Харківський національний університет імені В.Н. Каразіна

Факультет комп’ютерних наук

ЗВІТ
З ПРАКТИЧНОЇ РОБОТИ №6
Дослідження ефективності алгоритмів Heap Sort і Quick Sort
дисципліна «ТЕОРІЯ АЛГОРИТМІВ»

Виконав: студент групи КС22

Прохватілов Антон

Перевірив: к.т.н., Олешко Олег Іванович

Харків

2024

Завдання:

Провести порівняльне дослідження ефективності алгоритмів пірамідального та швидкого сортування (Heap and Quick sort). Основна задача - отримати практичне підтвердження теоретичних оцінок ефективності алгоритмів для "найкращого", "найгіршого" і "середнього" випадків.

Результати виконання завдання №1 наведено:

1. Лістинг 1
2. Рисунок 1
3. Графік 1-6
4. Таблиця 1-3

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

// Глобальні змінні для підрахунку порівнянь та перестановок
int comparisons = 0;
int swaps = 0;

// Функція для обміну двох елементів масиву
void Swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    swaps++;  // Збільшуємо кількість перестановок
}

// Функція, яка розбиває масив на дві частини навколо опорного елемента p
int Partition(int arr[], int low, int high) {
    int p = arr[low];
    int i = low;
    int j = high;

    while (i < j) {
        while (arr[i] <= p && i < high) {
            i++;
            comparisons++;  // Збільшуємо кількість порівнянь
        }
        while (arr[j] > p && j > low) {
            j--;
            comparisons++;  // Збільшуємо кількість порівнянь
        }
        if (i < j) {
            Swap(&arr[i], &arr[j]);
        }
    }
    Swap(&arr[low], &arr[j]);
    return j;
```

```
}

// Функція швидкого сортування
void QuickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = Partition(arr, low, high);
        QuickSort(arr, low, pi - 1);
        QuickSort(arr, pi + 1, high);
    }
}

// Функція, яка будує купу
void Heapify(int arr[], int n, int i) {
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;

    if (l < n && arr[l] > arr[largest]) {
        largest = l;
    }
    comparisons++;
    if (r < n && arr[r] > arr[largest]) {
        largest = r;
    }
    comparisons++;

    if (largest != i) {
        Swap(&arr[i], &arr[largest]);
        Heapify(arr, n, largest);
    }
}

// Функція сортування купою
void HeapSort(int arr[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--) {
        Heapify(arr, n, i);
```

```
    }
    for (int i = n - 1; i > 0; i--) {
        Swap(&arr[0], &arr[i]);
        Heapify(arr, i, 0);
    }
}

// Функція для зчитування масиву з текстового файлу
int* ArrFromTxt(const char* filename, int* n) {
    FILE* file = fopen(filename, "r");
    if (file == NULL) {
        printf("Не вдалося відкрити файл!\n");
        return NULL;
    }

    fscanf(file, "%d", n);  // Читаємо розмір масиву
    int* arr = (int*)malloc(*n * sizeof(int));

    for (int i = 0; i < *n; i++) {
        fscanf(file, "%d", &arr[i]);
    }

    fclose(file);
    return arr;
}

// Функція для запису масиву в текстовий файл
void ArrToTxt(const char* filename, int* arr, int n) {
    FILE* file = fopen(filename, "w");
    if (file == NULL) {
        printf("Не вдалося відкрити файл для запису!\n");
        return;
    }

