**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР.**

отчет

**по лабораторной работе №3**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: «Алгоритмы сортировки».

|  |  |  |
| --- | --- | --- |
| Студентка гр. 1301 |  | Носков Е. С. |
| Преподаватель |  | Родионова Е. А. |

Санкт-Петербург

2022

Оглавление

[Формулировка задания. 3](#_Toc20949)

[Оценка временной сложности. 3](#_Toc13623)

[Графики. 6](#_Toc26282)

[Текст программы. 8](#_Toc19449)

# Формулировка задания.

Реализовать следующие алгоритмы сортировки:

1. Сортировка вставками (Insertion sort)
2. Сортировка выбором (Selection sort)
3. Пузырьковая сортировка (Bubble sort)
4. Сортировка слиянием (Merge sort)
5. Сортировка Шелла (Shell sort)
6. Быстрая сортировка (Quick sort)

Для каждого алгоритма сортировки указать временную асимптотическую сложность для лучшего, худшего случая и среднего случая, а также пространственную сложность, подкрепив это логическими построениями (*в меру своих сил*). Свести получившиеся результаты в таблицу.

Построить график зависимости времени выполнения от размера входных данных и определить временную асимптотическую сложность для лучшего, худшего, среднего случая практически.

Определить наиболее быстрый алгоритм сортировки. Сравнить скорость его выполнения с одним из алгоритмов сортировки из базовых библиотек языка.

# Оценка временной сложности.

Приведённые ниже рассуждения о пространственной сложности основаны, на том, сколько места в памяти занимают параметры и локальные переменные каждой конкретной функции.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Название сортировки | код | Оценка временной сложности | | | Пространственная сложность |
| Лучший случай | Средний случай | Худший |
| Сортировка вставками | void insertSort(int numbers[], int size) {  for (int i = 1; i < size; i++)  for (int j = i; j > 0 && numbers[j - 1] > numbers[j]; j--)  swap(numbers[j - 1], numbers[j]);  return;  } | O(n) | O(n2) | O(n2) | Определяется размером памяти, выделяемым для 4 итераторов и переменных: numbers, size, i, j. |
| Сортировка выбором | void selectSort(int numbers[], int size) {  int min = 0;  int buf = 0;    for (int i = 0; i < size; i++)  {  min = i;  for (int j = i + 1; j < size; j++)  min = (numbers[j] < numbers[min]) ? j : min;    if (i != min)  {  buf = numbers[i];  numbers[i] = numbers[min];  numbers[min] = buf;}}  return;  } | O(n2) | O(n2) | O(n2) | Определяется размером памяти, выделяемым для 6 итераторов и переменных: numbers, size, i, j, min, buf |
| Пузырьковая сортировка | void bubbleSort(int numbers[], int size) {  for (int i = 0; i < size; i++) {  for (int j = 0; j < size-1; j++) {  if (numbers[j] > numbers[j + 1]) {  swap(numbers[j + 1], numbers[j]);  }  }  }  return;  } | O(n) | O(n2) | O(n2) | Определяется размером памяти, выделяемым для 4 итераторов и переменных: numbers, size, i, j. |
| Сортировка слиянием | void mergeSort(int numbers[], int size)  {  if (size < 2)return;  mergeSort(numbers, size / 2);  mergeSort(&numbers[size / 2], size - (size / 2));  int\* buf = new int[size];  int idbuf = 0, idl = 0, idr = size / 2;  while ((idl < size / 2) && (idr < size))  if (numbers[idl] < numbers[idr])  buf[idbuf++] = numbers[idl++];  else  buf[idbuf++] = numbers[idr++];  while (idl < size / 2) buf[idbuf++] = numbers[idl++];  while (idr < size) buf[idbuf++] = numbers[idr++];  for (idl = 0; idl < size; idl++)numbers[idl] = buf[idl];  delete[]buf;  } | O(nlogn) | O(nlogn) | O(nlogn) | Определяется размером памяти, выделяемым для 6 итераторов и переменных : numbers, size, i, j, idl, idr, buf. И количеством уровней рекурсии. |
| Сортировка Шелла | void ShellSort(int numbers[],int size)  {  int i, j, step;  int tmp;  for (step = size / 2; step > 0; step /= 2)  for (i = step; i < size; i++)  {  tmp = numbers[i];  for (j = i; j >= step; j -= step)  {  if (tmp < numbers[j - step])  numbers[j] = numbers[j - step];  else  break;  }  numbers[j] = tmp;  }  } | O(nlog2n) | Зависит от выбора шага | O(n2) | Определяется размером памяти, выделяемым для 4 итераторов и переменных: numbers, size, i, j. |
| Быстрая сортировка | void quickSort(int numbers[], int size) {  int i = 0;  int j = size - 1;  int mid = numbers[size / 2];  do {  while (numbers[i] < mid) {  i++;  }  while (numbers[j] > mid) {  j--;  }    if (i <= j) {  int tmp = numbers[i];  numbers[i] = numbers[j];  numbers[j] = tmp;  i++;  j--;  }  } while (i <= j);  if (j > 0) {  quickSort(numbers, j + 1);  }  if (i < size) {  quickSort(&numbers[i], size - i);  }  } | O(nlogn) | O(nlogn) | O(n2) | Определяется размером памяти, выделяемым для 5 итераторов и переменных: numbers, size, i, j, mid. И количеством уровней рекурсии. |

