Преподаватели:
к.ф.-м.н., заведующий кафедрой ВТиКГ ДВГУПС,
преподаватель программы IT-школа Samsung,
Пономарчук Юлия Викторовна
E-mail: yulia.ponomarchuk@gmail.com

Занятие 5.
Алгоритмы сортировок.

Алгоритмы сортировки

- ***Задача сортировки*** заключается в ***упорядочивании элементов*** в заданном списке
  - ***по невозрастанию*** – каждый следующий элемент *не больше предыдущего* или
  - ***по неубыванию*** – каждый следующий элемент *не меньше предыдущего*
- Обычно сортируются массивы чисел
- Сортируемые объекты могут быть *записями*, содержащими несколько полей. *Сравнение записей* производится по одному из полей – *ключу*

Критерии оценки времени сортировки:

1. ***Количество шагов***, необходимых для упорядочивания *N* записей
2. ***Количество сравнений*** между значениями ключей
3. Если размер записей большой, то необходимо учитывать ***время, необходимое на перестановку записей***

http://www.sorting-algorithms.com/

http://algo-rythmics.ms.sapientia.ro/

1. По устойчивости алгоритмы делятся на ***устойчивые*** и ***неустойчивые***.

   - ***Устойчивая сортировка*** не меняет взаимного расположения элементов с одинаковыми ключами,
   - ***неустойчивая*** – меняет.

2. Алгоритмы делятся на *основанные на сравнениях* и не *основанные на сравнениях*.

3. Алгоритмы делятся на алгоритмы ***внутренней сортировки*** и ***внешней сортировки***.

   - ***Внутренняя сортировка*** оперирует массивами в оперативной памяти.
   - ***Внешняя сортировка*** оперирует запоминающими устройствами большого объема с последовательным доступом (файлами).

1. Проверка уникальности
2. Удаление повторяющихся элементов
3. Распределение приоритетов событий
4. Поиск порядковой статистики (поиск k-го по величине элемента в массиве)
5. Расчет частоты встречаемости элемента в массиве
6. Восстановление первоначального порядка
7. Создание пересечения или объединения двух массивов
8. Применение бинарного поиска

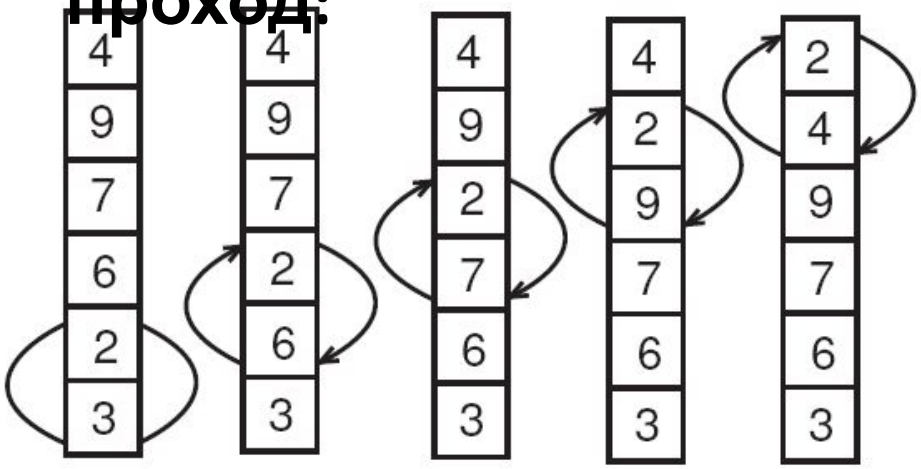
Идея:

На каждом шаге самый «легкий» элемент поднимается до своего места («всплывает»).

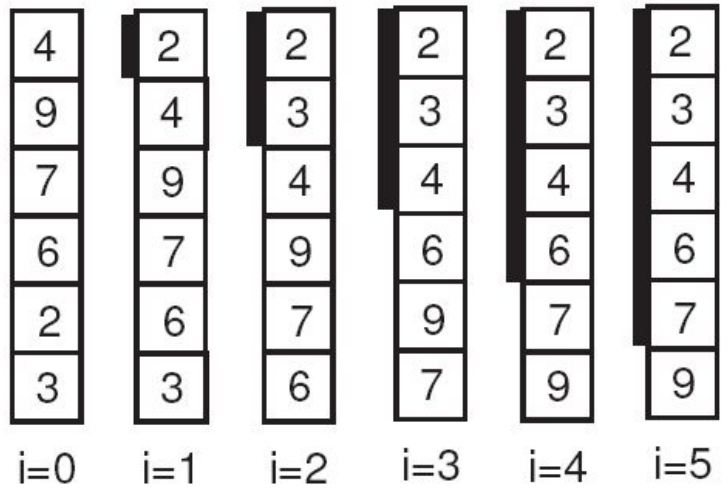
- Для этого просматриваем элементы снизу вверх,
- берем пару соседних элементов и, если они стоят неправильно, меняем их местами.