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

```
    }

    fclose(file);
}

// Функція для зчитування масиву з бінарного файлу
int* ArrFromBin(const char* filename, int* n) {
    FILE* file = fopen(filename, "rb");
    if (file == NULL) {
        printf("Не вдалося відкрити файл!\n");
        return NULL;
    }

    fread(n, sizeof(int), 1, file);  // Читаємо розмір масиву
    int* arr = (int*)malloc(*n * sizeof(int));

    fread(arr, sizeof(int), *n, file);  // Читаємо сам масив

    fclose(file);
    return arr;
}

// Функція для запису масиву в бінарний файл
void ArrToBin(const char* filename, int* arr, int n) {
    FILE* file = fopen(filename, "wb");
    if (file == NULL) {
        printf("Не вдалося відкрити файл для запису!\n");
        return;
    }

    fwrite(&n, sizeof(int), 1, file);  // Записуємо розмір масиву
    fwrite(arr, sizeof(int), n, file);  // Записуємо масив

    fclose(file);
}
```

```
// Масив відсортований за зростанням
void BestCase(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        arr[i] = i + 1;
    }
}

// Масив відсортований рандомно
void AverageCase(int arr[], int n) {
    srand(time(NULL));
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 100;
    }
}

// Масив відсортований за спаданням
void WorstCase(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        arr[i] = n - i;
    }
}

// Функція для зміни розміру масиву
int SizeArr() {
    int new_size;
    printf("Введіть новий розмір масиву: ");
    scanf("%d", &new_size);
    printf("Розмір масиву змінено на %d.\n", new_size);
    return new_size;
}

int main() {
    int n, choice;
    int* arr = NULL;

    do {
```

```
        printf("\n--- Меню ---\n");
        printf("1. QuickSort\n");
        printf("2. HeapSort\n");
        printf("3. Зчитати масив з текстового файлу\n");
        printf("4. Зчитати масив з бінарного файлу\n");
        printf("5. Записати відсортований масив у текстовий файл\n");
        printf("6. Записати відсортований масив у бінарний файл\n");
        printf("7. Змінити розмір масиву\n");
        printf("8. Вийти\n");
        printf("Введіть свій вибір: ");
        scanf("%d", &choice);

        clock_t start, end;
        double time_taken;

        switch (choice) {
            case 1:
                if (arr == NULL) {
                    printf("Масив не ініціалізований!\n");
                    break;
                }
                comparisons = 0;  // Очищаємо лічильники
                swaps = 0;

                BestCase(arr, n);
                start = clock();
                QuickSort(arr, 0, n - 1);
                end = clock();
                time_taken = ((double)(end - start)) / CLOCKS_PER_SEC *
1000000;
                printf("Best-case QuickSort: Час сортування: %f
мікросекунд\n", time_taken);
                printf("Порівнянь: %d, Перестановок: %d\n", comparisons,
swaps);

                comparisons = 0;  // Очищаємо лічильники
                swaps = 0;
```

```
                AverageCase(arr, n);
                start = clock();
                QuickSort(arr, 0, n - 1);
                end = clock();
                time_taken = ((double)(end - start)) / CLOCKS_PER_SEC *
1000000;
                printf("Average-case QuickSort: Час сортування: %f
мікросекунд\n", time_taken);
                printf("Порівнянь: %d, Перестановок: %d\n", comparisons,
