Харківський національний університет імені В. Н. Каразіна
Факультет комп'ютерних наук
Кафедра штучного інтелекту та програмного забезпечення

ЗВІТ
З ПРАКТИЧНОЇ РОБОТИ №9

дисципліна: «Алгоритми та структур и даних»

Виконала: студентка групи КС-22
Узенкова Дар’я
Перевірив: Олешко Олег

Харків
2024

Завдання. Порівняння Timsort та Quicksort

Провести порівняльне дослідження продуктивності алгоритмів сортування Timsort та Quick sort.

Рекомендації щодо проведення дослідження наведено у файлі "Методика тестування продуктивності алгоритмів".

Основне завдання - отримати практичне підтвердження теоретичних оцінок складності алгоритмів для "кращого", "найгіршого" та "середнього" випадку.

Підготувати доповідь за результатами порівняння.

Рекомендований матеріал: https://en.wikipedia.org/wiki/Timsort

Результати виконання завдання наведено:

1. У лістингах 1 - 3 – вихідний код програми.
2. У малюнках 1, 2, 3 – результати виконання програми.
3. У графіках залежності часу виконання сортування від розміру
4. У таблицях 1 - 5.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Прототипи функцій
void quickSort(int *arr, int low, int high, int *comparisons, int *swaps);
int partition(int *arr, int low, int high, int *comparisons, int *swaps);
void swap(int *a, int *b, int *swaps);
int *read_array_from_file(const char *filename, int *n);
void write_array_to_file(const char *filename, int *arr, int n);

int main() {
    char inputFile[256];
    const char *outputFile = "output.txt";
    int size = 0;
    int comparisons = 0;  // Счётчик сравнений
    int swaps = 0;        // Счётчик обменов

    printf("Введіть ім'я файлу для читання (наприклад, input.txt): ");
    scanf("%255s", inputFile);

    int *arr = read_array_from_file(inputFile, &size);
    if (arr == NULL) {
        printf("Помилка при зчитуванні масиву з файлу.\n");
        return 1;
    }
```

```
    clock_t start = clock();  // Початок вимірювання часу
    quickSort(arr, 0, size - 1, &comparisons, &swaps);
    clock_t end = clock();  // Кінець вимірювання часу

    double time_taken = (double)(end - start) * 1000000 / CLOCKS_PER_SEC; // В
мікросекундах
    printf("Час виконання сортування: %.2f мкс\n", time_taken);
    printf("Кількість порівнянь: %d\n", comparisons);
    printf("Кількість обмінів: %d\n", swaps);

    // Запис відсортованого масиву в файл
    write_array_to_file(outputFile, arr, size);

    // Звільнення пам'яті
    free(arr);
    return 0;
}

// Реалізація QuickSort з підрахунком порівнянь та обмінів
void quickSort(int *arr, int low, int high, int *comparisons, int *swaps) {
    if (low < high) {
        int pivot = partition(arr, low, high, comparisons, swaps);
        quickSort(arr, low, pivot - 1, comparisons, swaps);
        quickSort(arr, pivot + 1, high, comparisons, swaps);
    }
}

// Функція розподілу з підрахунком порівнянь та обмінів
int partition(int *arr, int low, int high, int *comparisons, int *swaps) {
    int pivot = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
        (*comparisons)++;  // Збільшуємо кількість порівнянь
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j], swaps);
        }
    }
    swap(&arr[i + 1], &arr[high], swaps);
    return (i + 1);
}

// Функція обміну елементів з підрахунком обмінів
void swap(int *a, int *b, int *swaps) {
    int temp = *a;
    *a = *b;
    *b = temp;
    (*swaps)++;  // Збільшуємо кількість обмінів
}

