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

ЗВІТ

З ПРАКТИЧНОЇ РОБОТИ №9

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 продуктивності алгоритмів сортування Timsort та Quick sort. Основне завдання - отримати практичне підтвердження теоретичних оцінок складності алгоритмів для "кращого", "найгіршого" та "середнього" випадку.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define RUN 32

// timsort

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) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = temp;
    }
}

void merge(int arr[], int l, int m, int r) {
    int len1 = m - l + 1, len2 = r - m;
    int *left = (int *)malloc(len1 * sizeof(int));
    int *right = (int *)malloc(len2 * sizeof(int));

    for (int i = 0; i < len1; i++)
        left[i] = arr[l + i];
    for (int i = 0; i < len2; i++)
        right[i] = arr[m + 1 + i];

    int i = 0, j = 0, k = l;

    while (i < len1 && j < len2) {
        if (left[i] <= right[j]) {
            arr[k] = left[i];
            i++;
        } else {
            arr[k] = right[j];
            j++;
        }
        k++;
    }

    while (i < len1) {
        arr[k] = left[i];
        i++;
        k++;
    }

    while (j < len2) {
        arr[k] = right[j];
        j++;
        k++;
    }

    free(left);
```

```
    free(right);
}

void timSort(int arr[], int n) {
    for (int i = 0; i < n; i += RUN)
        insertionSort(arr, i, (i + RUN - 1) < (n - 1) ? (i + RUN - 1) : (n -
1));

    for (int size = 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);
        }
    }
}

// quicksort

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

int partition(int arr[], int low, int high) {
      int p = arr[low],
        i = low,
        j = high;

      while (i < j) {
            while (arr[i] <= p && i <= high - 1) i++;
            while (arr[j] > p && j >= low + 1) j--;
            if (i < j) swap(&arr[i], &arr[j]);
      }

      swap(&arr[low], &arr[j]);
      return j;
}

void quickSort(int arr[], int low, int high) {
      if (low > high) return;
    int p = partition(arr, low, high);
      quickSort(arr, low, p - 1);
      quickSort(arr, p + 1, high);
}

void wrapquickSort(int arr[], int size) { quickSort(arr, 0, size); }

// general purpose

void printArray(int arr[], int size) {
    printf("[ ");
    for (int i = 0; i < size-1; i++)
        printf("%d, ", arr[i]);
    printf("%d ]\n", arr[size-1]);
}

void fillArray(int arr[], int size) {
    for (int i = 0; i < size; i++) arr[i] = i + 1;
}

void reverseArray(int arr[], int size) {
    for (int i = 0; i < size / 2; i++)
    {
        int temp = arr[i];
        arr[i] = arr[size - i - 1];
        arr[size - i - 1] = temp;
```

```
    }
}

void shuffleArray(int arr[], int size) {
    srand(time(0));
    for (int i = size - 1; i > 0; i--)
    {
        int j = rand() % (i + 1);
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}

void testSort(void (*sortFunction)(int*, int), int size) {
    int arr[size];

    // BEST CASE ========
    fillArray(arr, size);
    // /* DEBUG */ printArray(arr, size);
    clock_t start = clock();
    sortFunction(arr, size);
    clock_t end = clock();
    // /* DEBUG */ printArray(arr, size);
    printf("Best case: %lfms\n", ((double)(end - start)) / 1000 );

    // WORST CASE =======
    reverseArray(arr, size);
    // /* DEBUG */ printArray(arr, size);
    start = clock();
        sortFunction(arr, size);

    end = clock();
    // /* DEBUG */ printArray(arr, size);
    printf("Worst case: %lfms\n", ((double)(end - start)) / 1000 );

    // AVG CASE =======
    shuffleArray(arr, size);
    // /* DEBUG */ printArray(arr, size);
    start = clock();
        sortFunction(arr, size);

    end = clock();
    // /* DEBUG */ printArray(arr, size);
    printf("Avg. case: %lfms\n", ((double)(end - start)) / 1000 );
}

