Сортировки

Юрий Литвинов
y.litvinov@spbu.ru

21.09.2022

Комментарии по домашке

Коды ошибок

- Не используйте функцию system()
- Закомментированный код не нужен
- **if** (test == false) { всё плохо }
- Кстати, **if** (x == true) — это то же, что x + 0
- 0 и '\0'
- Если функция может не выполниться, используйте коды возврата

Пример

```
int fibonacci(int n, int *result)
{
    if (n <= 0)
        return 1;
    if (n <= 2) {
        *result = 1;
        return 0;
    }
    int previous = 0;
    fibonacci(n - 1, &previous);
    int prePrevious = 0;
    fibonacci(n - 2, &prePrevious);
    *result = previous + prePrevious;
    return 0;
}
...
int result = 0;
const int errorCode = fibonacci(x, &result);
if (errorCode != 0) {
    printf("Всё очень плохо")
} else {
    printf("%d-ое число Фибоначчи равно %d", x, result);
}
```

Или так

```
int fibonacci(int n, int *errorCode)
{
    if (n <= 0) {
        *errorCode = 1;
        return 0;
    }
    *errorCode = 0;
    if (n <= 2) {
        return 1;
    }
    return fibonacci(n - 1, errorCode) + fibonacci(n - 2, errorCode);
}
...
int errorCode = 0;
const int result = fibonacci(x, &errorCode);
if (errorCode != 0) {
    printf("Всё очень плохо")
} else {
    printf("%d-ое число Фибоначчи равно %d", x, result);
}
```

Ещё комментарии

Не пишите так:

```
if (x == 10)
{
    return true;
}
else
{
    return false;
}
```

Пишите так:

```
return x == 10;
```

И ещё комментарии

- Предупреждения компилятора
- Выделение памяти

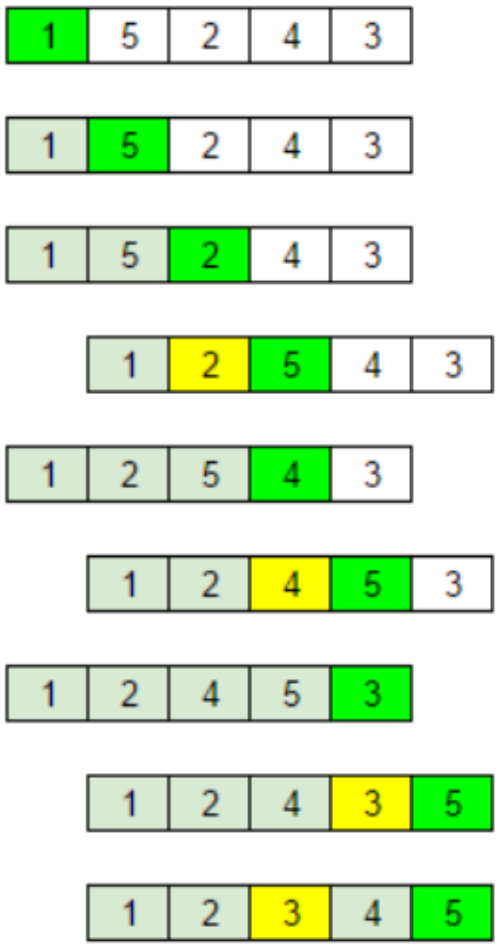
```
int *array = (int*)calloc(size, sizeof(int));
if (array == NULL)
{
    printf("Всё очень плохо :(")
    return -1;
}
...
free(array);
```

- Или `int *array = (int*)malloc(size * sizeof(int));`

Свойства сортировок

- Работают над любыми контейнерами данных
- Есть понятие “ключ”
- Устойчивость — сохраняется ли взаимное расположение элементов с одинаковым ключом
- Естественность — учёт степени отсортированности исходных данных
- Внутренняя сортировка — работает над данными, целиком помещающимися в память
- Внешняя сортировка работает над данными на устройствах с последовательным доступом, которые медленнее, чем память

Сортировка вставкой (insertion sort)



- $O(n^2)$
- Устойчива
- Естественная ($O(n)$ на отсортированном массиве)
- Данные могут приходить постепенно
- Позволяет выбрать наибольшие (или наименьшие) k чисел из входного потока

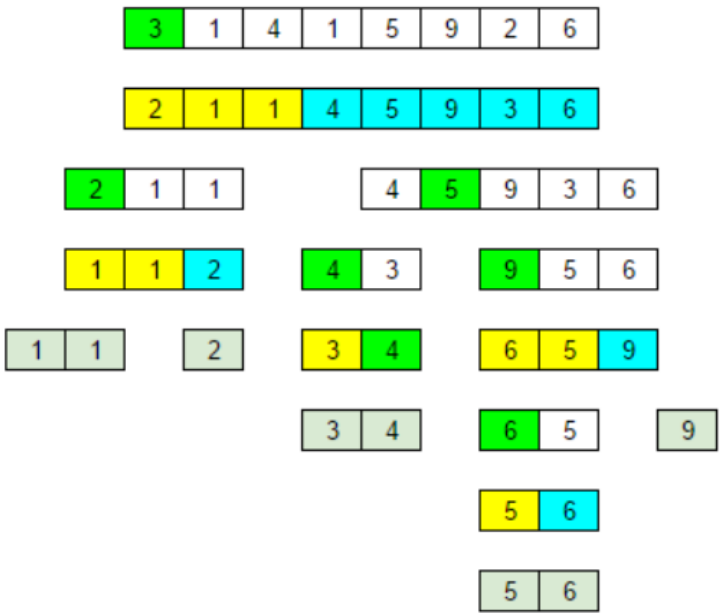
Сортировка Шелла (Shell sort)

