

МИНОБРНАУКИ РОССИИ

*Федеральное государственное бюджетное образовательное учреждение высшего образования*

***«МИРЭА – Российский технологический университет»***

Отчет

Практическая работа №8

Дисциплина Структуры и алгоритмы обработки данных

Тема. Определение эффективного алгоритма сортировки

|  |  |
| --- | --- |
| Выполнил студент | Смольников А.Б. |
|  | Фамилия И.О. |
| Группа | ИКБО-13-21 |
|  | Номер группы |

**Москва 2022**

**Задание 1**

**Вариант №20**

**Оценить зависимость времени выполнения алгоритма простой сортировки на массиве, заполненном случайными числами (средний случай)**

1. Разработать алгоритм сортировки одномерного целочисленного массива A[n] и реализовать его функцией, используя алгоритм согласно варианту, индивидуального задания – *сортировка простого выбора.* Провести тестирование программы на исходном массиве, сформированном вводом с клавиатуры, т.е. доказать ее работоспособность.
2. Разработать функцию заполнения рабочего массива A с использованием генератора псевдослучайных чисел.
3. Провести экспериментальную оценку вычислительной сложности алгоритма, для чего выполнить контрольные прогоны программы для размеров массива n = 100, 1000, 10000, 100000 и 200000 элементов с вычислением времени T(n) выполнения T(n) – (в миллисекундах/секундах). Полученные результаты свести в сводную таблицу Таблица 1.
4. Провести эмпирическую оценку вычислительной сложности алгоритма, определив функцию зависимости времени выполнения алгоритма от размера массива(задачи) и показать ее результат в таблице 1 в столбце **Тэт** =f(C+M).
5. Провести эмпирическую оценку фактического количества операций сравнения Сф и количества операций перемещения Мф. Полученные результаты вставить в сводную таблицу в столбец Тэп=Сф+Мф.

**Разработка алгоритма и его реализация**

Необходимо разработать алгоритм сортировки целочисленного массива методом простого выбора.

Алгоритм: Сортировка методом простого выбора строится на идее сравнивания минимального элемента с элементами, которые стоят после него. Если нашелся элемент, меньше локального минимума, обновляется локальный минимум и поменяем местами с элементом бывшего локального минимума.

Внешним циклом пройдем от нулевого до n-1 элемента. В переменную min\_id запишем индекс i. В теле внешнего цикла создадим внутренний цикл от элемента i-1 до n-того элемента. Если элемент массива с индексом j меньше элемента массива с индексом min\_id, обновим min\_id. Поменяем местами arr[min\_id] и arr[i].

Определим «О-большое» данная характеристика определяет временную эффективность алгоритма в зависимости от количества входных данных. О-большое для данной сортировки составит:

Так как в худшем случае придется рассмотреть все элементы в двумерном цикле.

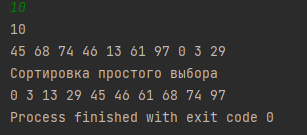
Рассмотрим код функции сортировки:

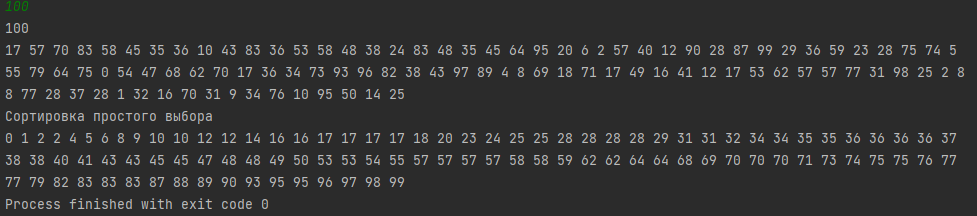
Предусловие – массив целых чисел arr, размерность n

Постусловие – отсортированный массив

1. void selectionSort(int \*arr, int n)  
   {  
    for (int i = 0; i < n - 1; i++)  
    {  
    int min\_id = i;  
     
    for (int j = i + 1; j < n; j++)  
    if (arr[j] < arr[min\_id])  
    min\_id = j;  
     
    int temp = arr[min\_id];  
    arr[min\_id] = arr[i];  
    arr[i] = temp;  
    //if(i%1000==0) cout<<"Selection sort: "<<i<<"/"<<n<<endl;  
    }  
   }

Рассмотрим тестовый пример для 10 и 100 элементов:





Определим зависимость времени от количества входных данных эмпирически.

|  |  |  |  |
| --- | --- | --- | --- |
| Номер оператора | Оператор | Время выполнения оператора | Количество выполнений |
| 1 | for (int i = 0; i < n-1; i++) | C1 | n |
| 2 | int min\_id = i; | C2 | n-1 |
| 3 | for(int j=i+1; j<n; j++) | C3 | n |
| 4 |  | C2 | n  ∑(n-j)  j=1 |
| 5 |  | C2 | n  ∑(n-j-1)  j=1 |
| 6 |  | C3 | n  ∑(n-j-1)  j=1 |
| 7 |  | C2 | n-1 |

***Таблица 1. Сводная таблица результатов***

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **T(n) время в мс** | **Тэт=f(C+M)-функция** | **Т**эп**=Cф+Mф- количество** |
| 100 | 0.3998 |  |  |
| 1000 | 2.2904 |  |  |
| 10000 | 120.383 |  |  |
| 100000 | 12001.5 |  |  |
| 1000000 |  |  |  |