Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Реферат

По дисциплине «Основы алгоритмизации и программирования»

На тему «Анализ и сравнение времени выполнения сортировок: пирамидальной, быстрой, слиянием, Шелла»

 Выполнил:

Студент 1 курса 6 группы

Романов Игорь Вячеславович

Преподаватель: доц Белодед Н.И.

2024, Минск

**Содержание**

[Введение 3](#_Toc166183038)

[Пирамидальная сортировка 3](#_Toc166183039)

[Быстрая сортировка 3](#_Toc166183040)

[Сортировка слиянием 4](#_Toc166183041)

[Сортировка вставка (Сортировка Шейла) 4](#_Toc166183042)

[Заключение 6](#_Toc166183043)

[Код 6](#_Toc166183044)

Введение

В данном исследовании будут рассмотрены четыре различные метода сортировки: пирамидальная, быстрая, сортировка слиянием и сортировка вставками. Каждая из них обладает уникальным алгоритмом, требующая изучения оценки времени О-большого подаваемых на вход данных и их количества.

Для оценки времени используются функции библиотеки chrono.

Пирамидальная сортировка

Пирамидальная сортировка, также известная как сортировка кучей, представляет собой модификацию пузырьковой сортировки. Основная идея алгоритма заключается в поиске максимального элемента в неотсортированной части массива и его перемещении в конец этой части. Для поиска максимума используется структура данных "сортирующее дерево" или двоичная куча. После каждого перемещения максимального элемента процедура повторяется для оставшейся части массива.

Пирамидальная сортировка обеспечивает стабильно время выполнения независимо от расположения элементов в исходном массиве. Это объясняется особенностью алгоритма: элементы массива всегда будут перестраиваться в бинарную кучу. И тогда итоговая О-нотация:

Хуший случай: O(n \* logn)

Средний случай: O (n \* logn)

Лучший случай: O (n \* logn)

Быстрая сортировка

Быстрая сортировка - это улучшенная версия пузырьковой сортировки с более высокой эффективностью. Основное отличие заключается в том, что элементы сортируются наиболее удаленными образом, разделяясь на две группы после каждого прохода.

Алгоритм включает в себя выбор опорного элемента, обычно медианы массива, и разделение массива на две части вокруг опорного элемента. Затем процедура повторяется рекурсивно для обеих частей

В случае быстрой сортировки в худшем варианте сложность алгоритма может достигать O(n2). если массив уже отсортирован в обратном порядке, а опорный элемент выбирается как первый или последний элемент, то при каждом разделении массив будет разбиваться только на один подмассив размером n - 1 и один пустой подмассив. Такая ситуация будет повторяться n раз, что приведет к квадратичной сложности выполнения.

Чтобы быстрая сортировка работала эффективно, важно правильно выбирать опорный элемент. Хорошей практикой является выбор опорного элемента случайным образом, либо использование оптимизированных методов выбора опорного элемента, таких как средний элемент или медиана трех. В таких случаях среднее и лучшее время выполнения быстрой сортировки составляет O(n \* logn), что делает ее одним из самых эффективных алгоритмов сортировки

Сортировка слиянием

Сортировка слиянием — алгоритм сортировки, который упорядочивает элементы массива в определённом порядке. Слияние означает объединение двух или более последовательностей в одну упорядоченную последовательность при помощи циклического выбора элементов, доступных в данный момент. Мы будем брать поочередно по одному элементу из каждого массива, сравнивать их и "сливать в один массив. Меньший элемент будем ставить первым, больший — вторым.

По сложности алгоритма данная сортировка идентична с пирамидальной сортировкой: во всех трех случаях лучшее время – O(n \* lognn). Это объясняется структурой: при любом раскладе массив будет элементы массива будут делиться на 2 половины до тех пор, пока каждый из их размеров не будет равен 1.

