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. 9382 _____ Кодуков А.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы:

Познакомиться с одним из часто используемых на практике алгоритмов сортировки данных, оценить его достоинства и недостатки.

Задание:

Индивидуальное задание 8:

Быстрая сортировка, рекурсивная реализация. Во время сортировки массив должен быть в состоянии:

элементы <x, неотсортированные элементы, элементы >=x.

Описание алгоритма:

Быстрая сортировка:

Данная сортировка использует технику “разделяй и властвуй”.

Сначала выбирается *опорный элемент*. В данной реализации он выбирается как элемент из середины массива, чтобы ускорить работу на уже отсортированных массивах.

Затем необходимо, чтобы все элементы меньшие опорного элемента оказались слева от него, а большие – справа. Для этого заводятся два указателя: на начало и на конец массива. Затем левый указатель сдвигается вправо по массиву, пока не встретит элемент больше опорного или опорный, а правый аналогичным образом сдвигается влево для обнаружения элемента меньше опорного или опорного. Когда оба указателя установлены на необходимые места, соответствующие элементы меняются местами. Данные действия повторяются пока указатели не зайдут друг за друга на опорном элементе. После этого опорный элемент установлен на свое место в отсортированном массиве.

Далее необходимо отсортировать подмассивы слева и справа от него. Подразбиение будет продолжаться до массива из одного элемента. Когда алгоритм закончит работу, все элементы побывают опорными, а значит будут установлены на свои места и массив будет отсортирован.

Достоинства:

- Один из самых быстродействующий в среднем случае алгоритмов сортировки общего назначения
- Использует сравнительно мало памяти
- Довольно короткая и простая в реализации

Недостатки:

- Скорость может сильно ухудшится при неудачных входных данных (механизм выбора опорного элемента)
- Рекурсивная реализация может вызвать переполнение стека при неудачных входных данных

Функции и структуры данных:

Реализованные функции:

Быстрая сортировка

Сигнатура: `void QuickSort(int *A, int size, int lvl)`

Аргументы:

- A –указатель на первый элемент текущего массива
- size – размер текущего массива
- lvl – уровень рекурсии

Алгоритм:

- Выбор опорного элемента как элемента из середины массива
- Пока итерация не завершена
    - Сдвинуть левый указатель до элемента не больше опорного
    - Сдвинуть правый указатель до элемента не меньше опорного
    - Если левый указатель больше правого – закончить итерацию
    - Иначе, поменять элементы под левым и правым указателем местами
- Запустить алгоритм от получившихся левого и правого подмассивов

Тестирование:

№	Входные данные	Результат
1	10 11 23 2 45 -30 4 99 7 -100 8 51 9 1 -123 5 41	Sorted: -123 -100 -30 1 2 4 5 7 8 9 10 11 23 41 45 51 99 Control: -123 -100 -30 1 2 4 5 7 8 9 10 11 23 41 45 51 99 Sorting is correct
2	1 1 1 1 1 1 1 1	Sorted: 1 1 1 1 1 1 1 1 Control: 1 1 1 1 1 1 1 1 Sorting is correct
3	1 1 2 1 2 3 2 2 1 1 3	Sorted: 1 1 1 1 1 2 2 2 2 3 3 Control: 1 1 1 1 1 2 2 2 2 3 3 Sorting is correct
4	5 4 3 2 1 0 -1 -2 -3 -4 -5	Sorted: -5 -4 -3 -2 -1 0 1 2 3 4 5 Control: -5 -4 -3 -2 -1 0 1 2 3 4 5 Sorting is correct
5	0 1 0 1 0 1 0 1 0 1	Sorted: 0 0 0 0 0 1 1 1 1 1 Control: 0 0 0 0 0 1 1 1 1 1 Sorting is correct
6	--1	Wrong input
7	asd	Wrong input
8	1a1	Wrong input

Вывод:

