

VERİ YAPILARILARI VE ALGORİTMALAR

Quicksort

Quicksort

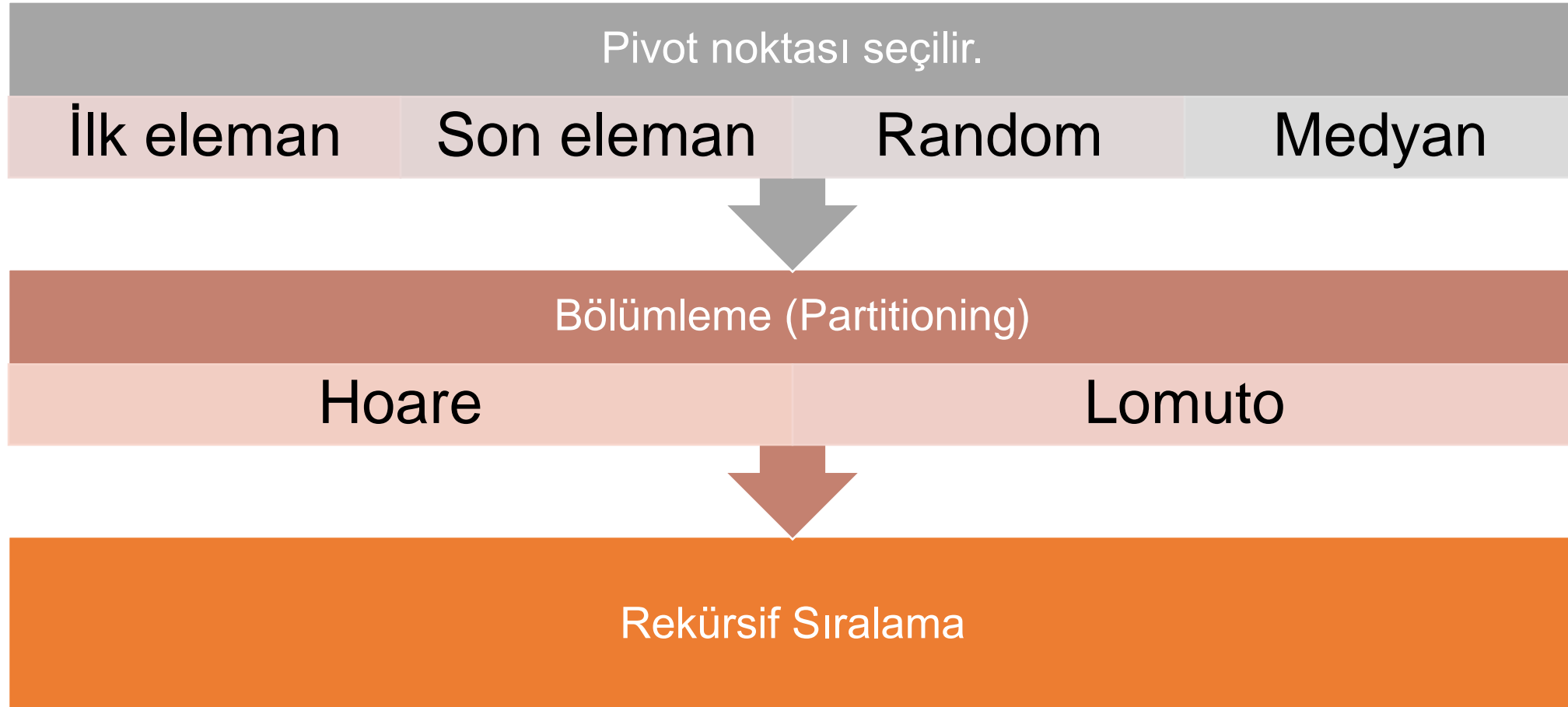
- Quicksort algoritması verimli bir sıralama algoritmasıdır.
- İyi bir şekilde uygulandığında **Merge sort** ya da **Heap sort** gibi algoritmalara kıyasla iki ya da üç kat daha hızlı olabilir.
- Bir böl ve yönet (**divide and conquer**) algoritmasıdır.