// Читання масиву з файлу без указания размера
```

```c
int *read_array_from_file(const char *filename, int *n) {
    FILE *file = fopen(filename, "r");
    if (!file) {
        perror("Помилка при відкриванні файлу.");
        return NULL;
    }

    int capacity = 10;
    int *arr = (int *)malloc(capacity * sizeof(int));
    if (!arr) {
        perror("Помилка виділення пам'яті");
        fclose(file);
        return NULL;
    }

    int value;
    *n = 0;
    while (fscanf(file, "%d", &value) == 1) {
        if (*n >= capacity) {
            capacity *= 2;
            int *temp = realloc(arr, capacity * sizeof(int));
            if (!temp) {
                perror("Помилка перевиділення пам'яті");
                free(arr);
                fclose(file);
                return NULL;
            }
            arr = temp;
        }
        arr[*n] = value;
        (*n)++;
    }

    fclose(file);
    return arr;
}

// Запис масиву в файл
void write_array_to_file(const char *filename, int *arr, int n) {
    FILE *file = fopen(filename, "w");
    if (!file) {
        perror("Помилка при відкриванні файлу.");
        return;
    }

    for (int i = 0; i < n; i++) {
        fprintf(file, "%d ", arr[i]);
    }
    fclose(file);
}
```

Лістинг 1 – вихідний код програми для сортування QuickSort

```
Введіть ім'я файлу для читання (наприклад, input.txt): sorted_input.txt
Час виконання сортування: 361000.00 мкс
Кількість порівнянь: 49995000
Кількість обмінів: 50004999
PS C:\Users\uzenk>
```

Малюнок 1 – результат виконання програми

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MIN_RUN 32

// Прототипи функцій
void timSort(int *arr, int n);
void insertionSort(int *arr, int left, int right);
void merge(int *arr, int left, int mid, int right);
int *read_array_from_file(const char *filename, int *n);
void write_array_to_file(const char *filename, int *arr, int n);

int comparisons = 0;
int swaps = 0;

int main() {
    char inputFile[256];
    const char *outputFile = "output.txt";
    int size = 0;

    printf("Введіть ім'я файлу для читання (наприклад, input.txt): ");
    scanf("%255s", inputFile);

    int *arr = read_array_from_file(inputFile, &size);
    if (arr == NULL) {
        printf("Помилка при зчитуванні масиву з файлу.\n");
        return 1;
    }

    clock_t start = clock();  // Початок вимірювання часу
    timSort(arr, size);
    clock_t end = clock();  // Кінець вимірювання часу

    double time_taken = (double)(end - start) * 1000000 / CLOCKS_PER_SEC; // В
мікросекундах
    printf("Час виконання сортування: %.2f мкс\n", time_taken);
    printf("Кількість порівнянь: %d\n", comparisons);
    printf("Кількість обмінів: %d\n", swaps);

    // Запис відсортованого масиву в файл
```

```
    write_array_to_file(outputFile, arr, size);

    // Звільнення пам'яті
    free(arr);
    return 0;
}

// Реалізація TimSort
void timSort(int *arr, int n) {
    // Сортування частин масиву методом вставки
    for (int i = 0; i < n; i += MIN_RUN) {
        insertionSort(arr, i, (i + MIN_RUN - 1 < n - 1) ? (i + MIN_RUN - 1) : (n
- 1));
    }

    for (int size = MIN_RUN; size < n; size = 2 * size) {
        for (int left = 0; left < n; left += 2 * size) {
            int mid = left + size - 1;
            int right = (left + 2 * size - 1 < n - 1) ? (left + 2 * size - 1) :
(n - 1);
            if (mid < right) {
                merge(arr, left, mid, right);
            }
        }
    }
}

// Функція вставки для сортування маленьких діапазонів
void insertionSort(int *arr, int left, int right) {
    for (int i = left + 1; i <= right; i++) {
        int temp = arr[i];
        int j = i - 1;
        while (j >= left && arr[j] > temp) {
            comparisons++;
            arr[j + 1] = arr[j];
            swaps++;
            j--;
        }
        comparisons++;
        arr[j + 1] = temp;
        if (j + 1 != i) swaps++;
    }
}

// Злиття для двох відсортованих частин масиву
void merge(int *arr, int left, int mid, int right) {
    int len1 = mid - left + 1, len2 = right - mid;
    int *leftArr = (int *)malloc(len1 * sizeof(int));
    int *rightArr = (int *)malloc(len2 * sizeof(int));

    for (int i = 0; i < len1; i++)
```

```
        leftArr[i] = arr[left + i];
    for (int i = 0; i < len2; i++)
        rightArr[i] = arr[mid + 1 + i];

    int i = 0, j = 0, k = left;
    while (i < len1 && j < len2) {
        comparisons++;
        if (leftArr[i] <= rightArr[j]) {
            arr[k] = leftArr[i];
            i++;
        } else {
            arr[k] = rightArr[j];
            j++;
        }
        k++;
    }

    while (i < len1) {
        arr[k] = leftArr[i];
        i++;
        k++;
    }
    while (j < len2) {
        arr[k] = rightArr[j];
        j++;
        k++;
    }

    free(leftArr);
    free(rightArr);
}

// Читання масиву з файлу без указания размера
int *read_array_from_file(const char *filename, int *n) {
    FILE *file = fopen(filename, "r");
    if (!file) {
        perror("Помилка при відкриванні файлу.");
        return NULL;
    }

    int capacity = 10;
    int *arr = (int *)malloc(capacity * sizeof(int));
    if (!arr) {
        perror("Помилка виділення пам'яті");
        fclose(file);
        return NULL;
    }

    int value;
    *n = 0;
    while (fscanf(file, "%d", &value) == 1) {
```

```
        if (*n >= capacity) {
            capacity *= 2;
            int *temp = realloc(arr, capacity * sizeof(int));
            if (!temp) {
                perror("Помилка перевиділення пам'яті");
                free(arr);
                fclose(file);
                return NULL;
            }
            arr = temp;
        }
        arr[*n] = value;
        (*n)++;
    }

    fclose(file);
    return arr;
}

// Запис масиву в файл
void write_array_to_file(const char *filename, int *arr, int n) {
    FILE *file = fopen(filename, "w");
    if (!file) {
        perror("Помилка при відкриванні файлу.");
        return;
    }

    for (int i = 0; i < n; i++) {
        fprintf(file, "%d ", arr[i]);
    }
    fclose(file);
}
```