В лучшем случае, сложность по времени: *O(N)*

Первый проход:



Состояние массива после каждого прохода:



```
for(int i = 0; i < N; i++)
{
    for(int j = 0; j < N-i; j++)
    {
        if(myArray[j] > myArray[j+1])
        {
            currentElem = myArray[j];
            myArray[j] = myArray[j+1];
            myArray[j+1] = currentElem;
            reshuffle++;
        }
        comparison ++;
    }
}
```

Идея:

- Будем выбирать минимальный элемент в оставшейся части массива и приписывать его к уже отсортированной части.
- Повторив действия N раз, получим отсортированный массив.

Количество присваиваний: *O(N).*

В целом: ***медленный метод***, но используется, если необходимо минимизировать количество присваиваний

```
for(int i = 0; i < N; i++)
{
    int min = i;

    for(int j = i+1; j < N; j++)
    {
        if(myArray[j] < myArray[min])
        {
            min = j;
        }
    }

    currentElem = myArray[i];
    myArray[i] = myArray[min];
    myArray[min] = currentElem;
}
```

Быстрая сортировка Хоара

Идея:

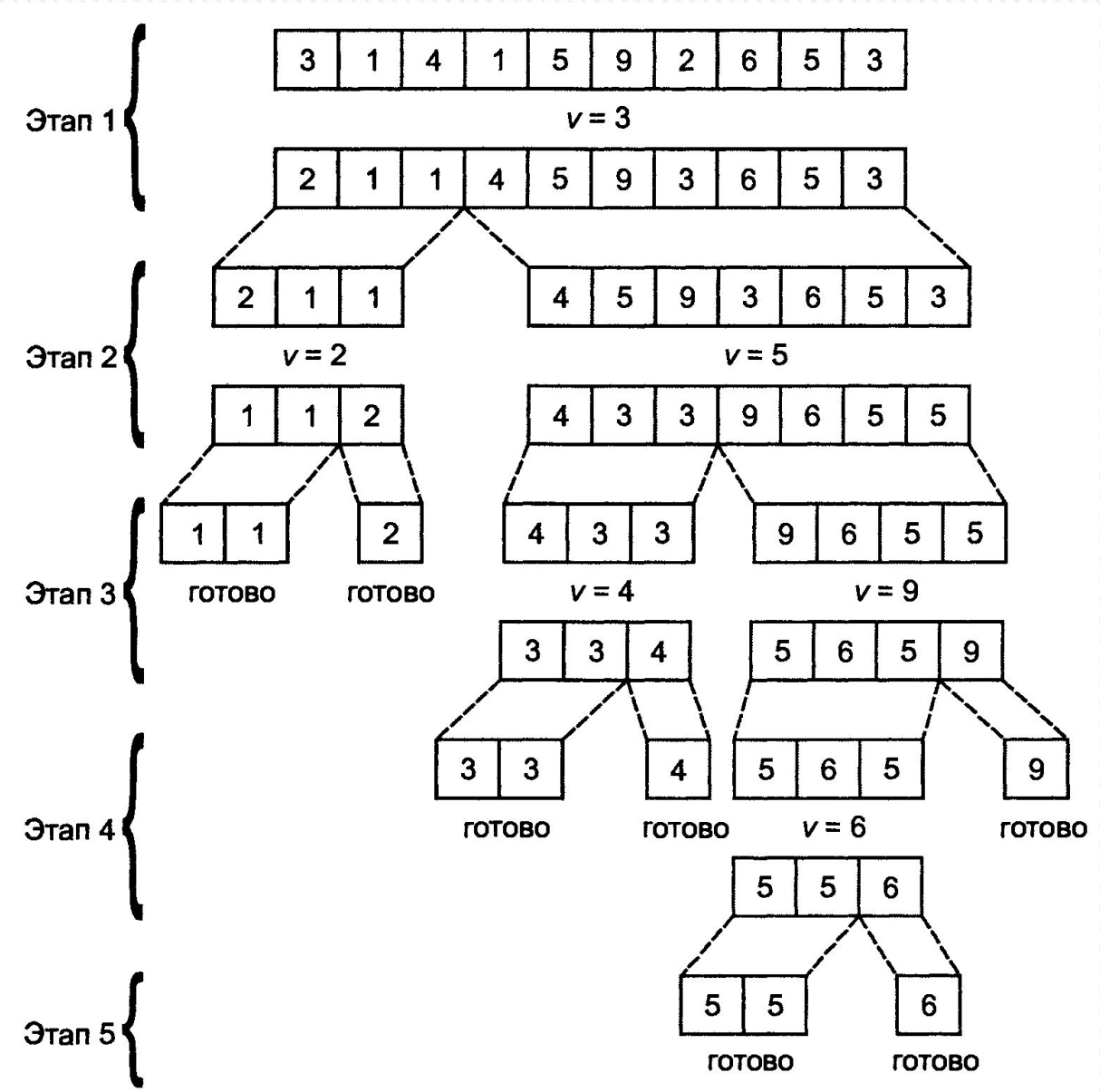
1. Для сортировки элементов массива *A[1],…,A[n]* из этих элементов выбирается некоторое значение *v* в качестве ***опорного элемента*** (желательно посередине).