swaps);

                comparisons = 0;  // Очищаємо лічильники
                swaps = 0;

                WorstCase(arr, n);
                start = clock();
                QuickSort(arr, 0, n - 1);
                end = clock();
                time_taken = ((double)(end - start)) / CLOCKS_PER_SEC *
1000000;
                printf("Worst-case QuickSort: Час сортування: %f
мікросекунд\n", time_taken);
                printf("Порівнянь: %d, Перестановок: %d\n", comparisons,
swaps);
                break;

            case 2:
                if (arr == NULL) {
                    printf("Масив не ініціалізований!\n");
                    break;
                }
                comparisons = 0;  // Очищаємо лічильники
                swaps = 0;

                BestCase(arr, n);
                start = clock();
                HeapSort(arr, n);
```

```
                end = clock();
                time_taken = ((double)(end - start)) / CLOCKS_PER_SEC *
1000000;
                printf("Best-case HeapSort: Час сортування: %f
мікросекунд\n", time_taken);
                printf("Порівнянь: %d, Перестановок: %d\n", comparisons,
swaps);


                comparisons = 0;  // Очищаємо лічильники
                swaps = 0;


                AverageCase(arr, n);
                start = clock();
                HeapSort(arr, n);
                end = clock();
                time_taken = ((double)(end - start)) / CLOCKS_PER_SEC *
1000000;
                printf("Average-case HeapSort: Час сортування: %f
мікросекунд\n", time_taken);
                printf("Порівнянь: %d, Перестановок: %d\n", comparisons,
swaps);


                comparisons = 0;  // Очищаємо лічильники
                swaps = 0;


                WorstCase(arr, n);
                start = clock();
                HeapSort(arr, n);
                end = clock();
                time_taken = ((double)(end - start)) / CLOCKS_PER_SEC *
1000000;
                printf("Worst-case HeapSort: Час сортування: %f
мікросекунд\n", time_taken);
                printf("Порівнянь: %d, Перестановок: %d\n", comparisons,
swaps);
                break;


            case 3:
                arr = ArrFromTxt("ArrInput.txt", &n);
```

```
        if (arr != NULL) {
            printf("Масив зчитано з текстового файлу.\n");
        }
        break;

    case 4:
        arr = ArrFromBin("ArrInput.bin", &n);
        if (arr != NULL) {
            printf("Масив зчитано з бінарного файлу.\n");
        }
        break;

    case 5:
        if (arr == NULL) {
            printf("Масив не ініціалізований!\n");
            break;
        }
        ArrToTxt("ArrOutput.txt", arr, n);
        printf("Масив записано у текстовий файл.\n");
        break;

    case 6:
        if (arr == NULL) {
            printf("Масив не ініціалізований!\n");
            break;
        }
        ArrToBin("ArrOutput.bin", arr, n);
        printf("Масив записано у бінарний файл.\n");
        break;
    case 7:
        n = SizeArr();
        arr = (int *)realloc(arr, n * sizeof(int));
        break;
    case 8:
        printf("Вихід з програми...\n");
        break;
```

```
            default:
                printf("Невірний вибір! Спробуйте ще раз.\n");
                break;
        }
    } while (choice != 8);

    free(arr);  // Звільняємо пам'ять
    return 0;
}
```