Лістинг 2 – вихідний код програми сортування TimSort

```
Введіть ім'я файлу для читання (наприклад, input.txt): sorted_input.txt
Час виконання сортування: 8000.00 мкс
Кількість порівнянь: 337429
Кількість обмінів: 0
PS C:\Users\uzenk>
```

Малюнок 2 – результат виконання програми

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void generate_sorted_array(const char *filename, int n) {
    FILE *file = fopen(filename, "w");
    if (!file) {
        perror("Помилка при створенні файлу.");
```

```
        return;
    }

    for (int i = 0; i < n; i++) {
        fprintf(file, "%d ", i + 1);
    }
    fclose(file);
}

void generate_reverse_sorted_array(const char *filename, int n) {
    FILE *file = fopen(filename, "w");
    if (!file) {
        perror("Помилка при створенні файлу.");
        return;
    }

    for (int i = n; i > 0; i--) {
        fprintf(file, "%d ", i);
    }
    fclose(file);
}

void generate_random_array(const char *filename, int n) {
    FILE *file = fopen(filename, "w");
    if (!file) {
        perror("Помилка при створенні файлу.");
        return;
    }

    srand(time(NULL));

    for (int i = 0; i < n; i++) {
        fprintf(file, "%d ", rand() % n);
    }
    fclose(file);
}

int main() {
    int n;
    printf("Введіть кількість елементів у масиві: ");
    scanf("%d", &n);

