було відомо множину можливих елементів. Гарним набором для сортування підрахунком є цілі числа в певному інтервалі.

Асимптотична швидкість алгоритму - O(N + K), де К – кількість можливих значень у сортованому масиві, так же як і трудомісткість алгоритму, тому що в процесі сортування використовується додаткове місце, що залежить від кількості елементів в масиві і їх різноманітності. Результати роботи програми наведено у таблиці 3.

Таблиця 3

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| N | Метод Шелла | | | | | |
| Кількість порівнянь | | | Кількість копіювань | | |
| Теор. | Експерим. | Відношення | Теор. | Експерим. | Відношення |
|  | Найкращий випадок | | | | | |
| 1000 | 8000 | 8006 | 0.999 | 0 | 0 | 1 |
| 10000 | 120000 | 120005 | 0.999 | 0 | 0 | 1 |
| 100000 | 1500000 | 1500006 | 0.999 | 0 | 0 | 1 |
|  | Середній випадок | | | | | |
| 1000 | 8000 | 8006 | 0.999 | 5000 | 5121 | 0.976 |
| 10000 | 120000 | 120005 | 0.999 | 90000 | 86802 | 0.964 |
| 100000 | 1500000 | 1500006 | 0.999 | 1200000 | 1157995 | 0.965 |
|  | Найгірший випадок | | | | | |
| 1000 | 8000 | 8006 | 0.999 | 4500 | 4468 | 0.993 |
| 10000 | 120000 | 120005 | 0.999 | 65000 | 62280 | 0.958 |
| 100000 | 1500000 | 1500006 | 0.999 | 750000 | 763515 | 0.982 |

**4. Висновки**

У ході лабораторної роботи було розглянуто три алгоритми сортування: обміном з фіксацією наявності пересувань, Шелла та підрахунком. Під час виконання роботи було досліджено їх асимптотичну складність: для сортування обміном O(n2) в середньому та найгіршому випадках; для сортування Шелла O(n).; для сортування за розрядами O(N + K), де М – кількість можливих елементів у сортованому масиві.

Кількість порівнянь та обмінів в результаті виконання програми гарно узгоджується із теоретичними значеннями. Похибка становить 1% і зумовлена особливістю реалізації алгоритму мовою Python.