Quicksort

```
/* low --> Starting index, high --> Ending index */
quicksort(arr[], low, high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[pi] is now
           at right place */
        pi = partition(arr, low, high);

        quicksort(arr, low, pi - 1); // Before pi
        quicksort(arr, pi + 1, high); // After pi
    }
}
```

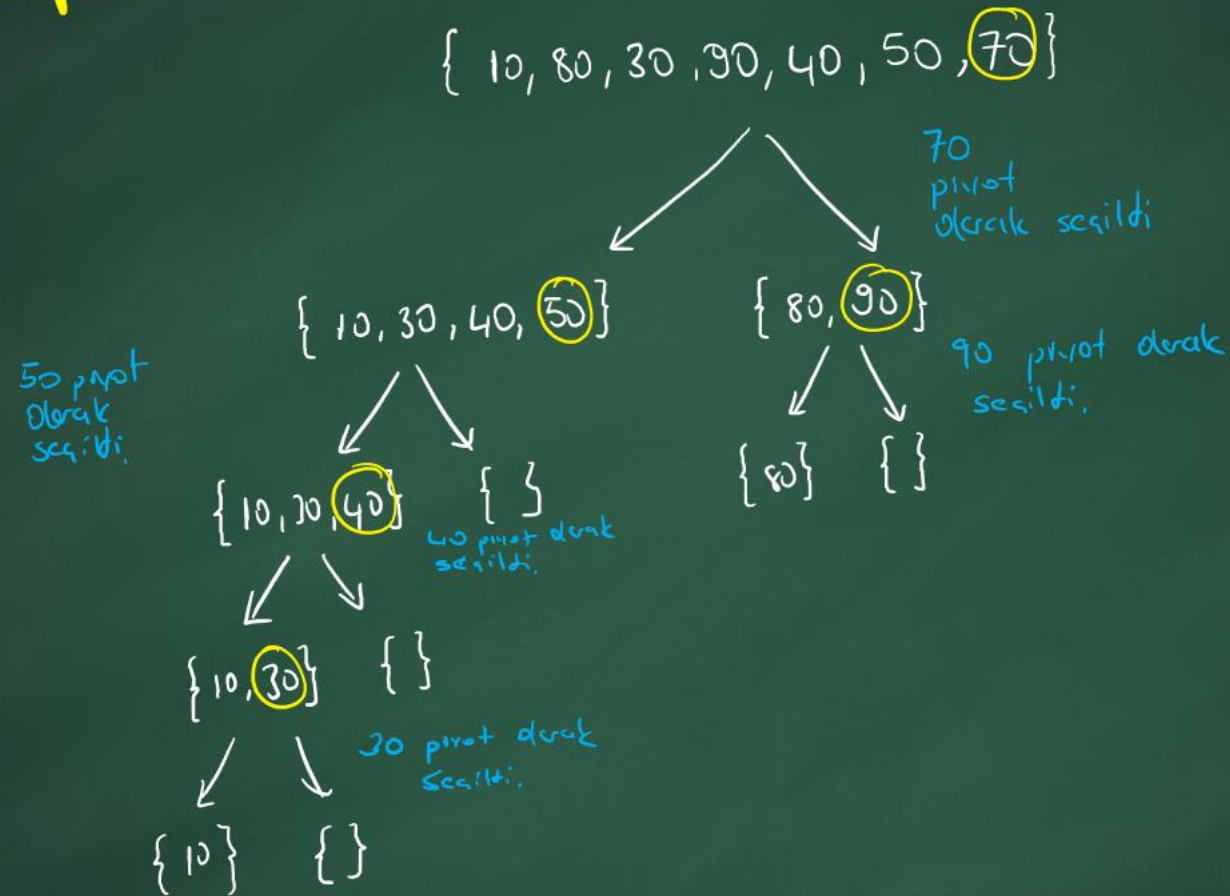
Quicksort



Quicksort

Sınıf	Sıralama algoritması
En-kötü performans	$O(n^2)$
En-iyi performans	$O(n \log n)$ (simple partition) or $O(n)$ (three-way partition and equal keys)
Ortalama-performans	$O(n \log n)$
En-kötü alan karmaşıklığı	$O(n)$ auxiliary (naive) $O(\log n)$ auxiliary (Hoare 1962)