В результате выполнения работы был изучен и реализован алгоритм быстрой сортировки, а также выявлены его достоинства и недостатки.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

void PrintArray(int *A, int size, int left, int right, int lvl) {
  for (int i = 0; i < lvl; i++) std::cout << "  ";
  for (int k = 0; k < size; k++)
    std::cout << (k == left ? "|" : "") << A[k] << (k == right ? "| " : " ");
  if (right == size) std::cout << "|";
  std::cout << "\n";
}

void QuickSort(int *A, int size, int lvl) {
  long i = 0, j = size - 1;  // set pointers
  int temp, p;

  p = A[size / 2];  // pivot element

  for (int i = 0; i < lvl; i++) std::cout << "  ";
  std::cout << "Pivot: " << p << "\n";

  bool done = false;
  while (!done) {
    // move left pointer
    while (i <= j && A[i] < p) {
      PrintArray(A, size, i, j, lvl);
      i++;
    }
    // move right pointer
    while (i <= j && A[j] > p) {
      PrintArray(A, size, i, j, lvl);
      j--;
    }

    for (int k = 0; k < lvl; k++) std::cout << "  ";
    for (int k = 0; k < size; k++)
      std::cout << (k == i ? "|" : "") << A[k] << (k == j ? "| " : " ");

    if (A[i] != A[j] && i != j) {
      if (j < i) {
        std::cout << "\n";
        done = true;
      }
      // swap elements
      else {
        std::cout << "swap(" << A[i] << ", " << A[j] << ")\n";
        temp = A[i];
        A[i] = A[j];
        A[j] = temp;
      }
    } else {
      j--, i++;
      done = true;
      std::cout << "\n";
    }
  }
```

```
  // sort left subarray
  if (j > 0) QuickSort(A, j + 1, lvl + 1);
  // sort right subarray
  if (size > i) QuickSort(A + i, size - i, lvl + 1);
}

std::vector<int> Read(std::istream &s) {
  std::vector<int> v;
  std::string str;
  std::getline(s, str);
  int n = 0;
  bool passed = false;
  bool neg = false;
  for (auto &ch : str) {
    if (ch == '-') {
      if (!passed) {
        std::cout << "Wrong input";
        return std::vector<int>({});
      }
      neg = true;
    }
    else if (ch == ' ') {
      if (neg) n *= -1;
      v.push_back(n);
      passed = true;
      neg = false;
      n = 0;
    } else if (ch <= '9' && ch >= '0'){
      n = n * 10 + ch - '0';
      passed = false;
    }
    else {
      std::cout << "Wrong input";
      return std::vector<int>({});
    }
  }
  if (!passed) {
    if (neg) n *= -1;
    v.push_back(n);
  }
  return v;
}

int main() {
  std::vector<int> v;
  bool m;
  while (1) {
    char mode;

    system("cls");
    std::cout << "Choose mode:\n1 - console input\n2 - file input\n";
    mode = getchar();
    if (mode == '1') {
      m = false;
      break;
    } else if (mode == '2') {
      m = true;
      break;
    } else {
      std::cout << "Wrong option";
      getchar();
    }
  }
```

```
  std::cin.clear();
  std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
  system("cls");
  if (m) {
    char s[200];
    std::ifstream f("Tests/input.txt");
    f.getline(s, 200);
    if (f.is_open()) {
      std::cout << "File input: " << s;
      f.seekg(0, std::ios_base::beg);
      v = Read(f);
      f.close();
    }
  } else {
    v = Read(std::cin);
  }
  std::cout << "\n";
  if (!v.empty()) {
    std::vector<int> v1 = v, v2 = v;
    QuickSort(v1.data(), v1.size(), 0);
    std::cout << "\nSorted:  ";
    PrintArray(v1.data(), v1.size(), -1, -1, 0);
    std::sort(v2.begin(), v2.end());
    std::cout << "Control: ";
    PrintArray(v2.data(), v2.size(), -1, -1, 0);
    if (v1 == v2)
      std::cout << "Sorting is correct";
    else
      std::cout << "Wrong sorting";
  }
  getchar();
  return 30;
}
```