Лістинг 1. Вихідний код програми

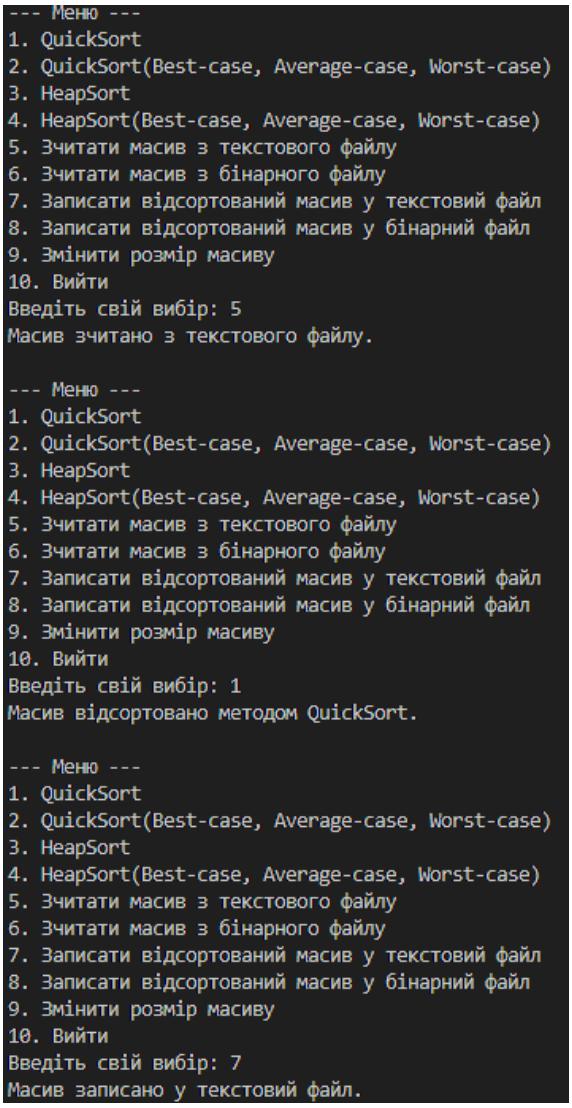