// Main function
int main() {
    int N;

    printf("Enter array length: ");
    scanf("%d", &N);
    if (N < 1) {
        printf("Arrays can't have negative length.");
        return 1;
    }

    printf("TIMSORT:\n");
    testSort(timSort, N);
    printf("QUICKSORT:\n");
    testSort(wrapquickSort, N);

    return 0;
}
```

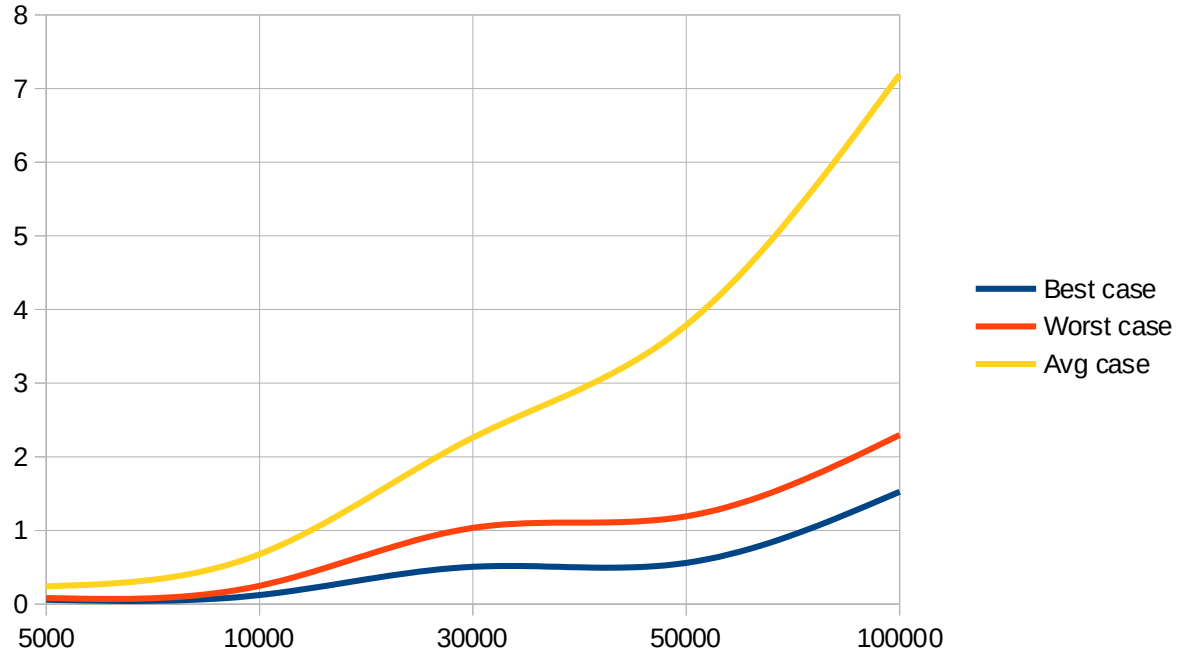
Лістинг - вихідний код програми

