МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Алгоритмы и структуры данных»
Тема: Сортировки.

Студент гр. 9384 Звега А.Р.

Преподаватель Ефремов М.А.

Санкт-Петербург
2020

Цель работы.

Реализовать сортировку и сравнить с другими возможными.

Задание.

ВАРИАНТ 6.

Реализовать бинго сортировку.

Выполнение работы.

Бинго сортировка является модификацией сортировки выбором. Массив обходится 1 раз для поиска максимального элемента. Затем производится проверка на упорядоченность, если массив уже отсортирован или частично отсортирован, то индекс крайней позиции(стена) смещается. Основной алгоритм сортировки обходит весь массив и сравнивает все значения с максимальным и следующим максимальным, если находится элемент равный максимальному, то он меняется местами с элементом стены, затем стена смещается. Если встречается элемент, который больше максимального следующего, то он становится максимальным следующим. Алгоритм повторяется пока массив не будет отсортирован. Так же в конце каждой итерации производится проверка на упорядоченность.

Сравнения данной сортировки с другими представлены в табл. 1.

Таблица 1 – Сравнения сортировок

Название сортировки	Лучшее время	Худшее время
Бинго сортировка	O(n+m^2)	O(nm)
Сортировка выбором	O(n^2)	O(n^2)
Быстрая сортировка	O(nlogn)	O(n^2)

Тестирование.

Результаты тестирования представлены в табл. 2.

Таблица 2 – Результаты тестирования

Входные данные	Выходные данные
4 8 9 3	3 4 8 9
3 2 1 4	1 2 3 4
5 5 5 5 5 6 6 6	5 5 5 5 5 6 6 6

Выводы.

В ходе выполнения лабораторный работы была реализована бинго сортировка.

ПРИЛОЖЕНИЕ
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iterator>
#include <functional>

using namespace std;

template<typename T>
auto print(const char* msg, T& vec) {
   cout << msg;
   copy(vec.cbegin(), vec.cend(), ostream_iterator<int>{cout, " "});
   cout << endl;
};

template<typename T>
bool cmp_equal(T a, T b)
{
   return a == b;
}

template<typename T>
bool cmp_more(T a, T b)
{
   return a > b;
}

template<typename T>
T& bingo(T &data, function<bool(typename T::value_type, typename T::value_type)> more,
function<bool(typename T::value_type, typename T::value_type)> equal)
{
   typename T::size_type max = data.size() - 1;
   typename T::value_type nextValue = *max_element(data.begin(), data.end());
   while (max && equal(data[max], nextValue) && max--) {
      cout << "max-- because items are already sorted" << endl;
   }
   cout << "First search for max element" << endl;
   cout << nextValue << endl;
   cout << endl;
   while (max)
   {
      typename T::value_type value = nextValue;
      nextValue = data[max];
      for_each(data.rbegin() + (data.size() - max - 1), data.rend(),
         [&data, &max, &value, &nextValue, more , equal](typename T::value_type& it)
         {
            if (equal(it,value))
            {
               cout << "swap curent item with data[max]" << endl;
               cout << "because curent item " << it << " == max value " << value << endl;
```

```
                swap(it, data[max]);
                --max;
                cout << "max-- because items after data[max] are already sorted" << endl;
                cout << endl;
            }
            else if (more(it,nextValue)){
                cout << "change next value\nbecause value > next value" << endl;
                cout << endl;
                nextValue = it;
            }
        });
    while (max && equal(data[max], nextValue) && max--){
        cout << "max-- because items are already sorted" << endl;;
    }
  }
  return data;
}

int main()
{
  function<bool(int, int)> more = cmp_more<int>;
  function<bool(int, int)> equal = cmp_equal<int>;
  vector<int> vec{ 8, 9, 1, 6, 11, 4, 3, 14, 1 };
  print("Before: ", vec);
  bingo(vec, more, equal);
  print("After:  ", vec);
  return 0;
}
```