Лабораторная работа 12. Алгоритмы сортировки

1. ЦЕЛЬ РАБОТЫ

Изучение и сравнение алгоритмов сортировки данных.

2. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Многие эффективные алгоритмы основаны на использовании упорядоченного набора данных, т.к. благодаря сортировке данных задачу часто можно решить проще.

Основная задача сортировки формулируется следующим образом: дан массив из n элементов, требуется расположить элементы в порядке возрастания.

Легко спроектировать алгоритм с временной сложностью $O(n^2)$, но существуют и более эффективные алгоритмы. Наилучше варианты дают временную сложность $O(n \times \log n)$.

1. Обменная сортировка

Если два элемента расположены не по порядку, то они меняются местами. Этот процесс повторяется до тех пор, пока элементы не будут упорядочены ("пузырьковая" сортировка, так как большие элементы, подобно пузырькам "всплывают" вверх (вправо) на соответствующую позицию).

2. Сортировка выбором

Сначала выделяется наименьший (или наибольший элемент) и каким-либо образом отделяется от остальных, затем выбирается наименьший (наибольший) из оставшихся и т.д.

Разберем этот алгоритм подробнее. Будем сортировать элементы по возрастанию. Пусть начальный массив из 10 элементов следующий:

3 8 17 15 7 19 17 1 18 7

Минимальный элемент (с 0 по 9) – 1. Поставим его на место нулевого, а нулевой элемент на место единицы:

1 8 17 15 7 19 17 3 18 7

Затем найдем минимальный элемент с 1 по 9. Это 3. Поменяем его местами с первым:

1 3 17 15 7 19 17 8 18 7

Затем найдем минимальный элемент со 2 по 9. Это 7. Поменяем ее местами со вторым элементом:

1 3 7 15 17 19 17 8 18 7

Выполним сортировку до конца (см. рис. с результатами работы программы):

1 8 17 15 7 19 17 3 18 7

1 3 17 15 7 19 17 8 18 7

1 3 7 15 17 19 17 8 18 7

1 3 7 7 17 19 17 8 18 15

1 3 7 7 8 19 17 17 18 15

1 3 7 7 8 15 17 17 18 19

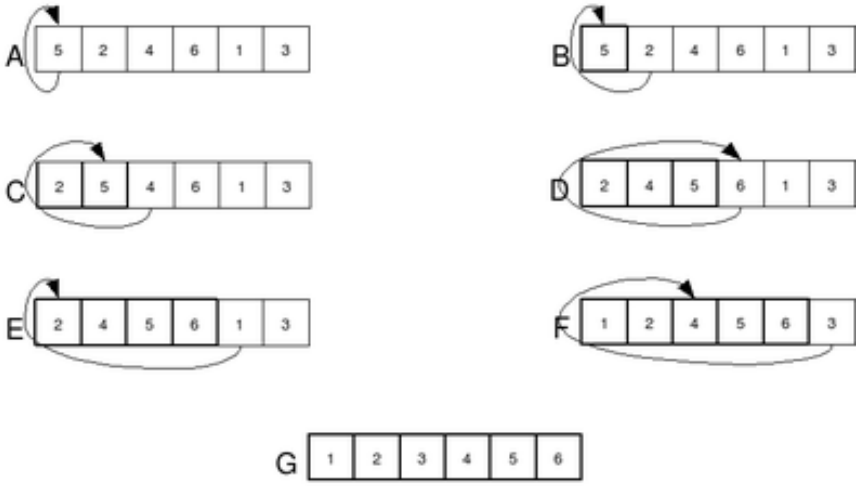
1 3 7 7 8 15 17 17 18 19

1 3 7 7 8 15 17 17 18 19

1 3 7 7 8 15 17 17 18 19

3. Сортировка вставками

Сортировка вставками — алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.



На вход процедуре сортировки подаётся массив A, состоящий из элементов последовательности A[1], A[2], …, которые требуется отсортировать. n соответствует размеру исходного массива.

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма.

Данный алгоритм можно ускорить при помощи использования бинарного поиска для нахождения места текущему элементу в отсортированной части. Проблема с долгим сдвигом массива вправо решается при помощи смены указателей

Для сортировки не требуется привлечения дополнительной памяти, кроме постоянной величины для одного элемента, так как выполняется перестановка в пределах массива.

