Харківський національний університет імені В.Н. Каразіна
Навчально-науковий інститут комп’ютерних наук та штучного інтелекту

ЗВІТ

З ПРАКТИЧНОЇ РОБОТИ №6

дисципліна: «Алгоритмізація та програмування»

Виконав: студент 2 курсу групи КС22
Спеціальності 122 «Комп’ютерні науки»
Скрипняк Тарас Артемович
Прийняв: викладач
Олешко О.І.

Харків – 2024

Завдання №1: Провести порівняльне дослідження ефективності алгоритмів пірамідального та швидкого сортування (Heap and Quick sort).
Рекомендації по проведенню дослідження наведені в файлі "Методика тестування ефективності алгоритмів.doc".
Основна задача - отримати практичне підтвердження теоретичних оцінок ефективності алгоритмів для "найкращого", "найгіршого" і "середнього" випадків.

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

// struct for stats
typedef struct {
    long long comparisons;
    long long swaps;
    long long iterations;
    double time;
} SortStats;

SortStats quicksortStats;
SortStats heapsortStats;

void resetStats(SortStats *stats);
void quicksort(int arr[], int low, int high);
int partition(int arr[], int low, int high);
void heapsort(int arr[], int n);
void heapify(int arr[], int n, int i);
void testSorts(int arr[], int n);

void resetStats(SortStats *stats) {
    stats->comparisons = 0;
    stats->swaps = 0;
    stats->iterations = 0;
    stats->time = 0.0;
}

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

void quicksort(int arr[], int low, int high) {
    quicksortStats.iterations++;
    if (low < high) {
        int pi = partition(arr, low, high);

        quicksort(arr, low, pi - 1);
        quicksort(arr, pi + 1, high);
    }
}

int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {
        quicksortStats.comparisons++;
         if (arr[j] < pivot) {
            i++;
            quicksortStats.swaps++;
            swap(&arr[i], &arr[j]);
        }
    }
```

```
    quicksortStats.swaps++;
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

void heapsort(int arr[], int n) {
    heapsortStats.iterations++;

    for (int i = n / 2 - 1; i >= 0; i--) heapify(arr, n, i);

    for (int i = n - 1; i >= 0; i--) {
        heapsortStats.swaps++;
        swap(&arr[0], &arr[i]);
        heapify(arr, i, 0);
    }
}

void heapify(int arr[], int n, int i) {
    heapsortStats.iterations++;
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    heapsortStats.comparisons++;
    if (left < n && arr[left] > arr[largest]) largest = left;

    heapsortStats.comparisons++;
    if (right < n && arr[right] > arr[largest]) largest = right;

    if (largest != i) {
        heapsortStats.swaps++;
        swap(&arr[i], &arr[largest]);
        heapify(arr, n, largest);
    }
}

void shuffleArray(int arr[], int n) {
    srand(time(0));

    for (int i = n - 1; i > 0; i--) {
        int j = rand() % (i + 1);

        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}

void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) printf("%d ", arr[i]);
    printf("\n");
}

void testSorts(int arr[], int n) {
    // create a copy of the array to prevent original array mutation till heapsort
    int *arrCopy = (int *) malloc(n * sizeof(int));

    for (int i = 0; i < n; i++) arrCopy[i] = arr[i];

    resetStats(&quicksortStats);
    clock_t start = clock();
    quicksort(arrCopy, 0, n - 1);
    clock_t end = clock();
    quicksortStats.time = (double)(end - start) / CLOCKS_PER_SEC * 1000000;  // time in
microseconds

    printf("Quicksort - Comparisons: %lld, Swaps: %lld, Iterations: %lld, Time: %.6f us\n",
           quicksortStats.comparisons, quicksortStats.swaps, quicksortStats.iterations,
quicksortStats.time);

    for (int i = 0; i < n; i++) arrCopy[i] = arr[i];

    resetStats(&heapsortStats);
    start = clock();  // Start timing
```

```
    heapsort(arrCopy, n);
    end = clock();  // End timing
    heapsortStats.time = (double)(end - start) / CLOCKS_PER_SEC * 1000000;  // time in
microseconds

    printf("Heapsort - Comparisons: %lld, Swaps: %lld, Iterations: %lld, Time: %.6f us\n",
           heapsortStats.comparisons, heapsortStats.swaps, heapsortStats.iterations,
heapsortStats.time);

    free(arrCopy);
}

int main() {
    int n;
    printf("Enter array length (uint): ");
    scanf("%d", &n);
    if (n < 1) return printf("Array length unacceptable. Try something bigger than 0");

    int arr[n];

