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: Алгоритмы сортировки

Студентка гр. 9303 _____ Отмахова М.А.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург
2020

Цель работы.

Написать программу, сортирующую целочисленный массив с помощью алгоритма быстрой сортировки.

Задание.

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

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

Описание алгоритма

Оценка сложности алгоритма

Лучший случай. В наиболее сбалансированном варианте при каждой операции разделения массив делится на две одинаковые (плюс-минус один элемент) части, следовательно, максимальная глубина рекурсии, при которой размеры обрабатываемых подмассивов достигнут 1, составит logn. В результате количество сравнений, совершаемых быстрой сортировкой, что даёт общую сложность алгоритма O(nlogn).

Худший случай. В самом несбалансированном варианте каждое разделение даёт два подмассива размерами 1 и n-1, то есть при каждом рекурсивном вызове больший массив будет на 1 короче, чем в предыдущий раз. Такое может произойти, если в качестве опорного на каждом этапе будет выбран элемент либо наименьший, либо наибольший из всех обрабатываемых. При простейшем выборе опорного элемента — первого или последнего в массиве, — такой эффект даст уже отсортированный (в прямом или обратном порядке) массив, для среднего или любого другого фиксированного элемента «массив худшего случая» также может быть специально подобран. В этом случае потребуется n-1 операций разделения, а общее время работы составит O(n*n) операций, то есть сортировка будет выполняться за квадратичное время. Но количество обменов и, соответственно, время работы — это не самый большой его недостаток. Хуже то, что в таком случае

глубина рекурсии при выполнении алгоритма достигнет n, что будет означать n-кратное сохранение адреса возврата и локальных переменных процедуры разделения массивов. Для больших значений n худший случай может привести к исчерпанию памяти (переполнению стека) во время работы программы.

Достоинства алгоритма:

- Один из самых быстродействующих (на практике) из алгоритмов внутренней сортировки общего назначения.
- Алгоритм очень короткий: запомнив основные моменты, его легко написать «из головы», невелика константа при nlogn.
- Требует лишь O(logn) дополнительной памяти для своей работы. (Не улучшенный рекурсивный алгоритм в худшем случае O(n) памяти)
- Хорошо сочетается с механизмами кэширования и виртуальной памяти.
- Допускает естественное распараллеливание (сортировка выделенных подмассивов в параллельно выполняющихся подпроцессах).
- Допускает эффективную модификацию для сортировки по нескольким ключам (в частности — алгоритм Седжвика для сортировки строк): благодаря тому, что в процессе разделения автоматически выделяется отрезок элементов, равных опорному, этот отрезок можно сразу же сортировать по следующему ключу.
- Работает на связных списках и других структурах с последовательным доступом, допускающих эффективный проход как от начала к концу, так и от конца к началу.

Недостатки алгоритма:

- Сильно деградирует по скорости (до O(n^2)) в худшем или близком к нему случае, что может случиться при неудачных входных данных.

- Прямая реализация в виде функции с двумя рекурсивными вызовами может привести к ошибке переполнения стека, так как в худшем случае ей может потребоваться сделать O(n) вложенных рекурсивных вызовов.
- Неустойчив.

Выполнение работы.

В ходе выполнения лабораторной работы была написана рекурсивная функция quickSort(), сортирующая массив целых чисел по возрастанию. На вход данная функция принимает массив чисел, два индекса – левый и правый (int right, int left), а также общее количество элементов в массиве.

Разрешающим элементом делаем первый (нулевой) элемент массива. Далее устанавливаем left на следующий за ним, a right – на самый последний. “Указатель” left движется вправо, пока не встретится элемент больше разрешающего pivot, a right движется влево, пока не встретится элемент меньше pivot. Элементы меняются местами. Движение указателей продолжается. Данные шаги продолжаются, пока right не станет меньше left. Тогда меняются местами разрешающий элемент и элемент, на который указывает right. Таким образом, разрешающий элемент оказывается на своем месте, слева от него – элементы меньше его, а справа – больше. Далее функция сортировки рекурсивно вызывается для левого и правого подмассивов.

Исходный код программы представлен в приложении А.

Тестирование программы представлено в приложении Б.

Выводы.

В ходе выполнения лабораторной работы был изучен алгоритм быстрой сортировки, а также написана программа, реализующая алгоритм быстрой сортировки на языке C++.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>

using namespace std;

void quickSort(int* num, int left, int right, int n)
{
    cout << "quickSort called for " << left << " to "
<< right << " : ";
    for( int i = 0; i < n; i++) {
        cout << num[i] << ' ';
    }
    cout << '\n';
    int pivot;
    int l_hold = left;
    int r_hold = right;
    pivot = num[left];
    while (left < right)
    {
        while ((num[right] >= pivot) && (left <
right))
            right--;
        if (left != right)
        {
            num[left] = num[right];
            left++;
        }
        while ((num[left] <= pivot) && (left <
right))
            left++;
        if (left != right)
        {
            num[right] = num[left];
            right--;
        }
    }
    num[left] = pivot;
    pivot = left;
    left = l_hold;
    right = r_hold;

    if (left < pivot) {
        quickSort(num, left, pivot - 1, n);
```

```
    }
    if (right > pivot) {
        quickSort(num, pivot + 1, right, n);

    }
}

int main() {
    int n;
    cout << "enter the number of array elements: " <<
endl;
    cin >> n;
    int* num = new int[n];
    for( int i = 0; i < n; i++) {
        cin >> num[i];
    }

    quickSort(num, 0, n-1, n);

    cout << "sorted array: " << endl;
    for( int i = 0; i < n; i++) {
        cout << num[i] << ' ';
    }
    delete[] num;
    return 0;
}
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ ПРОГРАММЫ

Входные данные	Результат работы программы
6 5 3 4 2 8 1	quickSort called for 0 to 5 : 5 3 4 2 8 1 quickSort called for 0 to 3 : 1 3 4 2 quickSort called for 1 to 3 : 3 4 2 quickSort called for 1 to 1 : 2 quickSort called for 3 to 3 : 4 quickSort called for 5 to 5 : 8 sorted array: 1 2 3 4 5 8
7 84 23 849 46 17 1 757	quickSort called for 0 to 6 : 84 23 849 46 17 1 757 quickSort called for 0 to 3 : 1 23 17 46 quickSort called for 1 to 3 : 23 17 46 quickSort called for 1 to 1 : 17 quickSort called for 3 to 3 : 46 quickSort called for 5 to 6 : 849 757 quickSort called for 5 to 5 : 757 sorted array: 1 17 23 46 84 757 849
10 577 465 254 87 36 123 56 37 9 58	quickSort called for 0 to 9 : 577 465 254 87 36 123 56 37 9 58 quickSort called for 0 to 8 : 58 465 254 87 36 123 56 37 9 quickSort called for 0 to 3 : 9 37 56 36 quickSort called for 1 to 3 : 37 56 36 quickSort called for 1 to 1 : 36 quickSort called for 3 to 3 : 56 quickSort called for 5 to 8 : 123 87 254 465 quickSort called for 5 to 5 : 87 quickSort called for 7 to 8 : 254 465 quickSort called for 8 to 8 : 465 sorted array: 9 36 37 56 58 87 123 254 465 577
4 1 5 1 5	quickSort called for 0 to 3 : 1 5 1 5 quickSort called for 1 to 3 : 5 1 5 quickSort called for 1 to 1 : 1 quickSort called for 3 to 3 : 5 sorted array: 1 1 5 5