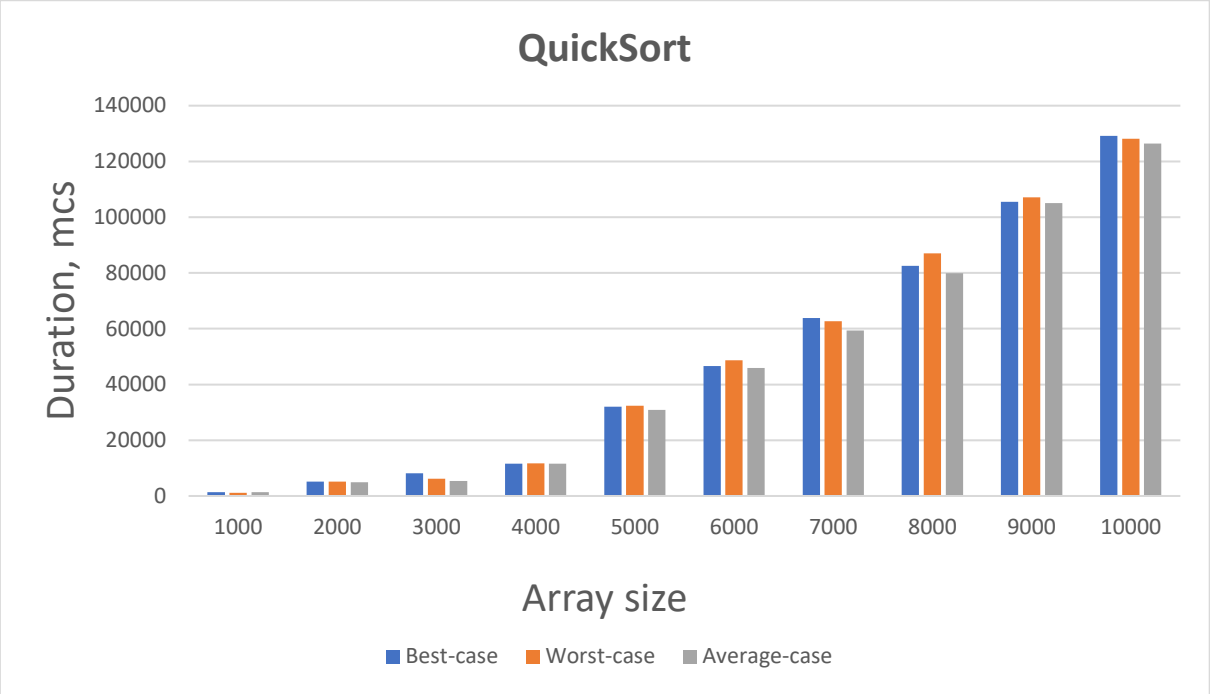
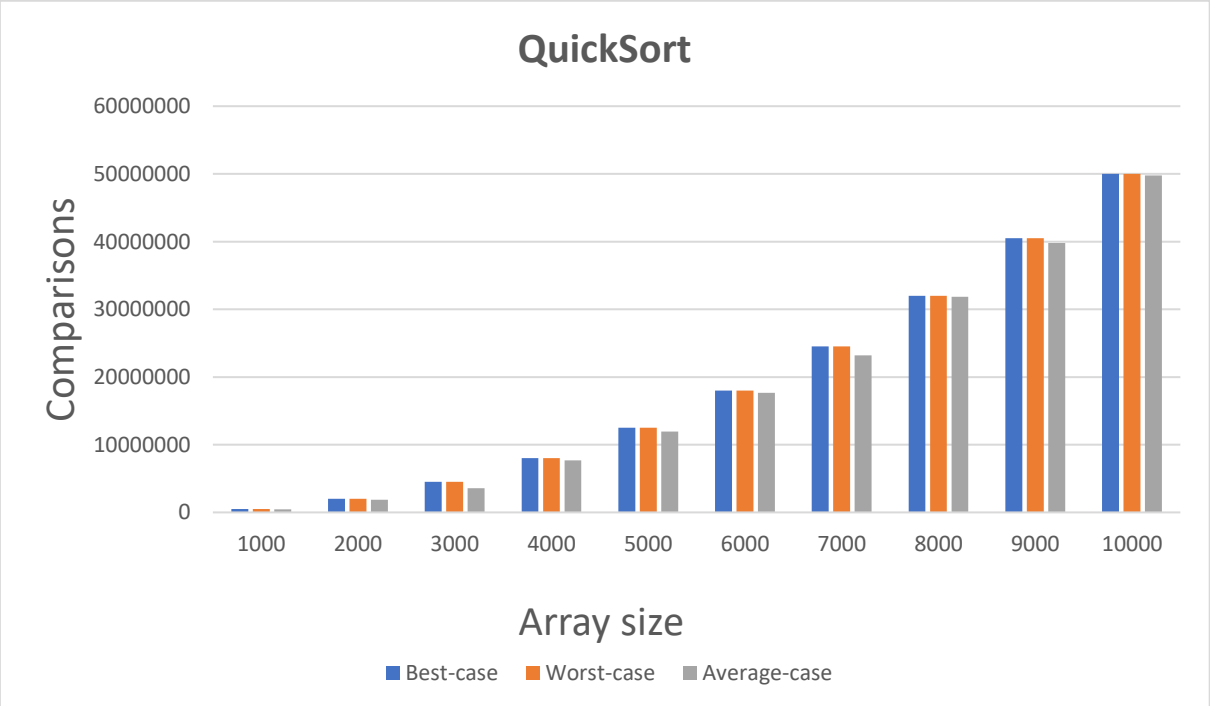