- Сортировка вставкой подпоследовательностей в массиве с постепенно убывающим шагом
- Элементы “быстрее” встают на свои места
  - Сортировка вставкой на каждом шаге уменьшает количество инверсий максимум на 1
- $O(n * log(n)^2)$ при правильном выборе h
- Неустойчива
- Легко пишется и довольно быстра
  - Не вырождается до квадратичной

Сортировка выбором (Selection sort)

- $O(n^2)$
- Обычно неустойчива ($[2_a, 2_b, 1_a]$ -> $[1_a, 2_b, 2_a]$)
- Отсортированность массива ничего не даёт
- Меньше всего операций обмена (меньше операций записи, что иногда позитивно)

Быстрая сортировка (qsort)



- $O(n * log(n))$, вырождается до $O(n^2)$
- Неустойчива
- Требует $O(n * log(n))$ дополнительной памяти
- Самый быстрый на практике алгоритм сортировки, используется в стандартных библиотеках
- Легко пишется (но тяжело отлаживается)

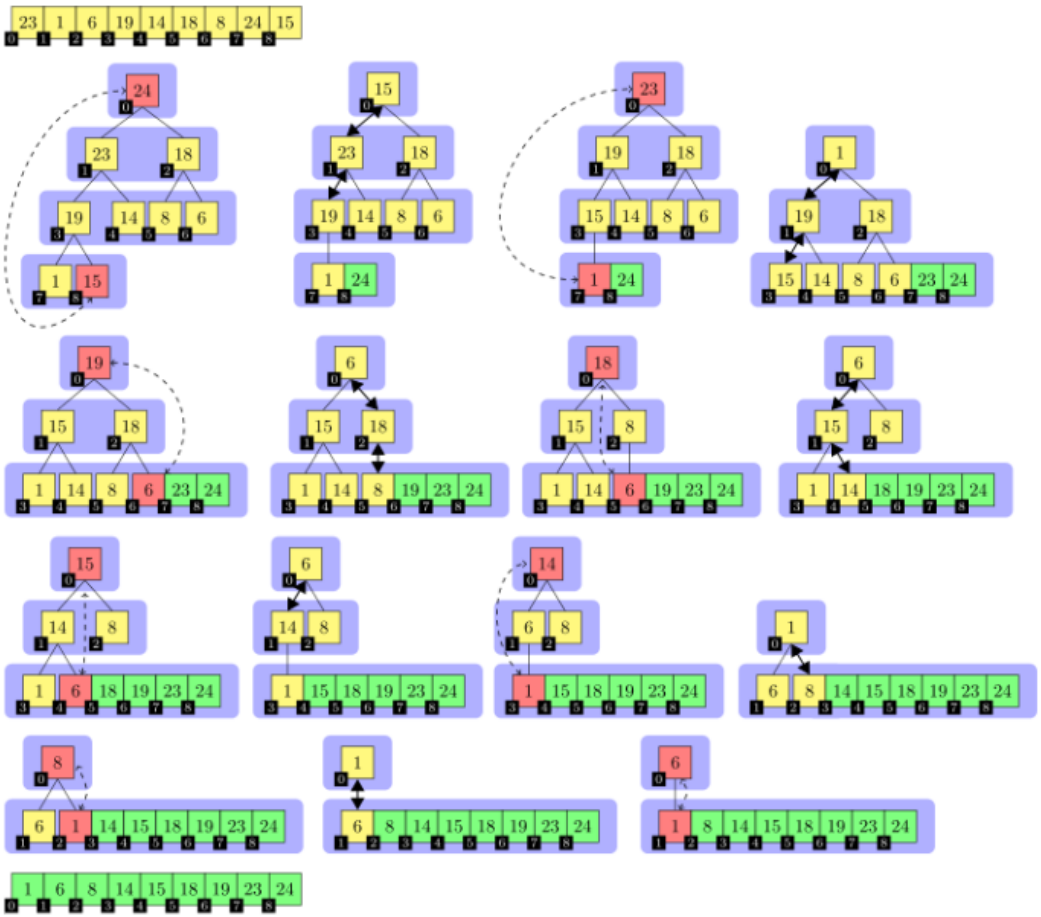
Псевдокод

```
algorithm quicksort(A, lo, hi) is
    if lo < hi then
        p := partition(A, lo, hi)
        quicksort(A, lo, p – 1)
        quicksort(A, p + 1, hi)
```

```
algorithm partition(A, lo, hi) is
    pivot := A[hi]
    i := lo
    for j := lo to hi – 1 do
        if A[j] ≤ pivot then
            swap A[i] with A[j]
            i := i + 1
    swap A[i] with A[hi]
    return i
```

Нерекурсивная реализация — через стек, в котором хранятся границы сортируемых кусков массива

Сортировка кучей (пирамидальная, heapsort)



- $O(n * log(n))$, не вырождается
- Не требует дополнительной памяти
- Неустойчива
- Требует произвольного доступа к памяти
- Сложна в реализации

Двоичный поиск

- Находит элемент в массиве за $O(log(n))$
- Легко напутать с индексами и уйти в бесконечный цикл