    // best case: sorted array
    for (int i = 0; i < n; i++) arr[i] = i;
    printf("Best Case (sorted array):\n");
    testSorts(arr, n);

    // worst case: reversed array
    for (int i = 0; i < n; i++) {
        int a = arr[n - i];
        arr[n - i] = arr[n];
        arr[n] = a;
    }
    printf("\nWorst Case (reversed array):\n");
    testSorts(arr, n);

    // average case: random array
    shuffleArray(arr, n);
    printf("\nAverage Case (random array):\n");
    testSorts(arr, n);

    return 0;
}
```

Лістинг - вихідний код програми

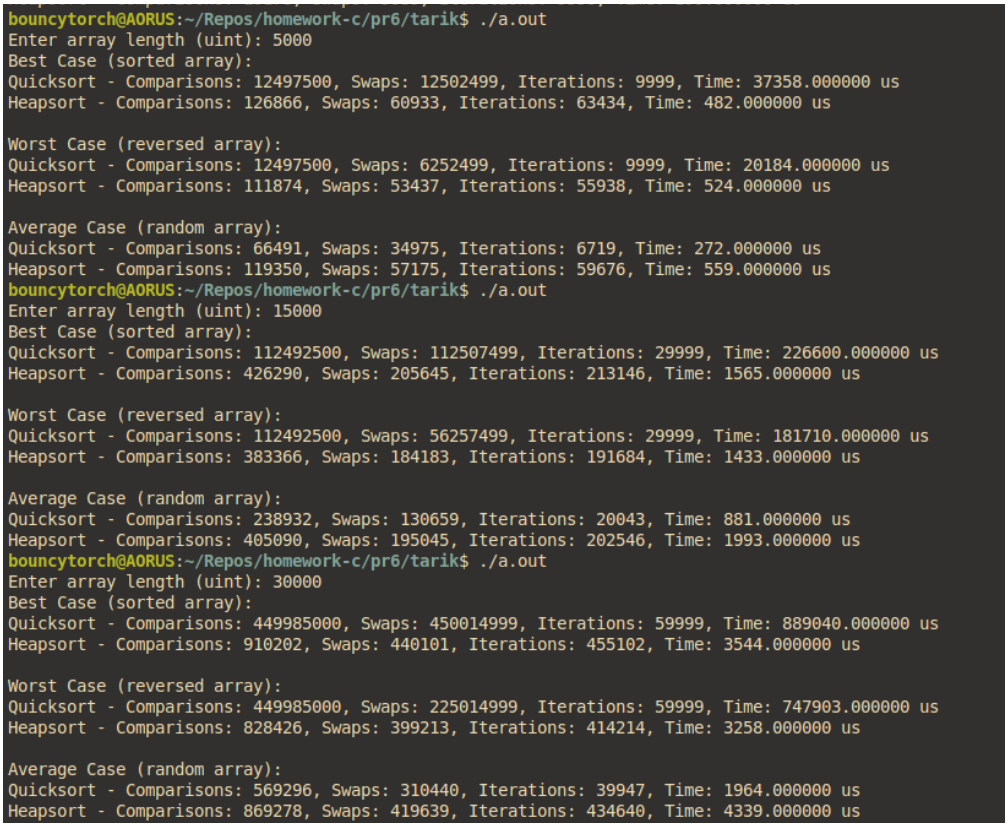

Рисунок 1 - результат виконання програми

N	quicksort			heapsort		
	best	worst	avg	best	worst	avg
5000	37358	20184	272	482	524	559
15000	226600	181710	881	1565	1433	1993
30000	889040	747903	3258	3544	3258	4339

Таблиця