# Графики.

На рис. 1 представлена зависимость скорости работы алгоритма (в микросекундах) от размера входных данных.

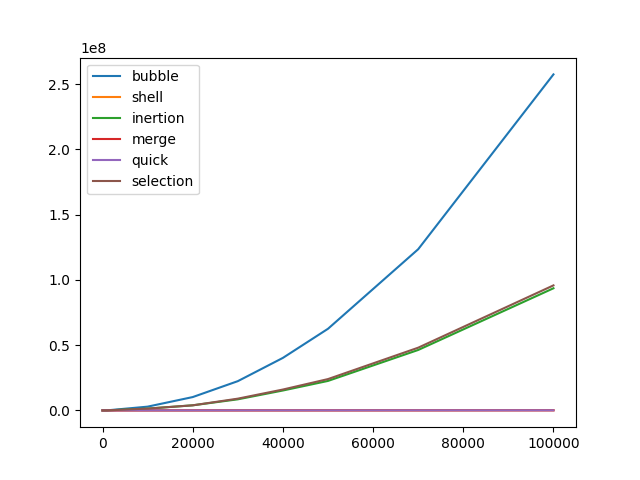


рис 1

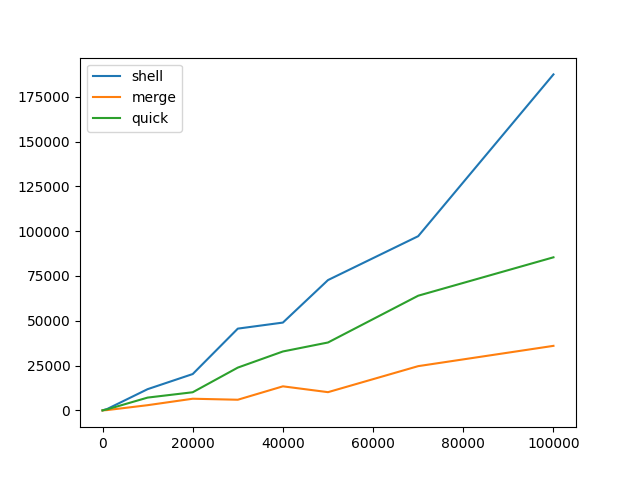


рис 2

На рис. 2 представлено то же самое за исключением квадратичных сортировок.

Таким образом, на произвольных данных реализация сортировки слиянием показала лучший результат.

рис 3

На рис 3 приведён такой же график для сортировки слиянием и функции сортировки std::sort из библиотеки STL. Следовательно реализованная в лабораторной работе сортировка слиянием чуть хуже по скорости работы с библиотечной.

# Текст программы.

Папка lab3. https://github.com/Menoitami/AISD-labs