Сортировка вставками (Сортировка Шелла

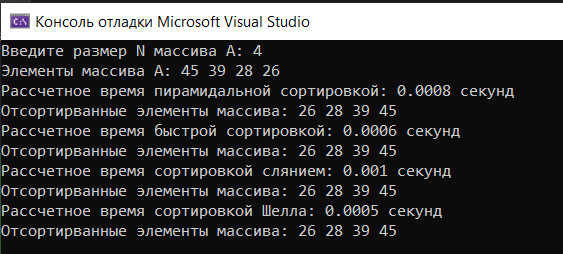
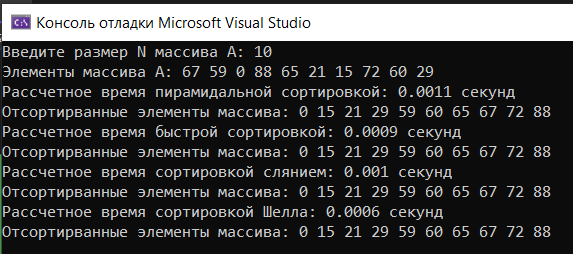
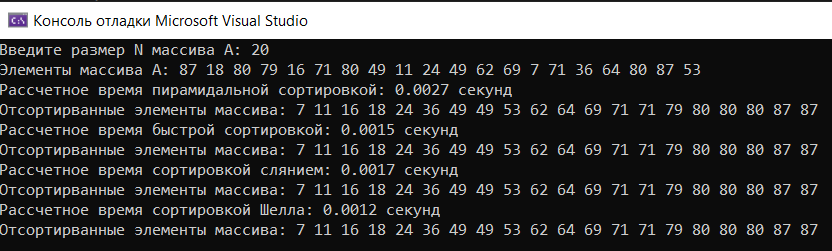
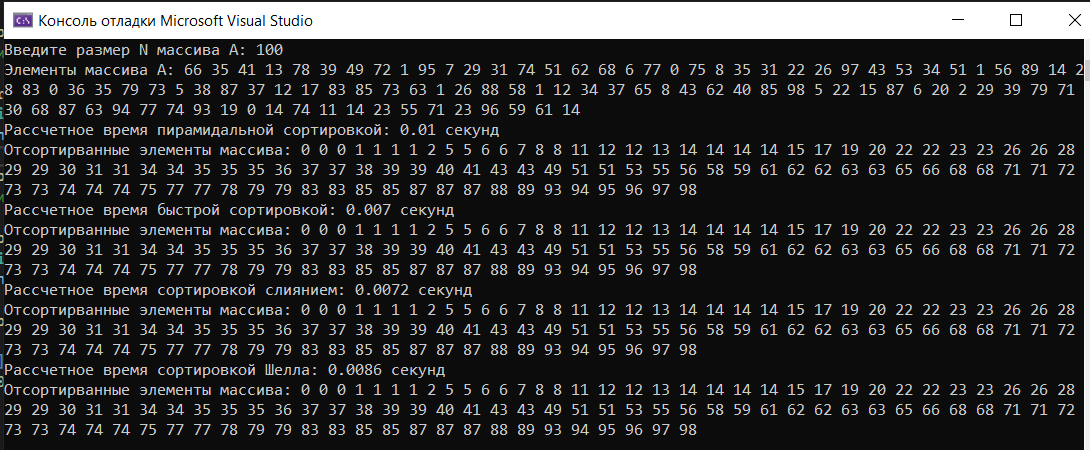
Сортировка вставками (Insertion Sort) - это простой алгоритм сортировки. Суть его заключается в том, что на каждом шаге алгоритма мы берем один из элементов массива, находим позицию для вставки и вставляем. Нужно отметить что массив из 1-го элемента считается отсортированным.

В лучшем случае, когда массив уже отсортирован или количество элементов не особо большое количество, сложность алгоритма составит O(n).

В худшем случае, когда элементы исходного массива расположены в обратном порядке, каждый новый элемент при вставке в уже отсортированную часть массива может требовать перемещения всех предыдущих элементов. Это приводит к тому к общей сложности O(n2).

В среднем случае, когда элементы исходного массива распределены случайным образом, сортировка вставками также может требовать O(n2) операций. Это происходит потому, что в среднем каждый элемент вставляется в середину отсортированной части массива, что требует примерно половину времени сортировки по сравнению с худшим случаем.

**Тестирование**

********

Заключение

В ходе тестирования былы проанализированы 4 вида сортировок и определены их О-нотации. Для сортировки массива с маленьким значением элементов лучше использовать сортировку вставками, для большего количества – сортировку быструю или слиянием. В таблице представлены временные сложности каждой из рассмотренной сортировок.