4. Сортировка перемешиванием (шейкерная сортировка)

Шейкерная сортировка отличается от пузырьковой тем, что она двунаправленная: алгоритм перемещается не строго слева направо, а сначала слева направо, затем справа налево.

5. Сортировка расчёской

Сортировка расчёской — улучшение сортировки пузырьком. Её идея состоит в том, чтобы «устранить» элементы с небольшими значения в конце массива, которые замедляют работу алгоритма. Если при пузырьковой и шейкерной сортировках при переборе массива сравниваются соседние элементы, то при «расчёсывании» сначала берётся достаточно большое расстояние между сравниваемыми значениями, а потом оно сужается вплоть до минимального.

Первоначальный разрыв нужно выбирать не случайным образом, а с учётом специальной величины — фактора уменьшения, оптимальное значение которого равно 1,247. Сначала расстояние между элементами будет равняться размеру массива, поделённому на 1,247; на каждом последующем шаге расстояние будет снова делиться на фактор уменьшения — и так до окончания работы алгоритма.

6. ???

Выходной массив заполняется значениями -1. Затем для каждого элемента определяется его место в выходном массиве путем подсчета количества элементов строго меньших данного. Естественно, что все одинаковые элементы попадают на одну позицию, за которой следует ряд значений -1. После этого оставшиеся в выходном массиве позиции со значением -1 заполняются копией предыдущего значения.

7. Сортировка подсчетом

Алгоритм подходит не для всех видов входных данных.

В этом алгоритме используется диапазон чисел сортируемого массива для подсчёта совпадающих элементов.

Применение сортировки подсчётом целесообразно лишь тогда, когда сортируемые числа имеют (или их можно отобразить в диапазон возможных значений, который достаточно мал по сравнению с сортируемым набором элементов, например, миллион натуральных чисел меньших 1000.

Предположим, что входной массив состоит из n целых чисел в диапазоне от 1 до k, где $k \in N$.

Исходная последовательность чисел длины n хранится в массиве A. Также используется вспомогательный массив C с индексами от 1 до k, изначально заполняемый нулями.

Последовательно пройдём по массиву A и запишем в C[i] количество чисел, равных i.

Теперь достаточно пройти по массиву C и для каждого number$\in${1,...,k} в массив A последовательно записать число C[number] раз.

<u>Пример</u>:

Исходный массив A из n=10 элементов:

3	**2**	**1**	**2**	**2**	**5**	**6**	**1**	**5**	**3**

Элементы массива A находятся в диапазоне от 1 до 6.

Исходный массив C из 6 элементов «0»:

Индекс	1	2	3	4	5	6
Значение	**0**	**0**	**0**	**0**	**0**	**0**

Последовательно пройдём по массиву A и запишем в C[i] количество чисел, равных i:

Индекс	1	2	3	4	5	6
Значение	**2**	**3**	**2**	**0**	**2**	**1**

Теперь достаточно пройти по массиву C и для каждого number$\in${1,...,6} в массив A последовательно записать число C[number] раз.

number = 1:

1	**1**								

number = 2:

1	**1**	**2**	**2**	**2**					

number = 3:

1	**1**	**2**	**2**	**2**	**3**	**3**			

и т.д.

….

number = 6:

1	**1**	**2**	**2**	**2**	**3**	**3**	**5**	**5**	**6**

В алгоритме первые два цикла работают за ***O*** (k) и ***O*** (n), соответственно; двойной цикл за ***O*** (n+k).

Алгоритм имеет линейную временную трудоёмкость ***O***(n+k).

Используемая дополнительная память равна ***O***(k).

Возникает несколько вопросов.

1. Что делать, если диапазон значений (*min* и *max*) заранее не известен?

2. Что делать, если минимальное значение больше нуля или в сортируемых данных присутствуют отрицательные числа?

Первый вопрос можно решить линейным поиском *min* и *max*, что не повлияет на асимптотику алгоритма. Второй вопрос несколько сложнее. Если *min* больше нуля, то следует при работе с массивом `C` из `A[i]` вычитать *min*, а при обратной записи прибавлять. При наличии отрицательных чисел нужно при работе с массивом `C` к `A[i]` прибавлять |*min*|, а при обратной записи вычитать.