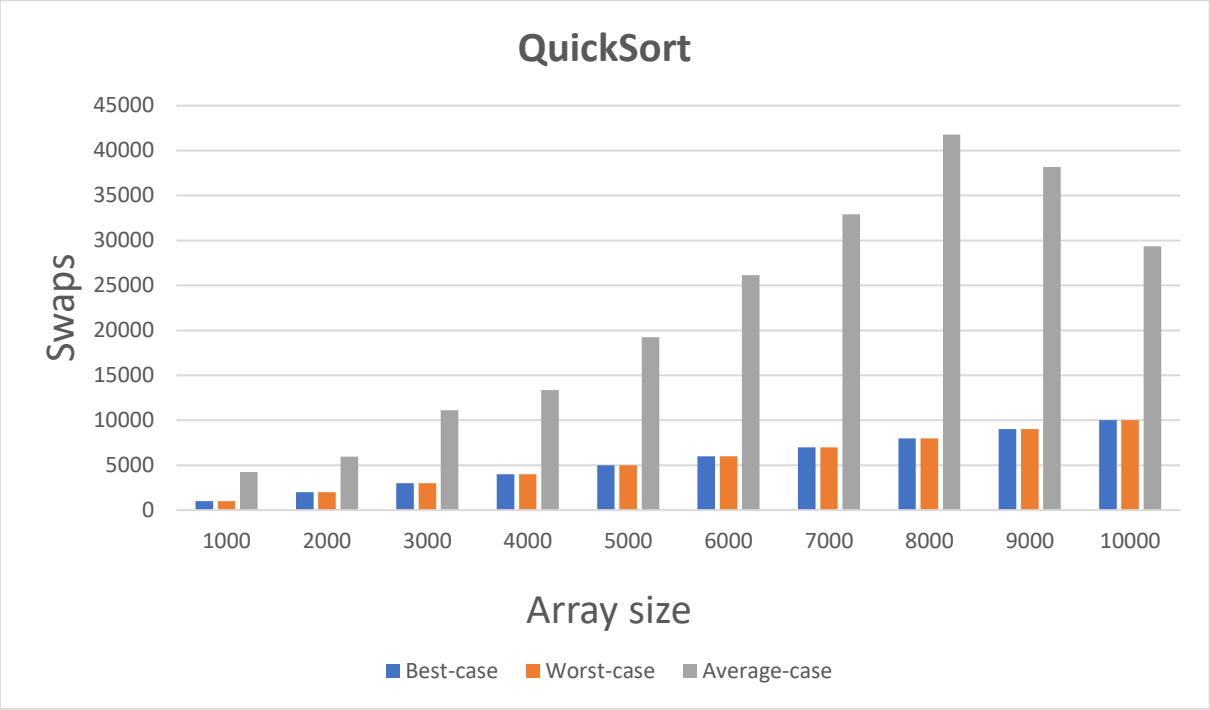
Рисунок 1. Результат виконання програми



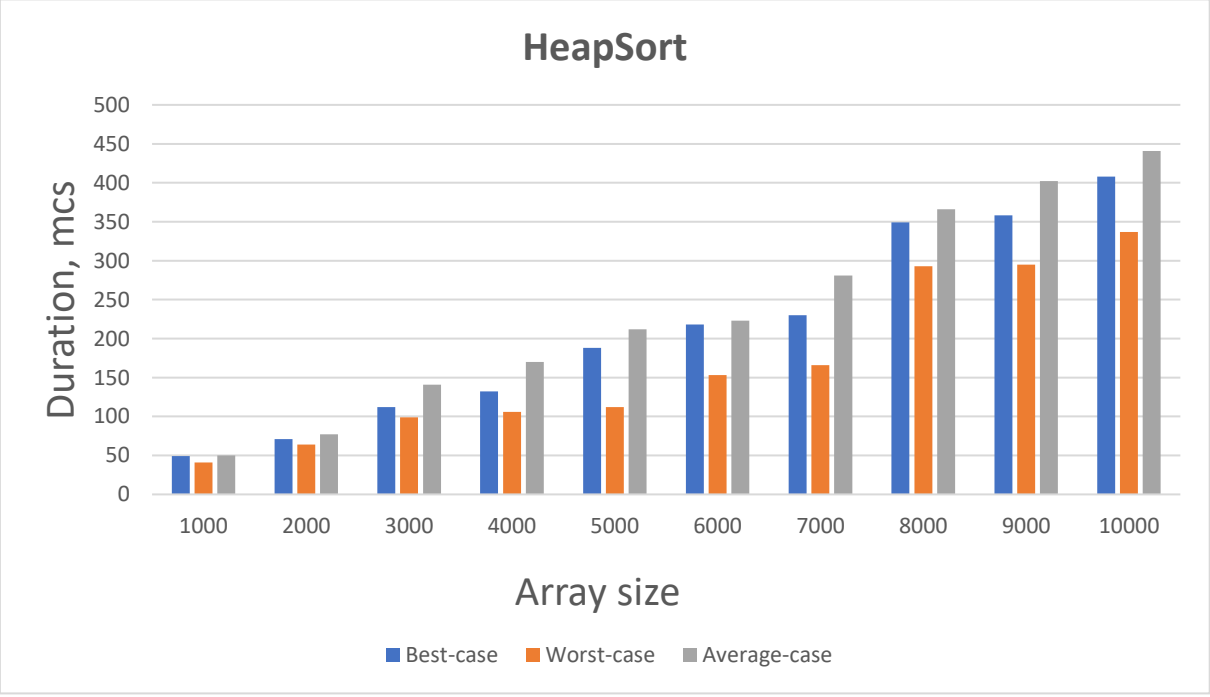
Графік 1. Графік швидкості сортування масиву алгоритма QuickSort у мікросекундах



