**Лабораторна робота № 2**

**МЕТОДИ СОРТУВАННЯ МАСИВІВ**

**ФБ-22 Дажук Павло**

**Варіант 5**

**Мета роботи**: познайомитися з роботою поширених методів сортування, з критеріями та методикою їх порівняння.

**Сортування методом вибору**

import random  
import time  
  
  
# Sort by selection with number of comparisons, moves and time  
def selection\_sort(array):  
 n = len(array)  
 comparisons = 0  
 moves = 0  
 start\_time = time.time()  
  
 for i in range(n - 1):  
 min\_index = i  
 for j in range(i+1, n):  
 comparisons += 1  
 if array[j] < array[min\_index]:  
 min\_index = j  
  
 moves += 1  
 array[i], array[min\_index] = array[min\_index], array[i]  
  
 end\_time = time.time()  
 time\_taken = round((end\_time - start\_time) \* 1000, 3)  
  
 return comparisons, moves, time\_taken  
  
  
# Generation of arrays with 100, 1000 and 10000 random elements  
def generate\_random\_array():  
 return ([random.randint(1, 100) for \_ in range(100)],  
 [random.randint(1, 100) for \_ in range(1000)],  
 [random.randint(1, 100) for \_ in range(10000)])  
  
  
# Array output  
def display\_array(array):  
 elements\_per\_line = len(array) / 5  
 for i, element in enumerate(array):  
 print(element, end=", ")  
 if (i + 1) % elements\_per\_line == 0:  
 print()  
 print()  
  
  
# The main function of outputting all data  
def main():  
 arrays = generate\_random\_array()  
 for i in range(3):  
 print(f"Array {10\*\*(i+2)}", "\n")  
 print("Unsorted array:")  
 display\_array(arrays[i])  
  
 comparisons, moves, time\_taken = selection\_sort(arrays[i])  
 print("Sorted array:")  
 display\_array(arrays[i])  
  
 print(f"Comparisons: {comparisons}")  
 print(f"Moves: {moves}")  
 print(f"Time taken: {time\_taken}ms", "\n")  
 print("-"\*200)  
  
  
# Calculation of average values of comparisons, moves and time for sorting  
def average\_stats(n):  
 total\_comparisons, total\_moves, total\_time\_taken = 0, 0, 0  
 num\_iterations = 100  
 print("Iterations:", num\_iterations, "\n")  
 for i in range(num\_iterations):  
 array = [random.randint(1, 100) for \_ in range(n)]  
 comparisons, moves, time\_taken = selection\_sort(array)  
 total\_comparisons += comparisons  
 total\_moves += moves  
 total\_time\_taken += time\_taken  
 return total\_comparisons/num\_iterations, total\_moves/num\_iterations, total\_time\_taken/num\_iterations  
  
  
# Output of average values of comparisons, moves and time for arrays  
def display\_average\_stats():  
 for i in range(3):  
 print(f"Array {10\*\*(i+2)}", "\n")  
 average\_comparisons, average\_moves, average\_time\_taken = average\_stats(10\*\*(i+2))  
 print(f"Average comparisons: {average\_comparisons}")  
 print(f"Average moves: {average\_moves}")  
 print(f"Average time taken: {average\_time\_taken}ms", "\n")  
 print("-"\*200)  
  
main()  
display\_average\_stats()

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

**Час рахував окремо, оскільки обрахунок кількості переміщень і порівнянь сильно впливає на час.**

A screenshot of a computer

Description automatically generated

**ШВИДКЕ СОРТУВАННЯ**

import random  
import time  
  
comparisons = 0  
moves = 0  
  
  
def stats\_reset():  
 global comparisons, moves  
 comparisons = 0  
 moves = 0  
  
  
# Quick sort algorithm  
def quicksort(array):  
 global comparisons, moves  
 count = 0  
 if len(array) <= 1:  
 comparisons += 1  
 return array  
 for i in range(len(array) - 1):  
 comparisons += 1  
 if array[i] < array[-1]:  
 array[i], array[count] = array[count], array[i]  
 moves += 1  
 count += 1  
  
 array[count], array[-1] = array[-1], array[count]  
 moves += 1  
  
 return quicksort(array[0:count]) + [array[count]] + quicksort(array[count+1:])  
  
  
# Generation of arrays with 100, 1000 and 10000 random elements  
def generate\_random\_array():  
 return ([random.randint(1, 100) for \_ in range(100)],  
 [random.randint(1, 100) for \_ in range(1000)],  
 [random.randint(1, 100) for \_ in range(10000)])  
  
