Харківський національний університет імені В. Н. Каразіна Факультет комп'ютерних наук

Кафедра штучного інтелекту та програмного забезпечення

ЗВІТ

Практична робота №6

дисципліна: «Алгоритми та структур и даних»

Виконав: студент групи КС-22

Малютiн Максим

Перевірив: викладач

Олешко Олег

Харків

**Завдання**. 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 підтверждення теоретичних оцінок ефективності алгоритмів для "найкращого", "найгіршого" і "середнього" випадків.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>  // Структура для зберігання статистики сортування  typedef struct {  long long comparisonCount; // Кількість порівнянь  long long swapCount; // Кількість обмінів  long long iterationCount; // Кількість ітерацій  double elapsedTime; // Час, витрачений на сортування  } SortMetrics;  // Глобальні змінні для статистики  SortMetrics quickMetrics;  SortMetrics heapMetrics;  // Прототипи функцій  void clearMetrics(SortMetrics \*metrics);  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 runSortingTests(int arr[], int n);  void shuffle(int arr[], int n);  void clearMetrics(SortMetrics \*metrics) {  metrics->comparisonCount = 0;  metrics->swapCount = 0;  metrics->iterationCount = 0;  metrics->elapsedTime = 0.0; // Скидання часу  }  // Функція для обміну елементів  void swapElements(int \*a, int \*b) {  int temp = \*a;  \*a = \*b;  \*b = temp;  }  // Реалізація швидкого сортування  void quickSort(int arr[], int low, int high) {  quickMetrics.iterationCount++;  if (low < high) {  int pivotIndex = partition(arr, low, high);  quickSort(arr, low, pivotIndex - 1);  quickSort(arr, pivotIndex + 1, high);  }  }  int partition(int arr[], int low, int high) {  int pivotValue = arr[high];  int i = low - 1;  for (int j = low; j < high; j++) {  quickMetrics.comparisonCount++;  if (arr[j] < pivotValue) {  i++;  quickMetrics.swapCount++;  swapElements(&arr[i], &arr[j]);  }  }  quickMetrics.swapCount++;  swapElements(&arr[i + 1], &arr[high]);  return (i + 1);  }  // Реалізація купчастого сортування  void heapSort(int arr[], int n) {  heapMetrics.iterationCount++;  // Побудова купи  for (int i = n / 2 - 1; i >= 0; i--)  heapify(arr, n, i);  // Витягування елементів з купи  for (int i = n - 1; i >= 0; i--) {  heapMetrics.swapCount++;  swapElements(&arr[0], &arr[i]);  heapify(arr, i, 0);  }  }  void heapify(int arr[], int n, int i) {  heapMetrics.iterationCount++;  int largest = i;  int leftChild = 2 \* i + 1;  int rightChild = 2 \* i + 2;  if (leftChild < n) {  heapMetrics.comparisonCount++;  if (arr[leftChild] > arr[largest])  largest = leftChild;  }  if (rightChild < n) {  heapMetrics.comparisonCount++;  if (arr[rightChild] > arr[largest])  largest = rightChild;  }  if (largest != i) {  heapMetrics.swapCount++;  swapElements(&arr[i], &arr[largest]);  heapify(arr, n, largest);  }  }  void shuffle(int arr[], int n) {  srand((unsigned int)time(NULL)); // Випадкове насіння  for (int i = n - 1; i > 0; i--) {  int j = rand() % (i + 1);  swapElements(&arr[i], &arr[j]);  }  }  // Функція для тестування сортувань  void runSortingTests(int arr[], int n) {  int \*tempArr = (int \*)malloc(n \* sizeof(int));  // Тест швидкого сортування  for (int i = 0; i < n; i++)  tempArr[i] = arr[i];  clearMetrics(&quickMetrics);  clock\_t start = clock(); // Початок вимірювання часу  quickSort(tempArr, 0, n - 1);  clock\_t end = clock(); // Кінець вимірювання часу  quickMetrics.elapsedTime = (double)(end - start) / CLOCKS\_PER\_SEC \* 1000000; // Час у мікросекундах  printf("Швидке Сортування - Порівняння: %lld, Обміни: %lld, Ітерації: %lld, Час: %.6f мкс\n",  quickMetrics.comparisonCount, quickMetrics.swapCount, quickMetrics.iterationCount, quickMetrics.elapsedTime);  // Тест купчастого сортування  for (int i = 0; i < n; i++)  tempArr[i] = arr[i];  clearMetrics(&heapMetrics);  start = clock(); // Початок вимірювання часу  heapSort(tempArr, n);  end = clock(); // Кінець вимірювання часу  heapMetrics.elapsedTime = (double)(end - start) / CLOCKS\_PER\_SEC \* 1000000; // Час у мікросекундах  printf("Купчасте Сортування - Порівняння: %lld, Обміни: %lld, Ітерації: %lld, Час: %.6f мкс\n",  heapMetrics.comparisonCount, heapMetrics.swapCount, heapMetrics.iterationCount, heapMetrics.elapsedTime);  free(tempArr);  }  // Основна функція для тестування  int main() {  int n;  printf("Введіть кількість елементів: ");  scanf("%d", &n);  int arr[n];  for (int i = 0; i < n; i++) {  arr[i] = i; // Відсортований масив  }  // Найкращий випадок: відсортований масив  printf("Найкращий Випадок (відсортований масив):\n");  runSortingTests(arr, n);  // Найгірший випадок: обернений масив  for (int i = 0; i < n / 2; i++) {  swapElements(&arr[i], &arr[n - 1 - i]);  }  printf("\nНайгірший Випадок (обернений масив):\n");  runSortingTests(arr, n);  // Середній випадок: випадковий масив  shuffle(arr, n);  printf("\nСередній Випадок (випадковий масив):\n");  runSortingTests(arr, n);  return 0;  } |