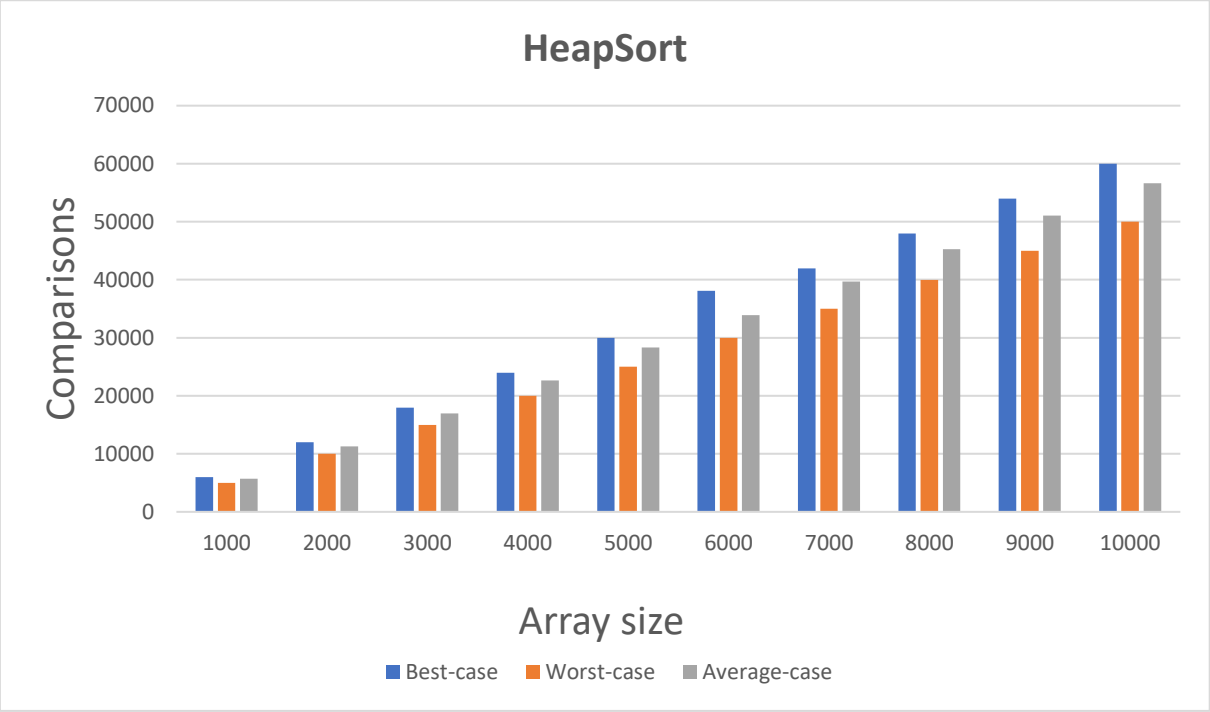
Графік 2. Графік порівнянь у масиві алгоритма QuickSort



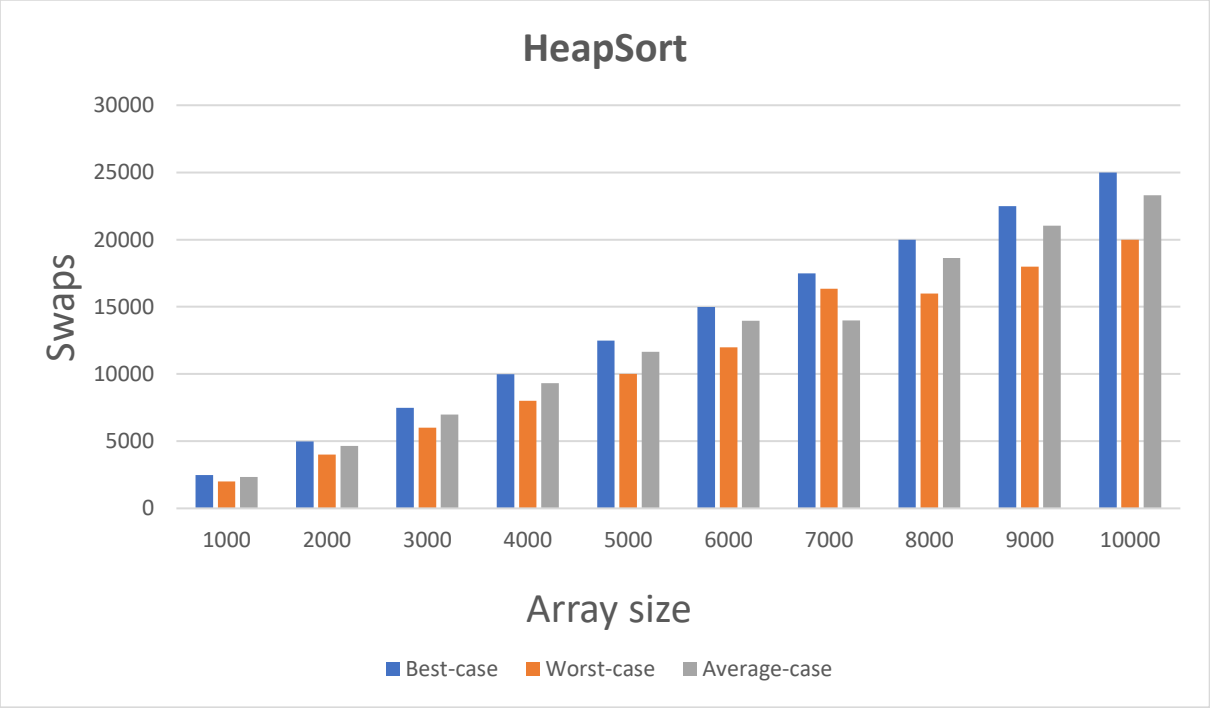
Графік 3. Графік перестановок у масиві алгоритма QuickSort