2. Элементы массива переставляются так, чтобы для некоторого индекса *j* все переставляемые элементы *A[1],…,A[j]* имели значения, меньшие чем *v*, а все элементы *A[j+1],…,A[n]* – значения, большие или равные *v*.

3. Процедура быстрой сортировки рекурсивно повторяется к множествам элементов *A[1],…,A[j]* и *A[j+1],…,A[n]* для упорядочивания этих множеств по отдельности.

Количество присваиваний – *O(N*logN)*.

В худшем случае: $O(N^2)$


Этап 1
3 1 4 1 5 9 2 6 5 3
v = 3
2 1 1 4 5 9 3 6 5 3
Этап 2
2 1 1
4 5 9 3 6 5 3
v = 2
v = 5
1 1 2
4 3 3 9 6 5 5
Этап 3
1 1
2
4 3 3
9 6 5 5
готово
готово
v = 4
v = 9
3 3 4
5 6 5 9
Этап 4
3 3
4
5 6 5
9
готово
готово
v = 6
готово
5 5 6
Этап 5
5 5
6
готово
готово

```
template<class T>
void quickSortR(T* a, long N)
{
    // На входе - массив a[], a[N] - его последний элемент.
    // поставить указатели на исходные места
    long i = 0, j = N-1;
    T temp, p;

    // центральный элемент
    p = a[ N>>1 ];

    // процедура разделения
    do
    {
        while ( a[i] < p ) i++;
        while ( a[j] > p ) j--;

        if (i <= j)
        {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
            i++;
            j--;
        }
    }
    while ( i<=j );

    // рекурсивные вызовы, если есть, что сортировать
    if ( j > 0 ) quickSortR(a, j);
    if ( N > i ) quickSortR(a+i, N-i);
}
```

Сортировка в соответствии с *лексикографическим порядком*

ключевое значение (a_1, a_2, …, a_k) меньше ключевого значения (b_1, b_2, …, b_k), если существует такой номер j ($1 \le j \le k-1$),

что $a_1=b_1$, $a_2=b_2$, …, $a_j=b_j$ и $a_{j+1}<b_{j+1}$.

Идея:

1. Отсортируем числа по последнему разряду (единиц)

2. Повторим то же самое для второго и последующих разрядов, пользуясь каким-либо устойчивым алгоритмом сортировки

Количество присваиваний – *O(k*N+k*m)*, *k*- количество знаков в числе, *m* – основание системы счисления.

0	1	2	3
329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	837	657	839

1. Пирамидальная сортировка *O(N*logN)*
2. Двоичная сортировка *O(N*logN)*
3. Сортировка слияниями *O(N*logN)*
4. Сортировка подсчетом *O(N+k)*

Сравнение производительности алгоритмов сортировок

Название	Сравнений	Присваиваний	Память
Пузырьком	$O(N^2)$	$O(N^2)$	0
Выбором	$O(N^2)$	$O(N)$	0
Пирамидальная	$O(N \log N)$	$O(N \log N)$	0
Быстрая	$O(N \log N)$	$O(N \log N)$	$O(\log N)$
Слиянием	$O(N \log N)$	$O(N \log N)$	$O(N)$
Подсчетом	0	$O(N)$	$O(MAXV)$
Поразрядная	0	$O(kN + km)$	$O(N + km)$

Сортировка в C++

Библиотека STL:

- функция sort – сортировка
- функция stable_sort – устойчивая сортировка
- функция nth_element – возвращает n-ю порядковую статистику
- функция unique – удаляет все последовательные дубликаты

У красотки Полли нет недостатка в прекрасно воспитанных ухажерах. Напротив, самой большой проблемой является отслеживание самых лучших из них.

Полли очень любит танцевать, и она считает, что **оптимальный рост** ее партнера составляет **180 см**. Поэтому первое ее требование состоит в том, чтобы найти кого-то, чей рост близок, насколько это возможно, к этой величине. **Среди всех кандидатов одного роста** ей нужен тот, чей **вес** близок, насколько это возможно к **75 кг**, **но не превышает этой величины.**

Напишите программу, ранжирующую мужчин **от более желаемого к менее желаемому**. **Если у двух или более людей рост и вес совпадают, то отсортируйте их по фамилии, а после, если это необходимо, по имени**.