**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 |  | Ищенко Д.О. |
| Преподаватель |  | Родионова Е. А. |

Санкт-Петербург

2022

Оглавление

[Формулировка задания. 4](#_Toc10771)

[Оценка временной сложности. 4](#_Toc11996)

[Графики. 7](#_Toc28601)

[Текст программы. 10](#_Toc6056)

# 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 | template<class T> void insertionSort(T begin, T end) {  for (auto i = begin + 1; begin != end && i != end; ++i) {  auto j = i;  while (\*j < \*(j - 1)) {  std::swap(\*j, \*(j - 1));  if (j - 1 == begin)  break;  j = (j - 1);  } } } | O(n) | O(n2) | O(n2) | Определяется размером памяти, выделяемым для 4 итераторов: begin, end, i, j. |
| Сортировка выбором | template<class T> void selectionSort(T begin, T end) {  for (T i = begin; begin != end && i != end - 1; ++i)  for (T j = i + 1; j != end; ++j)  if (\*i > \*j)  std::swap(\*i, \*j); } | O(n2) | O(n2) | O(n2) | Определяется размером памяти, выделяемым для 4 итераторов: begin, end, i, j. |
| Пузырьковая сортировка | template<class T> void bubbleSort(T begin, T end) {  for (T i = begin; i != end; ++i)  for (T j = begin; j != end - 1; ++j) {  if (\*j > \*(j + 1))  std::swap(\*j, \*(j + 1));  } } | O(n) | O(n2) | O(n2) | Определяется размером памяти, выделяемым для 5 итераторов: begin, end, i, j. |
| Сортировка слиянием | template<class T> void merge(T lbegin, T mid, T end) {  T rbegin = mid;  while (lbegin != mid && rbegin != end) {  if (\*lbegin > \*rbegin)  std::swap(\*lbegin, \*rbegin);  ++lbegin;  } } template<class T> void mergeSort(T begin, T end) {  if (begin == end || begin == end - 1)  return;   T mid = begin + (end - begin) / 2;  mergeSort(begin, mid);  mergeSort(mid, end);  merge(begin, mid, end); } | O(nlogn) | O(nlogn) | O(nlogn) | Определяется размером памяти, выделяемым для 4 итераторов: begin, end, mid, rbegin. И количеством уровней рекурсии. То есть для худшего случая это размер 4 итераторов\*nlogn. |
| Сортировка Шелла | template<class T> void shellSort(T begin, T end) {  for (auto d = (end - begin) / 2; d != 0; d /= 2)  for (T i = begin + d; i != end; ++i)  for (auto j = i; j - begin >= d && \*j < \*(j - d); j -= d)  std::swap(\*j, \*(j - d)); } | O(nlog2n) | Зависит от выбора шага | O(n2) | Определяется размером памяти, выделяемым для 4 итераторов: begin, end, i, j и переменно d типа [std::iterator\_traits](http://en.cppreference.com/w/cpp/iterator/iterator_traits)<It>::difference\_type. |
| Быстрая сортировка | template<class T> T partition(T begin, T end) {  T pivot = end - 1;  T firsthigh = begin;  for (auto i = begin; i != end - 1; ++i) {  if (\*i < \*pivot) {  std::swap(\*firsthigh, \*i);  ++firsthigh;  }  }  std::swap(\*pivot, \*firsthigh);  return firsthigh; }  template<class T> void quickSort(T begin, T end) {  if (begin != end) {  T pivot = partition(begin, end);   quickSort(begin, pivot);  quickSort(pivot + 1, end);  } } | O(nlogn) | O(nlogn) | O(n2) | Определяется размером памяти, выделяемым для 4 итераторов: begin, end, mid, rbegin. И количеством уровней рекурсии. То есть для худшего случая это размер 4 итераторов \* n2. |

# Графики.

На рис. 1 представлена зависимость скорости работы алгоритма (в микросекундах) от размера входных данных. В качестве входных данных был выбран std::vector<int>.

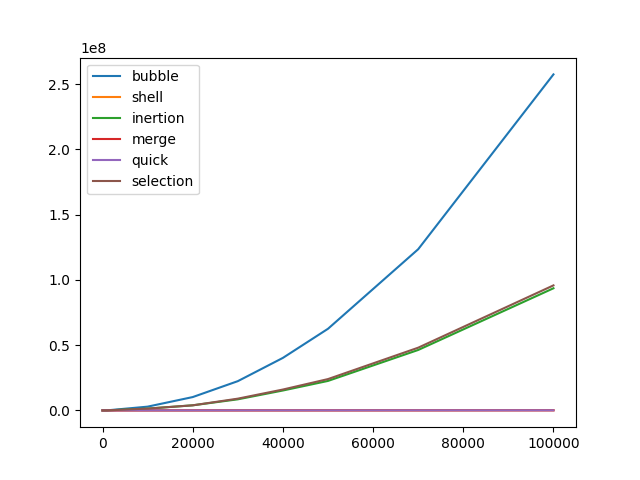


рис 1

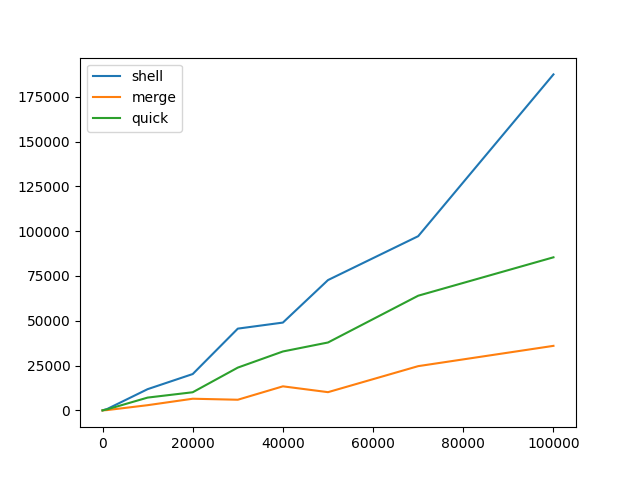
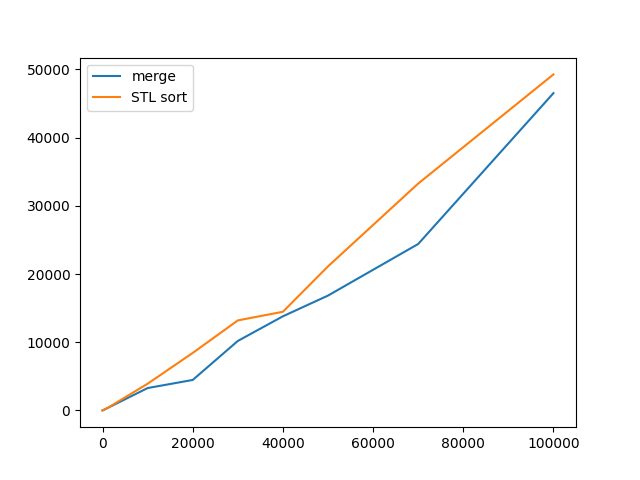


рис 2

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

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



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 сравнима по скорости работы с библиотечной.

Далее представлены графики, показывающие скорость работы для каждой сортировки для лучшего, худшего и рандомного случая. В качестве лучшего случая выступает вектор предварительно отсортированный по возрастанию, в качестве худшего - по убыванию.

|  |  |
| --- | --- |
| bubble | insertion |
| merge | quick |
| selection | shell |

# Текст программы.

[Ссылка на github](https://github.com/Nekttuman/Algosee)

См. папку algos\_lab\_3.