Принцип „Разделяй и властвуй“ на примере сортировки слиянием и бинарного поиска

Для начала стоит сказать о том, что такое метод «разделяй и властвуй»? Главная его идея – разбиение основной задачи на несколько более маленьких, пока эти подзадачи не станут элементарными. Такой метод сокращает время работы программы, что легко доказывается путем применения основной теоремы о рекуррентных соотношениях (*Master theorem*), что будет показано на следующей лекции.

Для начала рассмотрим бинарный (двоичный) поиск. В чем заключается задача? Есть некоторый элемент „**x**“, есть некоторая последовательность $a_1, a_2, \ldots, a_n$, в которой каждый следующий элемент (например) больше предыдущего или равен ему. Требуется определить, найдется ли в данной последовательности такой элемент, который по величине равен элементу „**x**“. Перед нами задача поиска. Можно осуществить полный перебор, но это не нужно, так как есть более эффективный подход.

Возьмем произвольный элемент a_k такой, что $1 \leq k \leq n$, и сравним его с „**x**“. Есть всего три варианта:

1) $x < a_k$. Значит, ищем в левой части: $\{ a_1, \ldots, a_{k-1} \}$

2) $x = a_k$. Элемент найден

3) $x > a_k$. Значит, ищем в правой части: $\{ a_{k+1}, \ldots, a_n \}$

Получается, что мы в определенное количество раз сокращаем длину массива за каждое сравнение. Если никакой дополнительной информации о структуре массива нет, то имеет смысл уменьшать длину массива вдвое за каждое сравнение, то есть $k = [\frac{n}{2}]$.

Какова сложность бинарного поиска? $O(n)$, затем $O(\frac{n}{2})$, $O(\frac{n}{4})$, $\ldots$, $O(\frac{n}{2^m})$. Получается следующее: $\frac{n}{2^m} = 1$; $n = 2^m$; $m = \log_2 n$. Это и есть сложность бинарного поиска $O(\log_2 n)$

Теперь рассмотрим задачи сортировки, а именно о сортировку слиянием (*Merge sort*). Главная идея алгоритма это — деление исходного массива на несколько массивов меньшего размера (Чтобы потом их быстро отсортировать). Рассмотрим принцип работы алгоритма на двух массивах. Пусть у нас есть массив „**a**“ длины „**n**“ и массив „**b**“ длины „**m**“, из них мы хотим получить массив с длины „**n**+**m**“. При этом два этих массива уже отсортированы. Реализуем алгоритм слияния.

```
i = 0;
j = 0;
for (int k = 0; k < n + m; k++)
{
    c[k] = min(a[i],b[j]);
    if (a[i] <= b[j])
    {
        i++;
    }
    else
    {
        j++;
    }
}
```

На каждом шаге мы берем два элемента из разных массивов, меньший из них записываем в новый массив, изменяем счетчик у этого массива, а после повторяем шаг. Надо только следить за счетчиками „**i**“ и „**j**“. Когда элементы в каком-то массиве закончатся, нужно прекратить выполнение программы – ведь если элементы закончились в массиве „**a**“, к примеру, то оставшиеся элементы в массиве „**c**“ будут состоять из оставшихся элементов массива „**b**“.

Данный метод можно применить для сортировки массива. Возьмем исходный, разобьем его на две части, чтобы потом „слить“ их воедино. Но эти два подмассива должны быть отсортированы. Тогда каждый из этих подмассивов нужно разбить еще на две части. А потом еще и еще. До тех пор, пока задача сортировки не станет элементарной, то есть, когда массивы не будут состоять из одного или двух элементов. Именно это и называется сортировкой слиянием. Какова же сложность такого алгоритма? На первом шаге у нас два массива длины $\frac{n}{2}$, то есть сложность $O(n)$; на втором шаге у нас уже четыре массива, длина каждого из которых $\frac{n}{4}$, то есть сложность $4 \cdot O(\frac{n}{4}) = O(n)$ и так далее. Получается, что у нас $\log_2 n$ шагов, сложность каждого из которых $O(n)$, то есть общая сложность — $O(n \cdot \log_2 n)$.

1)	Массивы:	$\{a_1, \ldots, a_{\frac{n}{2}}\}, \{a_{\frac{n}{2}}, \ldots, a_n\}$	$(2 \cdot O(\frac{n}{2}))$
...			
$log_2 n$)	Массивы:	$\{a_1\}, \ldots, \{a_n\}$	$(n \cdot O(1))$

Но при этом алгоритм сортировки слиянием неэффективен при массивах небольшой длины, так что обычно используют *timsort* — это сортировка представляет собой гибрид из сортировки слиянием и сортировки

вставками. Основной способ – сортировка слиянием, но с некоторого момента будет использоваться сортировка вставками, что эффективно для массивов небольшой длины.