Графік 4. Графік швидкості сортування масиву алгоритма HeapSort у мікросекундах



Графік 5. Графік порівнянь у масиві алгоритма HeapSort



Графік 6. Графік перестановок у масиві алгоритма QuickSort

	Duration, mcs			Comparisons			Swaps		
Size	Best-case	Worst-case	Average-case	Best-case	Worst-case	Average-case	Best-case	Worst-case	Average-case
1000	1357	1178	1343	500499	499999	451373	999	999	4229
2000	5188	5157	4888	2000999	1999999	1889561	1999	1999	5948
3000	8182	6180	5363	4501499	4499999	3581832	2999	2999	11129
4000	11641	11732	11601	8001999	7999999	7683541	3999	3999	13343
5000	32029	32387	30928	12502499	12499999	11950830	4999	4999	19232
6000	46642	48625	45903	18002999	17999999	17688691	5999	5999	26149
7000	63846	62689	59377	24503499	24499999	23179213	6999	6999	32928
8000	82535	87034	79872	32003999	31999999	31836479	7999	7999	41799
9000	105479	107089	105078	40504499	40499999	39810113	8999	8999	38179
10000	129118	128074	126458	50004999	49999999	49765975	9999	9999	29360

Таблиця 1. Таблиці швидкості, порівнянь та перестановок алгоритма сортування QuickSort

	Duration, mcs			Comparisons			Swaps		
Size	Best-case	Worst-case	Average-case	Best-case	Worst-case	Average-case	Best-case	Worst-case	Average-case
1000	49	41	50	5976	4994	5688	2488	1997	2344
2000	71	64	77	11974	9994	11290	4987	3997	4645
3000	112	99	141	17974	14994	16976	7487	5997	6988
4000	132	106	170	23972	19994	22630	9986	7997	9315
5000	188	112	212	29972	24994	28310	12486	9997	11655
6000	218	153	223	38095	29994	33924	14986	11997	13962
7000	230	166	281	41970	34994	39698	17485	16349	13997
8000	349	293	366	47970	39994	45270	19985	15997	18635
9000	358	295	402	53970	44994	51054	22485	17997	21027
10000	408	337	441	59970	49994	56618	24985	19997	23309

Таблиця 2. Таблиці швидкості, порівнянь та перестановок алгоритма сортування HeapSort

Алгоритм	**Найкращий випадок**	**Середній випадок**	**Найгірший випадок**
QuickSort	O(n log n)	O(n log n)	O(n^2)
HeapSort	O(n log n)	O(n log n)	O(n log n)
Алгоритм	**Тривалість (мкс), найкращий випадок**	**Тривалість (мкс), середній випадок**	**Тривалість (мкс), найгірший випадок**
QuickSort	1357 - 129118	1343 - 126458	1178 - 128074
HeapSort	49 - 408	50 - 441	41 - 337
Алгоритм	**Порівняння, найкращий випадок**	**Порівняння, середній випадок**	**Порівняння, найгірший випадок**
QuickSort	500499 - 50004999	451373 - 49765975	499999 - 49999999
HeapSort	5976 - 59970	5688 - 56618	4994 - 49994
Алгоритм	**Перестановки, найкращий випадок**	**Перестановки, середній випадок**	**Перестановки, найгірший випадок**
QuickSort	999 - 9999	4229 - 29360	999 - 9999
HeapSort	2488 - 24985	2344 - 23309	1997 - 19997

Таблиця 3. Оцінка ефективності алгоритмів