Код

|  |
| --- |
| #include <iostream>  #include <chrono> // библиотека для вызова функции chrono  #include <ctime>  using namespace std;  using namespace chrono;  void outputSortedArray(int\*, int);  void heapify(int\* A, int N, int pos);  void heapSort(int\* A, int N);  void quickSort(int\* B, int first, int last);  void merge(int\* C, int first, int last);  void mergeSort(int\* C, int low, int high);  void insertSort(int\* D, int n);  void outputSortedArray(int\* array, int N) {  cout << "Отсортирванные элементы массива: ";  for (int i = 0; i < N; i++) {  cout << array[i] << " ";  }  cout << endl;  }  //Построение пирамиды для пирамидальной сортировки  void heapify(int\* A, int N, int pos) {  int max = 0;  int temp = 0;  int left = 0;  int right = 0;  for (;;) {  left = 2 \* pos + 1;  right = left + 1;  if (left >= N) {  return;  }  else if (right >= N) {  max = left;  }  else if (A[left] > A[right]) {  max = left;  }  else {  max = right;  }  if (A[pos] > A[max]) {  return;  }  temp = A[pos];  A[pos] = A[max];  A[max] = temp;  pos = max;  }  }  //Пирамидальная сортировка  void heapSort(int\* A, int N) {  for (int i = N / 2; i >= 0; --i) {  heapify(A, N, i); //Строим пирамиду, перегруппировывая массив  }  for (int i = N - 1; i > 0; i--) {  //Перемещаем текущий корень в конец  int temp = A[0];  A[0] = A[i];  A[i] = temp;  //Продолжаем выстраивать пирамиду  heapify(A, i, 0);  }  }  //Быстрая сортировка  void quickSort(int\* E, int first, int last) {  // Объявление переменных  int mid, count;  int f = first, l = last;  mid = E[(f + l) / 2]; // Выбор опорного элемента  // Цикл разделения элементов массива  do {  // Поиск элемента, который больше или равен опорному, слева от опорного  while (E[f] < mid) {  f++;  }  // Поиск элемента, который меньше или равен опорному, справа от опорного  while (E[l] > mid) {  l--;  }  // Обмен элементами, чтобы все элементы слева от опорного были меньше или равны ему,  // а все элементы справа - больше или равны  if (f <= l) {  count = E[f];  E[f] = E[l];  E[l] = count;  f++;  l--;  }  } while (f < l); // Повторять, пока индексы не пересекутся  // Рекурсивный вызов функции для подмассивов слева и справа от опорного элемента  if (first < l) {  quickSort(E, first, l);  }  if (f < last) {  quickSort(E, f, last);  }  }  //Функция, сливающая массивы  void merge(int\* A, int first, int last) {  int middle, start, final, j;  int\* mas = new int[100];  middle = (first + last) / 2; //вычисление среднего элемента  start = first; //начало левой части  final = middle + 1; //начало правой части  for (j = first; j <= last; j++) //выполнять от начала до конца  if ((start <= middle) && ((final > last) || (A[start] < A[final]))) {  mas[j] = A[start];  start++;  }  else {  mas[j] = A[final];  final++;  }  for (j = first; j <= last; j++) A[j] = mas[j]; //возвращение результата в список  delete[]mas;  };  //Сортировка слиянием  void mergeSort(int\* A, int first, int last) {  if (first < last)  {  mergeSort(A, first, (first + last) / 2); //сортировка левой части  mergeSort(A, (first + last) / 2 + 1, last); //сортировка правой части  merge(A, first, last); //слияние двух частей  }  };  //Сортировка Шелла(вставками)  void insertSort(int\* D, int n) {  int buff, j;  for (int i = 1; i < n; i++) {  buff = D[i];  for (j = i - 1; j >= 0 && D[j] > buff; j--) {  D[j + 1] = D[j];  }  D[j + 1] = buff;  }  }  int main() {  setlocale(LC\_ALL, "Russian");  srand(time(NULL)); //устанавливаем новую точку отсчета времени  int N;  cout << "Введите размер N массива A: ";  cin >> N;  while (N < 0) {  cout << "Введено не натуральное число. Попробуйте еще раз: ";  cin >> N;  }  int\* A = new int[N]; //выделение динамической памяти дляA массива A  //заполнение массива A  for (int i = 0; i < N; i++) {  A[i] = rand() % 99;  }  //выделение динамической памяти для остальных массивов  int\* B = new int[N], \*C = new int[N], \*D = new int[N], \*E = new int[N];  //вывод массива  cout << "Элементы массива A: ";  for (int i = 0; i < N; i++) {  cout << A[i] << " ";  //копирование элементов из массива A в остальные массивы  B[i] = A[i];  C[i] = A[i];  D[i] = A[i];  E[i] = A[i];  }  // пирамидальная сортировка сортировка  auto start = high\_resolution\_clock::now();  heapSort(A, N);  auto end = high\_resolution\_clock::now();  duration<double, milli>duration = end - start;  cout << "\nРассчетное время пирамидальной сортировкой: " << duration.count() << " секунд" << endl;  outputSortedArray(A, N);  // быстрая сортировка  start = high\_resolution\_clock::now();  quickSort(B, 0, N - 1);  end = high\_resolution\_clock::now();  duration = end - start;  cout << "Рассчетное время быстрой сортировкой: " << duration.count() << " секунд" << endl;  outputSortedArray(B, N);  // сортировка слиянием  start = high\_resolution\_clock::now();  quickSort(C, 0, N - 1);  end = high\_resolution\_clock::now();  duration = end - start;  cout << "Рассчетное время сортировкой слянием: " << duration.count() << " секунд" << endl;  outputSortedArray(C, N);  // сортировка Шелла  start = high\_resolution\_clock::now();  insertSort(D, N);  end = high\_resolution\_clock::now();  duration = end - start;  cout << "Рассчетное время сортировкой Шелла: " << duration.count() << " секунд" << endl;  outputSortedArray(D, N);    delete[] A, B, C, D, E;  return 0;  } |