Quicksort



Quicksort

```
quicksort(arr, low, high)
```

```
{
```

```
  if (low < high)
```

```
  {
```

```
    pi = partition(arr, low, high);
```

```
    quicksort(arr, low, pi-1)
```

```
    quicksort(arr, pi+1, high)
```

```
  }
```

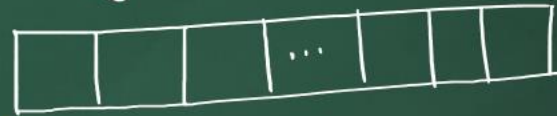
```
}
```

- Farklı bölümlendirme (*partition*) algoritmaları kullanılabilir.

- Pivot elemanının son eleman seçildiği yaklaşımda elemanlar üzerinde gezilirken;

* Pivot değerinde daha küçük bir eleman ile karşılaşırsa *Swap* yapılır.

Array



$i = (low - 1)$

$j = \text{from } low \text{ to } high - 1$

Quicksort

Partition

partition (arr[], low, high)

```
{
    i = (low - 1);
    for (j = low; j ≤ high - 1; j++)
    {
        if (arr[j] < pivot)
        {
            i++;
            swap arr[i] and arr[j];
        }
    }
    swap arr[i+1] and arr[j];
    return i;
}
```

arr = { 10, 80, 30, 90, 40, 50, ^{pivot}70 }

Index 0 1 2 3 4 5 6

low = 0 high = 6 Pivot = arr[high] = 70

Quicksort

Partition

```

partition(arr[], low, high)
{
    i = (low - 1); pivot = arr[high];
    for (j = low; j ≤ high - 1; j++)
    {
        if (arr[j] < pivot)
        {
            i++;
            swap arr[i] and arr[j];
        }
    }
    swap arr[i+1] and arr[j];
    return i+1;
}
    
```

$arr = \{ 10, 80, 30, 90, 40, 50, \overset{\text{pivot}}{70} \}$
 Index 0 1 2 3 4 5 6

low = 0 high = 6 Pivot = arr[high] = 70

partition(arr, 0, 6)

(10 < 70) ✓ → i = -1 j = 0 → i++ / arr[0] - arr[0] / j++

(80 < 70) ✗ → i = 0 j = 1 → j++

(30 < 70) ✓ → i = 0 j = 2 → i++ / arr[1] - arr[2] / j++
 { 10, 30, 80, 90, 40, 50, 70 }

(90 < 70) ✗ → i = 1 j = 3 → j++

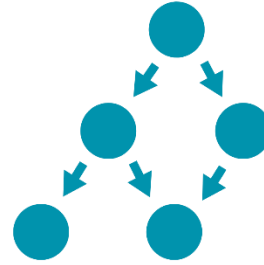
(40 < 70) ✓ → i = 1 j = 4 → i++ / arr[2] - arr[4] / j++
 { 10, 30, 40, 90, 80, 50, 70 }

(50 < 70) ✓ → i = 2 j = 5 → i++ / arr[3] - arr[5] / j++
 { 10, 30, 40, 50, 80, 90, 70 }

i = 3 j = 6

{ 10, 30, 40, 50, 80, 90, 70 }

{ 10, 30, 40, 50, 70, 90, 80 }



Veri Yapıları ve Algoritmalar

ZAFER CÖMERT

Öğretim Üyesi