**Федеральное агентство связи**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра Математической Кибернетики и Информационных Технологий



**Отчет по лабораторной работе**

по предмету «СиАОД»

на тему:

«Методы сортировки»

Выполнил: студент группы УБВТ2101 Ярошенко Никита Алексеевич

Руководитель:

Кутейников Иван Алексеевич

Москва 2023

# Цель работы

Реализовать заданный метод сортировки числовой матрицы в соответствии с индивидуальным заданием. Для всех вариантов добавить реализацию быстрой сортировки (quicksort). Оценить время работы каждого алгоритма сортировки и сравнить его со временем стандартной функции сортировки, используемой в выбранном языке программирования.

**Вариант 26**

Сортировка Вставкой (Insertion Sort)

**Ход работы**

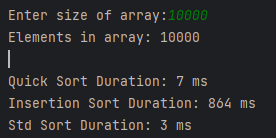
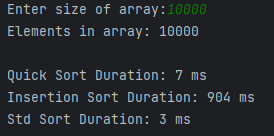
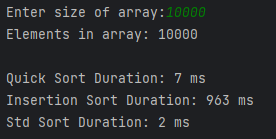
В соответствии с заданием, реализовали алгоритм быстрой сортировки на языке C++:

template <class T, class Compare>  
T Partition(T begin, T end)  
{  
 auto base = std::prev(end, 1);  
 auto i = begin;  
 for (auto j = first; j != base; ++j){  
 if (Compare{}(\*j, \*base)){  
 std::swap(\*i++, \*j);  
 }  
 }  
 std::swap(\*i, \*base);  
 return i;  
}  
  
template <class T, class Compare = std::less<>>  
void QuickSort(T begin, T end) {  
 if (std::distance(begin, end) > 1){  
 T bound = Partition<T, Compare>(begin, end);  
 QuickSort<T, Compare>(begin, bound);  
 QuickSort<T, Compare>(bound + 1, end);  
 }  
}

Сортировка Вставкой:

template <class T, class Compare = std::less<>>  
void InsertionSort(T begin, T end) {  
 for(T i = begin + 1; i < end; ++i) {  
 for(T j = i; j > begin && Compare{}(\*j, \*(j - 1)); --j) {  
 std::swap(\*(j - 1), \*j);  
 }  
 }  
}

Сравнение времени выполнения алгоритмов, для массива размером 10 000:



|  |  |  |  |
| --- | --- | --- | --- |
| n | Insertion Sort, мс | std::sort, мс | Quick Sort, мс |
| 1 | 963 | 2 | 7 |
| 2 | 904 | 3 | 7 |
| 3 | 864 | 3 | 7 |

Среднее время выполнения каждого из алгоритмов:

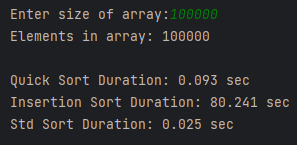
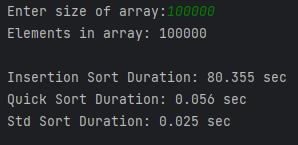
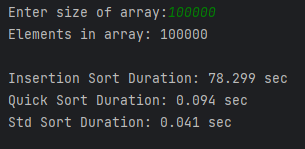
Insertion Sort – 910,3 мс

std::sort – 2.6 мс

Quick sort – 7 мс

Как мы видим, лучший результат показала стандартная функция sort. На втором месте моя реализация Quick Sort. И самый худший результат показала сортировка вставкой.

Сейчас рассмотрим эти алгоритмы с массивом на 100 000 элементов:



|  |  |  |  |
| --- | --- | --- | --- |
| n | Insertion Sort, сек | std::sort, мс | Quick Sort, мс |
| 1 | 78. 299 | 41 | 94 |
| 2 | 80.355 | 25 | 56 |
| 3 | 80.241 | 25 | 93 |

Среднее время выполнения каждого из алгоритмов:

Insertion Sort – 79,6316 мс

std::sort – 30,3 мс

Quick Sort – 81 мс

При большем размере массива, наилучший результат показала встроенная сортировка C++, а наихудший - Insertion Sort.

Обращаем внимание на то, что при сортировках использовались одинаковые массивы случайно сгенерированных чисел.

**Код программы**

#include <vector>  
#include <iostream>  
#include <ctime>  
#include <algorithm>  
  
template <class T, class Compare = std::less<>>  
void InsertionSort(T begin, T end) {  
 for(T i = begin + 1; i < end; ++i) {  
 for(T j = i; j > begin && Compare{}(\*j, \*(j - 1)); --j) {  
 std::swap(\*(j - 1), \*j);  
 }  
 }  
}  
  
template <class T, class Compare>  
T Partition(T begin, T end)  
{  
 auto base = std::prev(end, 1);  
 auto i = begin;  
 for (auto j = first; j != base; ++j){  
 if (Compare{}(\*j, \*base)){  
 std::swap(\*i++, \*j);  
 }  
 }  
 std::swap(\*i, \*base);  
 return i;  
}  
  
template <class T, class Compare = std::less<>>  
void QuickSort(T begin, T end) {  
 if (std::distance(begin, end) > 1){  
 T bound = Partition<T, Compare>(begin, end);  
 QuickSort<T, Compare>(begin, bound);  
 QuickSort<T, Compare>(bound + 1, end);  
 }  
}  
  
int main() {  
 std::vector<int> insert, quick, std\_sort;  
 int N{};  
 std::cout << "Enter size of array:";  
 std::cin >> N;  
 for (int i = 0; i < N; ++i) {  
 int tmp = std::rand();  
 insert.push\_back(tmp);  
 quick.push\_back(tmp);  
 std\_sort.push\_back(tmp);  
 }  
 std::cout << "Elements in array: " << N << std::endl << std::endl;  
  
 std::clock\_t start = std::clock();  
 QuickSort<std::vector<int>::iterator>(quick.begin(), quick.end());  
 std::cout << "Quick Sort Duration: " << (std::clock() - start) << " ms" << std::endl;  
  
 start = std::clock();  
 InsertionSort<std::vector<int>::iterator>(insert.begin(), insert.end());  
 std::cout << "Insertion Sort Duration: " << (std::clock() - start) << " ms" << std::endl;  
  
 start = std::clock();  
 std::sort<std::vector<int>::iterator>(std\_sort.begin(), std\_sort.end());  
 std::cout << "Std Sort Duration: " << (std::clock() - start) << " ms" << std::endl;  
}

**Вывод**

Результаты во всех тестах однозначные. Самый худший результат показывает алгоритм сортировки вставкой, затем идет алгоритм быстрой сортировки и самый быстрый алгоритм стандартной библиотеки sort. Сложность алгоритма сортировки вставкой O(n^2), сложность алгоритма быстрой сортировки O(n\*log2n). А сложность алгоритма встроенной сортировки могу предположить по итогам тестов O(n\*log2n/2).