Харківський національний університет імені В.Н. Каразіна

Навчально-науковий інститут комп’ютерних наук та штучного інтелекту

**ЗВІТ**

**З ПРАКТИЧНОЇ РОБОТИ №8**

дисципліна: «Алгоритмізація та програмування»

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

Завдання №1: Реалізувати сортування JSort. Обґрунтувати теоретичну складність для кращого та гіршого випадків. Виміряти практичну швидкодію алгоритму для кращого та гіршого випадків. Зробити висновки на підставі отриманих результатів

|  |
| --- |
| #include <stdio.h> #include <stdlib.h> #include <time.h>  void jSort(int array[], int size) {  int i, j, minIndex, temp;    for (i = 0; i < size - 1; i++) {  minIndex = i;  for (j = i + 1; j < size; j++)  if (array[j] < array[minIndex])   minIndex = j;  temp = array[minIndex];  array[minIndex] = array[i];  array[i] = temp;  } }  void printArray(int array[], int size) {  for (int i = 0; i < size; i++) {  printf("%d ", array[i]);  }  printf("\n"); }  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(int size) {  int arr[size];   // BEST CASE ========  fillArray(arr, size);  // /\* DEBUG \*/ printArray(arr, size);  clock\_t start = clock();  jSort(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();  jSort(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();  jSort(arr, size);  end = clock();  // /\* DEBUG \*/ printArray(arr, size);  printf("Avg. case: %lfms\n", ((double)(end - start)) / 1000 ); }  // Головна функція int main() {  int N;   printf("Enter array length: ");  scanf("%d", &N);  if (N < 1) {  printf("Arrays can't have negative length.");  return 1;  }   testSort(N);   return 0; } |

Лістинг - вихідний код програми

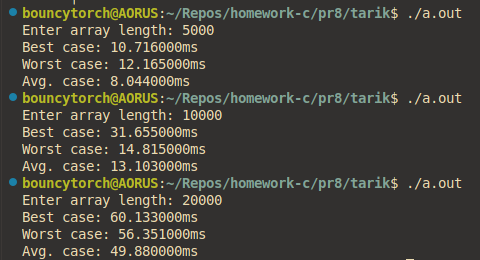


Рисунок 1 - результат виконання програми

Теоретичне обґрунтування складності:  
1. Кращий випадок: Найкраща складність виникає, коли масив вже відсортований або майже відсортований, що дозволяє алгоритму швидше виконати обхід. Теоретична оцінка може становити O(n^2).  
2. Гірший випадок: Гірший випадок має місце при масиві, що відсортований у зворотному порядку, або коли елементи потребують найбільшої кількості перестановок. Гірша оцінка для Jsort — O(n^2).

Висновок: Як видно з результатів, час виконання алгоритму збільшується зі зростанням розміру масиву. Це очікувано, оскільки складність Jsort є квадратичною O(n^2) в гіршому випадку. Результати показують, що різниця між часом виконання в найкращому і гіршому випадках не є значною для наведених прикладів. Це може свідчити про те, що алгоритм Jsort не надто чутливий до упорядкованості даних у масиві. Алгоритм демонструє обмежену ефективність для більших масивів, оскільки значення часу виконання швидко зростають. Це підтверджує, що Jsort не підходить для великих обсягів даних у порівнянні з більш ефективними алгоритмами сортування. Jsort є простим алгоритмом із передбачуваною квадратичною складністю. Хоча він працює добре для невеликих масивів, його ефективність різко падає на великих наборах даних. Практичні вимірювання підтверджують теоретичні оцінки, показуючи, що час виконання значно збільшується зі зростанням розміру масиву. Для застосувань, що потребують обробки великих обсягів даних, слід розглядати більш оптимальні алгоритми, такі як QuickSort або Merge Sort.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **Best (ms)** | **Worst (ms)** | **Avg (ms)** |
| 5000 | 10.716 | 12.165 | 8.044 |
| 10000 | 31.655 | 14.815 | 13.103 |
| 20000 | 60.133 | 56.351 | 49.88 |

Таблиця – Аналіз результатів виконання