Алгоритм быстрой сортировки. Реализация на C(C++) и на Паскале

1 Основная идея

Делим массив на 2 части:

1. Массив, состоящий из *больших* элементов, все элементы в нём больше или равны опорному $x_i \geq s$.
2. Массив, состоящий из *маленьких* элементов, все элементы в нём меньше или равны опорному $x_i \leq s$.

Любой элемент из первой части <= элементу из второй части.

После этого каждую часть опять делим на «маленькие» и большие «элементы».

Процесс деления на две части.

В первой части ищем первый элемент, который больше опорного. Во второй части ищем элемент меньший опорного. Эти элементы находятся не в своей части. Их меняем местами. И так продолжается до тех пор пока в первой части есть *большие*, а во второй – *маленькие* элементы.

Разделили массив на части. Для каждой из них рекурсивно применяем алгоритм деления массива «большие» и «маленькие». Повторяем алгоритм до тех пор, пока не отсортируем (пока в рекурсивном вызове не останется массив из одного элемента).

Количество делений на «большие» и «маленькие» элементы – $\log_2 n$.

Количество сравнений для деления массива на «большие» и «маленькие» элементы – в среднем n.

Всего вычислений – порядка $n \cdot \log_2 n$.

Алгоритм быстрой сортировки носит имя Хоара, по имени автора английского информатика Чарльза Хоара, который разработал его во время его стажировки в МГУ в 1960 году.

2 Формальное описание алгоритма

1. Задан массив X[N]. Вводим два счётчика (f, L), которые будут двигаться навстречу друг другу. Начальные значения счётчиков f=1, L=N. Выберем опорный элемент $s = X[\frac{f+L}{2}]$.

2. Будем увеличивать f на 1, до тех пор, пока $x[f]<s$.
3. Будем уменьшать L на 1, до тех пор, пока $x[L]>s$.

2-3. Алгоритм встречного поведения.

4. Если i<j, то меняем местами x[f] и x[L]. После этого f++, L--.
5. Повторяем п.2-4, до тех пор пока f не станет больше L.

Элементы правее l больше или равны опорного. Все элементы левее L меньше или равны опорному. Мы как бы раздели массив на две части (левая – маленькие и правая – большие).

6. Теперь рекурсивно запускаем алгоритм 1-5 на массивах x[f:N], и x[1:f-1]

https://www.youtube.com/watch?v=Xgaj0Vxz_to

https://www.youtube.com/watch?v=34q6gcWaE_8

Оценка времени быстродействия на C(C++)

```
#include <time.h>
time1=clock();
```

ЗДЕСЬ Фрагмент кода

```
time2=clock();
cout<<"\nВремя умножения матриц="<<
(time2-time1)/CLOCKS_PER_SEC<<endl;
```

Функция clock

Прототип функции clock:

```
clock_t clock( void );
```

Заголовочный файл

Название	Язык
time.h	C
ctime	C++

Описание:

Функция возвращает количество временных тактов, прошедших с начала запуска программы. С помощью макроса CLOCKS_PER_SEC функция получает количество пройденных тактов за 1 секунду. Таким образом, зная сколько выполняется тактов в секунду, зная время запуска программы можно посчитать

время работы всей программы или отдельного её фрагмента, что и делает данная функция.

Возвращаемое значение

Число тактов прошедшее с момента запуска программы. В случае ошибки, функция возвращает значение -1. Возвращаемое значение функции clock имеет тип данных clock_t, который определен в <ctime>. Тип данных clock_t способен представлять временные такты, а также поддерживает арифметические операции.

БЫСТРАЯ СОРТИРОВКА!!!!

Исходный массив: 5570 5660 5470

Результирующий массив: 3 4997 9999

Время быстрой сортировки=0.001101

БЫСТРАЯ СОРТИРОВКА!!!!

Исходный массив: 8464 7179 1104

Результирующий массив: 0 4874 9999

Время сортировки методом пузырька=0.267147

Паскаль

Время быстрой сортировки:0.01000000

Время сортировки методом пузырька:2.43000000