    generate_sorted_array("sorted_input.txt", n);
    generate_reverse_sorted_array("reverse_sorted_input.txt", n);
    generate_random_array("random_input.txt", n);

    printf("Файли згенеровані: sorted_input.txt, reverse_sorted_input.txt,
random_input.txt\n");

    return 0;
}
```

Лістинг 3 – вихідний код програми для генерування файлу з вхідними даними

QuickSort

Таблиця 1 - Дані для найкращого випадку

Час виконання	105000	361000	1801000	2765000	4095000
Розмір масиву	5000	10000	20000	25000	30000

Таблиця 2 - Дані для середнього випадку

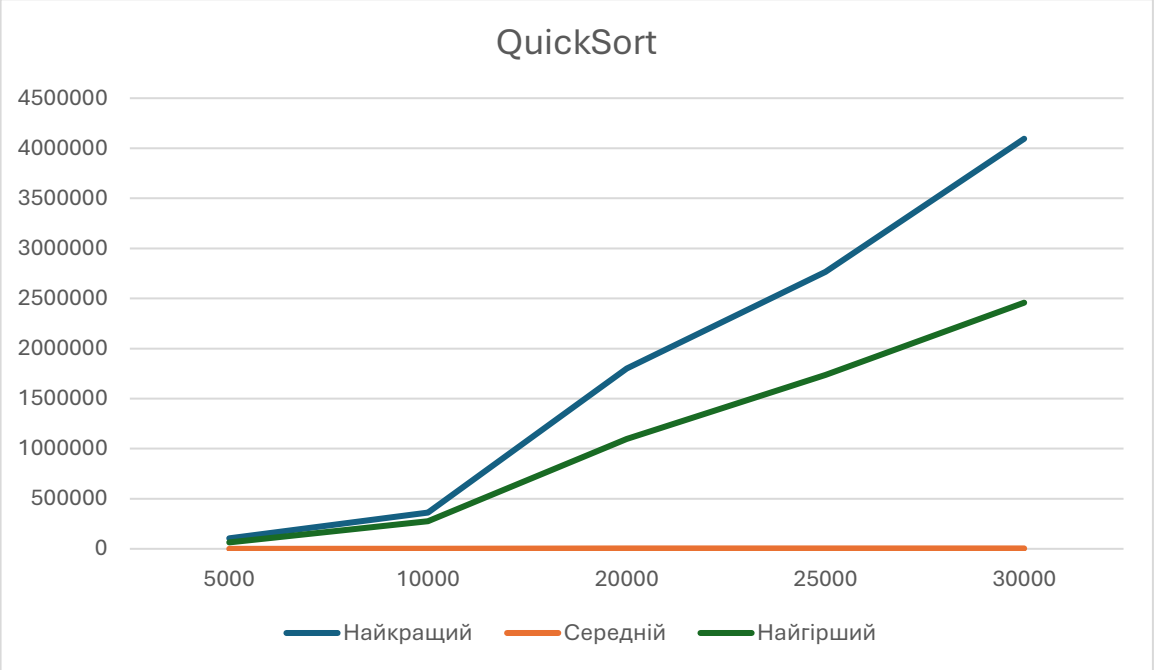
Час виконання	0	1000	3000	4000	4000
Розмір масиву	5000	10000	20000	25000	30000

Таблиця 3 - Дані для найгіршого випадку

Час виконання	64000	277000	1097000	1736000	2458000
Розмір масиву	5000	10000	20000	25000	30000

Таблиця 4 – QuickSort. Операції при сортуванні (беремо випадок з 10 000 елементів)

Випадок	Порівняння	Перестановки	Час
Найкращий	49995000	50004999	361000
Середній	152974	80373	1000
Найгірший	49995000	25004999	262000



Графік залежності часу виконання QuickSort від розміру

TimSort

Таблиця 1 - Дані для найкращого випадку

Час виконання	1000	2000	2000	4000	4000
Розмір масиву	5000	10000	20000	25000	30000