8. Сортировка слиянием

Алгоритм использует принцип «разделяй и властвуй»: задача разбивается на подзадачи меньшего размера, которые решаются по отдельности, после чего их решения комбинируются для получения решения исходной задачи. Конкретно процедуру сортировки слиянием можно описать следующим образом:

- Если в рассматриваемом массиве один элемент, то он уже отсортирован — алгоритм завершает работу.
- Иначе массив разбивается на две части, которые сортируются рекурсивно.
- После сортировки двух частей массива к ним применяется процедура слияния, которая по двум отсортированным частям получает исходный отсортированный массив.

9. Быстрая сортировка

Этот алгоритм состоит из трёх шагов. Сначала из массива нужно выбрать один элемент — его обычно называют опорным. Затем другие элементы в массиве перераспределяют так, чтобы элементы меньше опорного оказались до него, а большие или равные — после. А дальше рекурсивно применяют первые два шага к подмассивам справа и слева от опорного значения.

Функция `std::sort`

На практике обычно нет смысла писать свой алгоритм/функцию сортировки, т.к. во всех языках есть готовые решения. Преимущества: работают быстрее (примерно ***O(n·log n)***), работают правильно.

C++:

Функция `std::sort`

Определена в заголовочном файле `<algorithm>`

```
void sort( first, last );
void sort( first, last, comp );
```

Сортировка элементов в диапазоне `[first, last)` в порядке возрастания.

Параметры

first, last	—	диапазон элементов, подлежащих сортировке
cmp	—	функция сравнения, возвращающая `true` если первый аргумент *меньше* второго.

Примеры:

1)
```
vector<int> v  = {5, 8, 1, 15, -3, 4};
```

```
sort(v.begin(), v.end());
```

2)
```
#include <bits/stdc++.h>
using namespace std;
int main()
{ int arr[] = {1, 5, 8, 9, 6, 7, 3, 4, 2, 0};
  int n = sizeof(arr)/sizeof(arr[0]);

  sort(arr, arr+n);

  cout << "\nArray after sorting is : \n";
  for (int i = 0; i < n; ++i)
         cout << arr[i] << " ";
  return 0;
}
```

Как замерить время выполнения программы?

Самый простой вариант — это посчитать разницу между начальным временем и конечным. То есть, есть начальное значение времени, после которого объявлен фрагмент кода, время выполнения которого необходимо измерить. После фрагмента кода фиксируется ещё одно, конечное, значение времени. После чего, из конечного значения времени вычитаем начальное время и получим время, за которое выполнился измеряемый фрагмент кода или вся программа.

C++:

```
// заголовочный файл с прототипом функции clock()
#include <ctime>
//          ...
std::clock_t c_start = std::clock();
// Наш алгоритм, время работы которого замеряем
std::clock_t c_end = std::clock();
cout <<(c_end-c_start)/ CLOCKS_PER_SEC <<" s\n"; //время
   работы пр-мы
```

3. ЗАДАНИЕ НА РАБОТУ

1.Реализовать заданный алгоритм сортировки. Проверить его работу на разных наборах входных данных.

2. Реализовать сортировку данных с помощью функции *sort*.

3. Создаем программу по алгоритму:

- Заполняем массив из N целых чисел случайными значениями в диапазоне от 0 до 1000.
- Выводим.
- Засекаем первый момент времени T1.

- Упорядочиваем массив алгоритмом, заданным по варианту.

- Засекаем второй момент времени.

- Выводим (для контроля), можно в файл. Для больших значений можно не выводить.

4. Заполняем таблицу.

Исследование алгоритмов сортировки

Количество чисел N	Алгоритм по варианту	Стандартная (из библиотеки алгоритмов)
100	*0,0001 с*	
1000		
10 000		
100 000		
1 000 000		

4. ОФОРМЛЕНИЕ ОТЧЕТА

Отчет по работе должен содержать:

· название и цель работы;

· вариант задания;

· описание алгоритма сортировки по варианту, код программ(ы), результаты тестирования алгоритмов (таблица), оценка разработанного алгоритма по своему варианту по сложности по времени и памяти, выводы.

5. БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Шилдт Г. C++: базовый курс, 3-е издание. : Пер. с англ. – М.: «Издательский дом «Вильямс», 2005. – 624 с.

2. Пахомов Б.И. C/C++ и MS Visual C++ для начинающих. – СПб.: БХВ-Петербург, 2008. – 624 с.