Лiстинг 1 – вихiдний код програми

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Малюнок 1,2,3 – Скрiншот виконання програми

**QuickSort**

**Розмiр масиву 5000**

|  |  |  |  |
| --- | --- | --- | --- |
| **Випадок** | **Кiлькiсть порiвнянянь** | **Кiлькiсть пер-вок** | **Час виконання** |
| Найкращий | 12497500 | 12502499 | 40000 мкс |
| Найгiрший | 12497500 | 6252499 | 27000 мкс |
| Середнiй | 70823 | 40707 | 0 мкс |

**Розмiр масиву 15000**

|  |  |  |  |
| --- | --- | --- | --- |
| **Випадок** | **Кiлькiсть порiвнянянь** | **Кiлькiсть пер-вок** | **Час виконання** |
| Найкращий | 112492500 | 112507499 | 363000 мкс |
| Найгiрший | 112492500 | 56257499 | 242000 мкс |
| Середнiй | 247110 | 124111 | 1000 мкс |

**Розмiр масиву 30000**

|  |  |  |  |
| --- | --- | --- | --- |
| **Випадок** | **Кiлькiсть порiвнянянь** | **Кiлькiсть пер-вок** | **Час виконання** |
| Найкращий | 449985000 | 450014999 | 1450000 мкс |
| Найгiрший | 449985000 | 225014999 | 968000 мкс |
| Середнiй | 535998 | 287900 | 2000 мкс |

**Графiк витраченого часу QuickSort**

**HeapSort**

**Розмiр масиву 5000**

|  |  |  |  |
| --- | --- | --- | --- |
| **Випадок** | **Кiлькiсть порiвнянянь** | **Кiлькiсть пер-вок** | **Час виконання** |
| Найкращий | 112126 | 60933 | 0 мкс |
| Найгiрший | 103227 | 53437 | 1000 мкс |
| Середнiй | 70823 | 40707 | 0 мкс |

**Розмiр масиву 15000**

|  |  |  |  |
| --- | --- | --- | --- |
| **Випадок** | **Кiлькiсть порiвнянянь** | **Кiлькiсть пер-вок** | **Час виконання** |
| Найкращий | 383177 | 205645 | 2000 мкс |
| Найгiрший | 357789 | 184183 | 2000 мкс |
| Середнiй | 370448 | 195188 | 2000 мкс |

**Розмiр масиву 30000**

|  |  |  |  |
| --- | --- | --- | --- |
| **Випадок** | **Кiлькiсть порiвнянянь** | **Кiлькiсть пер-вок** | **Час виконання** |
| Найкращий | 826347 | 440101 | 4000 мкс |
| Найгiрший | 775687 | 399213 | 4000 мкс |
| Середнiй | 801460 | 420797 | 5000 мкс |

**Графiк витраченого часу HeapSort**

**Загальний Висновок:**

HeapSort показав переваги у найгіршому та найкращому випадках, виконуючи менше операцій і працюючи швидше за QuickSort у найгіршому випадку. Проте в середньому випадку QuickSort продемонстрував кращі результати за часом і ефективністю виконання. QuickSort може бути більш чутливим до початкового стану масиву, в той час як HeapSort забезпечує стабільніші результати.

Загалом, QuickSort підходить для швидкого сортування випадкових даних, тоді як HeapSort краще підходить для сценаріїв, де важлива стабільність часу виконання, особливо для відсортованих чи обернених масивів**.**