Sortowanie szybkie quicksort

Sortowanie szybkie (ang. quicksort) – jeden z popularnych algorytmów sortowania działających na zasadzie „dziel i zwyciężaj”

Sortowanie szybkie (ang. QuickSort) zostało wynalezione w 1962 przez C.A.R. Hoare’a.

Algorytm sortowania szybkiego jest wydajny: jego średnia złożoność obliczeniowa jest rzędu O(n log n). Ze względu na szybkość i prostotę implementacji jest powszechnie używany. Jego implementacje znajdują się w bibliotekach standardowych wielu środowisk programowania.

Wyjaśnimy jak quicksort wykorzystuje strategię dziel-i-rządź (divide-and-conquer). Podobnie jak w przypadku merge sort, załóżmy, że chcemy posortować podtablicę `array[p..r]`, zakładając na początku, że nasza tablica to `array[0..n-1]`.

1. **Podzielmy** poprzez wybranie elementu w podtablicy `array[p..r]`. Nazwijmy ten element **elementem rozdzielającym**. Przestawmy elementy w podtablicy `array[p..r]`, tak aby wszystkie elementy w podtablicy `array[p..r]`, które są mniejsze lub równe elementowi rozdzielającemu były na lewo od niego, a wszystkie pozostałe elementy w podtablicy były od niego na prawo. Tę procedurę nazywamy **partycjonowaniem**. W tym momencie, nie jest istotne, w jakim porządku względem siebie znajdują się zarówno elementy na lewo od elementu rozdzielającego, jak i na prawo. Dbamy jedynie o to, aby każdy z elementów znajdował się w pewnym miejscu po odpowiedniej stronie elementu rozdzielającego.

   Z powodów praktycznych zawsze wybieramy najbardziej prawy element w podtablicy `array[r]` jako element rozdzielający. Więc, na przykład, jeśli podtablica składa się z elementów [9, 7, 5, 11, 12, 2, 14, 3, 10, 6], wybieramy 6 jako element rozdzielający. Po partycjonowaniu, podtablica może wyglądać następująco: [5, 2, 3, 6, 12, 7, 14, 9, 10, 11]. Przyjmijmy, że 'q' jest indeksem elementu rozdzielającego.

2. **Zwyciężaj** rekurencyjnie sortując podtablice `array[p..q-1]` (wszystkie elementy na lewo od piwotu, o wartości mniejszej bądź równej wartości piwotu) i `array[q+1..r]` (wszystkie elementy na prawo od piwotu, o wartości większej niż piwot).

3. **Połącz** nie robiąc nic. Skończyliśmy kiedy krok dzielenia posortował rekurencyjnie. Dlaczego? Wszystkie elementy na lewo od piwotu w tablicy `array[p..q-1]` są mniejsze

lub równe piwotowi i są posortowane oraz wszystkie elementy na prawo od piwotu w `array[q+1..r]` są większe niż piwot i są posortowane. Elementy w `array[p..r]` nic na to nie poradzą ale są posortowane!

Zastanów się nad naszym przykładem. Po sortowaniu rekurencyjnym pod-tablica na lewo od piwotu wynosi [2, 3, 5], a tablica na prawo od piwotu wynosi [7, 9, 10, 11, 12, 14]. Więc pod-tablica posiada [2, 3, 5] następnie 6 oraz [7, 9, 10, 11, 12, 14]. Pod-tablica jest posortowana!

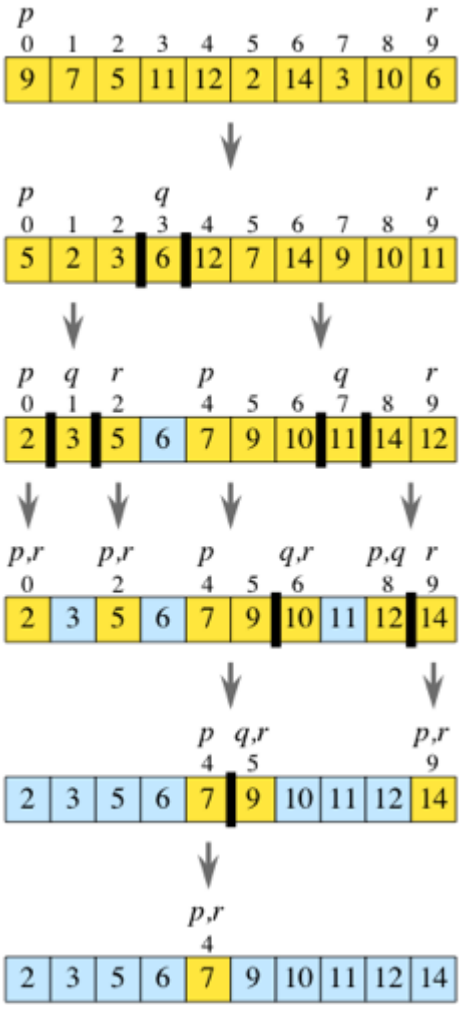
Podstawowymi przypadkami są tablice o mniej niż dwu elementach, tak jak w sortowaniu przez złączanie (merge sort). W sortowaniu przez złączanie, nie spotyka się pod-tablicy bez elementów, ale w quicksort może się tak zdarzyć, jeżeli pozostałe elementy pod-tablicy są wszystkie mniejsze lub wszystkie większe niż piwot.

Cofnijmy się do drugiego kroku i prześledźmy rekurencyjne sortowanie pod-tablic. Po pierwszym podziale, mamy pod-tablice [5, 2, 3] i [12, 7, 14, 9, 10, 11], z 6 jako piwotem.

Aby posortować pod-tablicę [5, 2, 3], wybieramy 3 jako piwot. Po podziale mamy [2, 3, 5]. Pod-tablica [2], po lewej stronie piwotu, jest podstawowym przypadkiem, podobnie jak pod-tablica [5] po prawej.

Aby posortować pod-tablicę [12, 7, 14, 9, 10, 11], wybieramy 11 jako piwot, co daje w rezultacie [7, 9, 10] po lewej stronie piwotu i [14, 12] po prawo. Po posortowaniu tych pod-tablic, otrzymujemy kolejno [7, 9, 10], 11, [12, 14].

Poniżej zaprezentowano cały przebieg algorytmu quicksort. Na niebiesko zaznaczono miejsca, gdzie były piwoty w poprzednich przebiegach rekurencji, zatem wartości w tych miejscach już nie będą sprawdzane ani przenoszone:



Funkcja sortowania w języku Python

```
def qsort(arr, l=0, r=None):
    if r is None: r = len(arr) - 1
    i, j = l, r
    pivot = arr[(l + r) / 2]
    while i <= j:
        while arr[i] < pivot: i += 1
        while arr[j] > pivot: j -= 1
        if i <= j:
            arr[i], arr[j] = arr[j], arr[i]
            i += 1; j -= 1
    if l < j: qsort(arr, l, j)
    if r > i: qsort(arr, i, r)
```