Таблиця 2 - Дані для середнього випадку

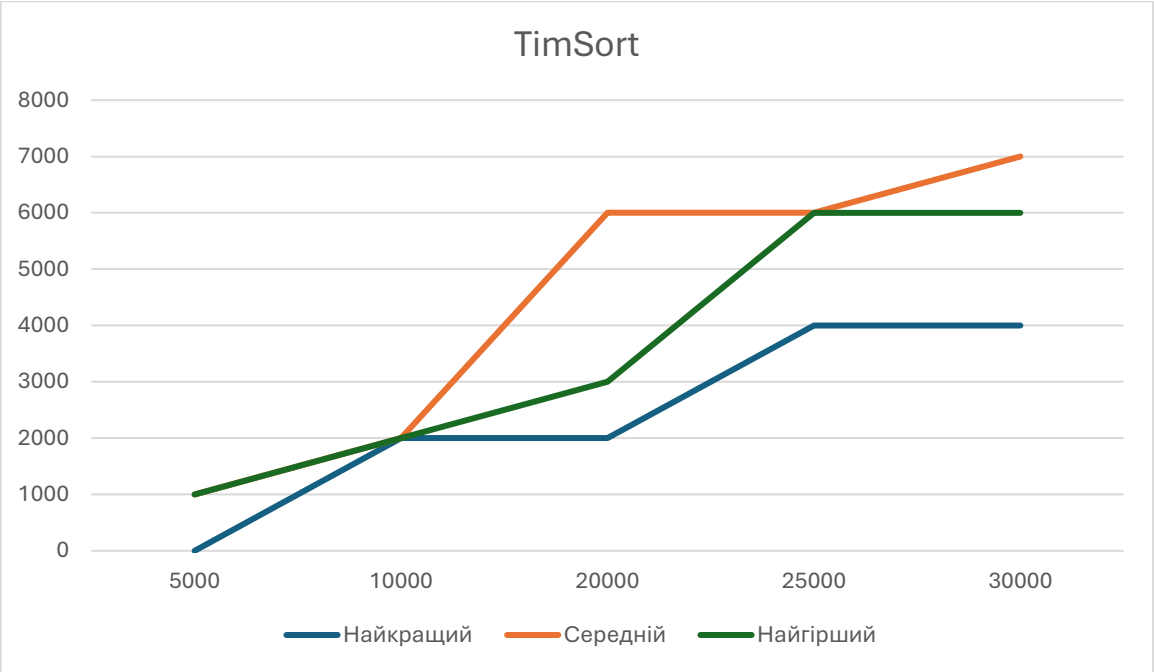
Час виконання	1000	2000	6000	6000	7000
Розмір масиву	5000	10000	20000	25000	30000

Таблиця 3 - Дані для найгіршого випадку

Час виконання	1000	2000	3000	6000	6000
Розмір масиву	5000	10000	20000	25000	30000

Таблиця 4 – TimSort. Операції при сортуванні (беремо випадок з 10 000 елементів)

Випадок	Порівняння	Перестановки	Час
Найкращий	56407	0	2000
Середній	172061	85388	2000
Найгірший	204175	164559	2000



Графік залежності часу виконання TimSort від розміру

Таблиця 5 – Оцінка ефективності алгоритмів QuickSort та TimSort

Алгоритм сортування	Найкращий випадок	Середній випадок	Найгірший випадок
QuickSort	O(n^2)	O(n logn)	O(n^2)
TimSort	O(n)	O(n logn)	O(n logn)

Висновок:

TimSort забезпечує стабільну продуктивність у всіх випадках, тоді як QuickSort може демонструвати високий рівень продуктивності, але з можливими суттєвими падіннями ефективності у найгірших та найкращих випадках.

TimSort краще підходить для задач, де важлива стабільність, або для вже частково впорядкованих масивів. QuickSort може бути вибраний для задач, де швидкість критична, і дані не мають особливих вимог до стабільності.