```
bouncytorch@AORUS:~/Repos/homework-c/pr9/tarik$ ./a.out
Enter array length: 5000
TIMSORT:
Best case: 0.060000ms
Worst case: 0.083000ms
Avg. case: 0.242000ms
QUICKSORT:
Best case: 3.651000ms
Worst case: 4.174000ms
Avg. case: 0.269000ms
bouncytorch@AORUS:~/Repos/homework-c/pr9/tarik$ ./a.out
Enter array length: 10000
TIMSORT:
Best case: 0.123000ms
Worst case: 0.249000ms
Avg. case: 0.678000ms
QUICKSORT:
Best case: 16.915000ms
Worst case: 10.538000ms
Avg. case: 0.399000ms
bouncytorch@AORUS:~/Repos/homework-c/pr9/tarik$ ./a.out
Enter array length: 30000
TIMSORT:
Best case: 0.507000ms
Worst case: 1.035000ms
Avg. case: 2.260000ms
QUICKSORT:
Best case: 84.321000ms
Worst case: 91.650000ms
Avg. case: 1.272000ms
bouncytorch@AORUS:~/Repos/homework-c/pr9/tarik$ ./a.out
Enter array length: 50000
TIMSORT:
Best case: 0.560000ms
Worst case: 1.192000ms
Avg. case: 3.788000ms
QUICKSORT:
Best case: 232.222000ms
Worst case: 254.091000ms
Avg. case: 2.140000ms
bouncytorch@AORUS:~/Repos/homework-c/pr9/tarik$ ./a.out
Enter array length: 100000
TIMSORT:
Best case: 1.526000ms
Worst case: 2.296000ms
Avg. case: 7.189000ms
QUICKSORT:
Best case: 917.413000ms
Worst case: 1031.534000ms
Avg. case: 4.506000ms
```

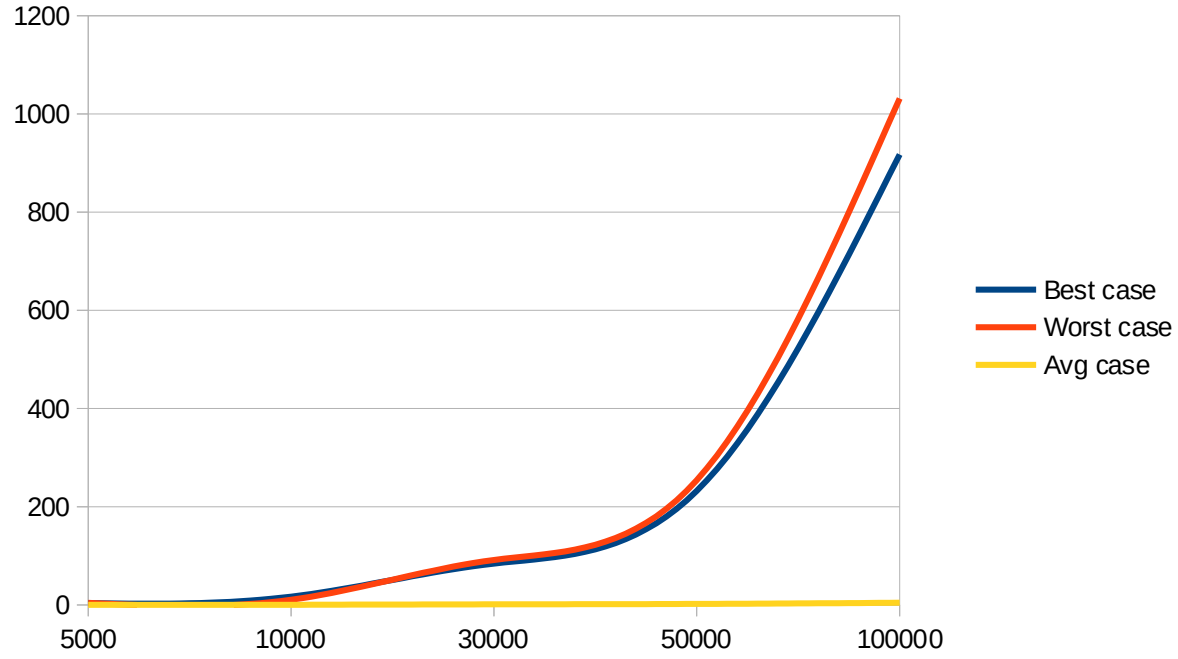
Рисунок 1 - результат виконання програми

Висновок: Timsort підтвердив свою ефективність із складністю O(nlogn) у найгіршому випадку і O(n) у кращому, показуючи стабільно швидкий час

виконання для різних розмірів масивів. Quicksort, хоча і демонструє O(nlogn) у середньому випадку, суттєво уповільнюється в найгіршому випадку зі складністю O(n^2). Експериментально ми бачимо, що Timsort залишається надійнішим і швидшим, особливо для великих масивів, тоді як Quicksort виявляє нестабільність у найгірших сценаріях.



Графік часу проти кількості елементів Timsort



Графік часу проти кількості елементів Quicksort