  
# Array output  
def display\_array(array):  
 elements\_per\_line = len(array) / 5  
 for i, element in enumerate(array):  
 print(element, end=", ")  
 if (i + 1) % elements\_per\_line == 0:  
 print()  
 print()  
  
  
# The main function of outputting all data  
def main():  
 arrays = generate\_random\_array()  
 for i in range(3):  
 print(f"Array {10\*\*(i+2)}", "\n")  
 print("Unsorted array:")  
 display\_array(arrays[i])  
  
 stats\_reset()  
 start\_time = time.time()  
 sorted\_array = quicksort(arrays[i])  
 time\_taken = (time.time() - start\_time) \* 1000  
 print("Sorted array:")  
 display\_array(sorted\_array)  
  
 print(f"Comparisons: {comparisons}")  
 print(f"Moves: {moves}")  
 print(f"Time taken: {time\_taken} ms", "\n")  
 print("-"\*200)  
  
  
# Calculation of average values of comparisons, moves and time for sorting  
def average\_stats(n):  
 total\_comparisons, total\_moves, total\_time\_taken = 0, 0, 0  
 num\_iterations = 100  
 print("Iterations:", num\_iterations, "\n")  
 for i in range(num\_iterations):  
 array = [random.randint(1, 100) for \_ in range(n)]  
 stats\_reset()  
 start\_time = time.time()  
 quicksort(array)  
 time\_taken = (time.time() - start\_time) \* 1000  
 total\_comparisons += comparisons  
 total\_moves += moves  
 total\_time\_taken += time\_taken  
 return total\_comparisons/num\_iterations, total\_moves/num\_iterations, total\_time\_taken/num\_iterations  
  
  
# Output of average values of comparisons, moves and time for arrays  
def display\_average\_stats():  
 for i in range(3):  
 print(f"Array {10\*\*(i+2)}", "\n")  
 average\_comparisons, average\_moves, average\_time\_taken = average\_stats(10\*\*(i+2))  
 print(f"Average comparisons: {average\_comparisons}")  
 print(f"Average moves: {average\_moves}")  
 print(f"Average time taken: {average\_time\_taken} ms", "\n")  
 print("-"\*200)  
  
  
main()  
display\_average\_stats()

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Алгоритм реалізації Швидкого Сортування:**

1. **Базовий випадок:**

if len(array) <= 1: comparisons += 1 return array

Якщо довжина масиву менше або дорівнює 1, то він вже відсортований, і функція повертає масив без змін.

1. **Ініціалізація лічильника переміщень:**

count = 0

Лічильник **count** використовується для визначення позиції останнього елемента, який менший за опорний елемент, а також для визначення позицій елементів, які ми будемо міняти місцями з елементами, які менші за опорний, тобто всі елементи менші за опорний переміщуємо в ліву частину масива.

1. **Ітерація через елементи масиву:**

for i in range(len(array) - 1): if array[i] < array[-1]:

array[i], array[count] = array[count], array[i] count += 1

Проходиться по всім елементам, і якщо елемент менший за опорний (останній елемент), то він міняється місцями з елементом на позиції **count**, і **count** збільшується на одиницю.

1. **Обмін опорного елемента:**

array[count], array[-1] = array[-1], array[count]

Опорний елемент міняється місцем з елементом на позиції **count**, розташовуючи всі менші елементи ліворуч від опорного, а всі більші - праворуч.

1. **Рекурсивний виклик для лівої та правої частин:**

return quicksort(array[0:count]) + [array[count]] + quicksort(array[count+1:])

Функція рекурсивно викликає себе для лівої частини (елементи менше опорного) і правої частини (елементи більше опорного). Результати об'єднуються, вставляючи опорний елемент між відсортованими лівою та правою частинами.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Сортування методом вибору** | | | | | **ШВИДКЕ СОРТУВАННЯ** | | | | |
| **N** | **К-ть копіювань**  **(M)** | | **К-ть порівнянь**  **(C)** | | **Час мс. (T)** | **К-ть копіювань**  **(M)** | | **К-ть порівнянь**  **(C)** | | **Час мс. (T)** |
| **Теорет.** | **Експерим.** | **Теорет.** | **Експерим.** | **Теорет.** | **Експерим.** | **Теорет.** | **Експерим.** |
| 100 | 50 | 99 | 4950 | 4950 | 0.12 | 664 | 366 | 664 | 717 | 0.05 |
| 1000 | 500 | 999 | 499500 | 499500 | 13.1 | 9966 | 4918 | 9966 | 13879 | 0.76 |
| 10000 | 5000 | 9999 | 49995000 | 49995000 | 1275 | 132877 | 51999 | 132877 | 593588 | 20.7 |

A math equations with numbers

Description automatically generated with medium confidence