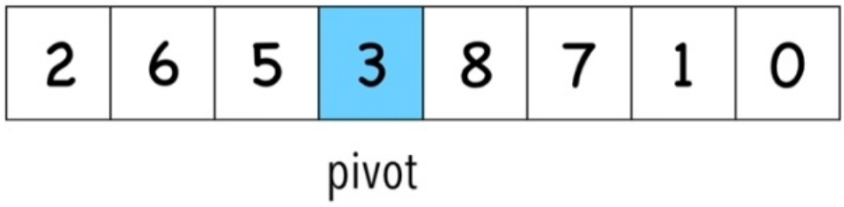
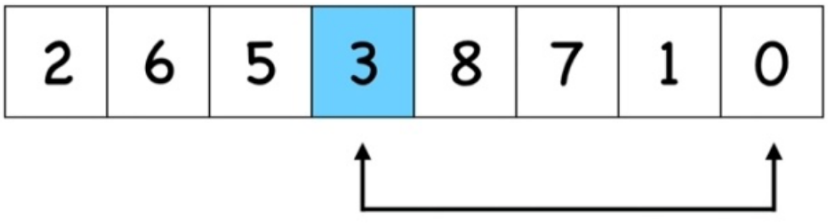
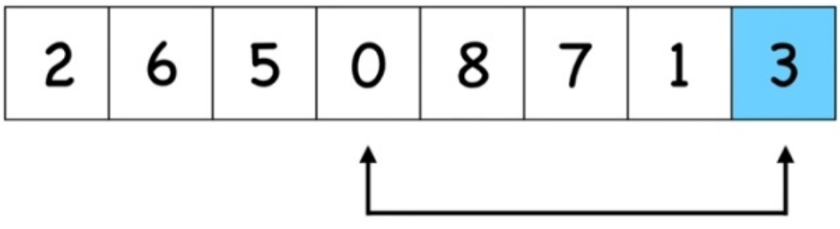
QUICKSORT

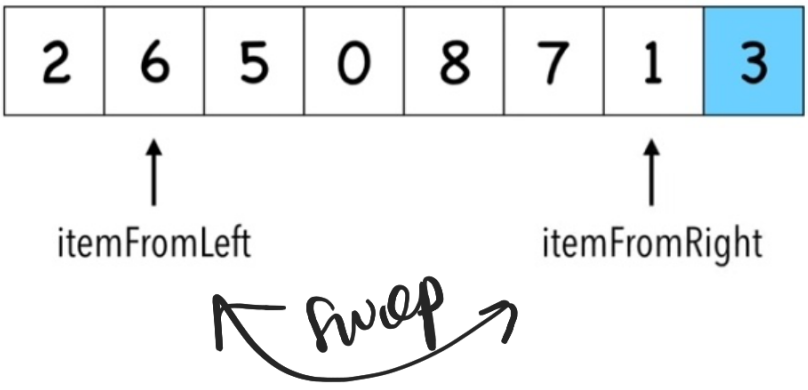


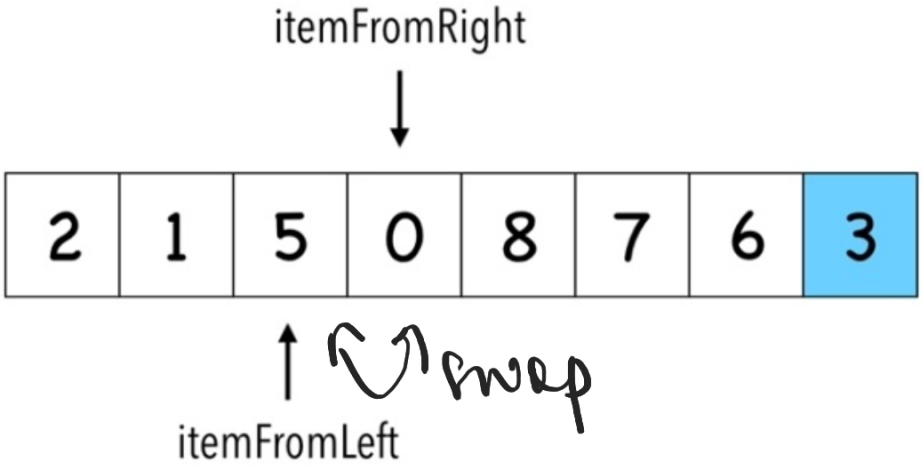


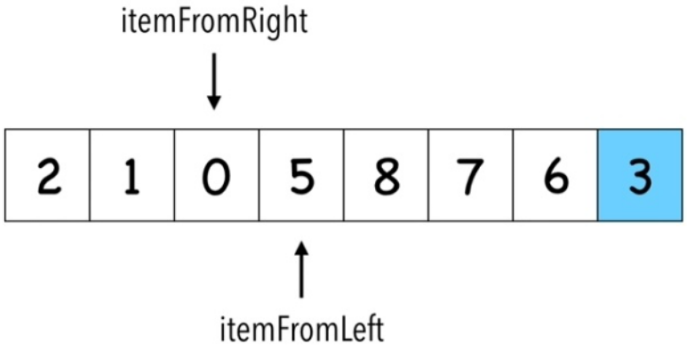


1. itemFromLeft that is larger than pivot

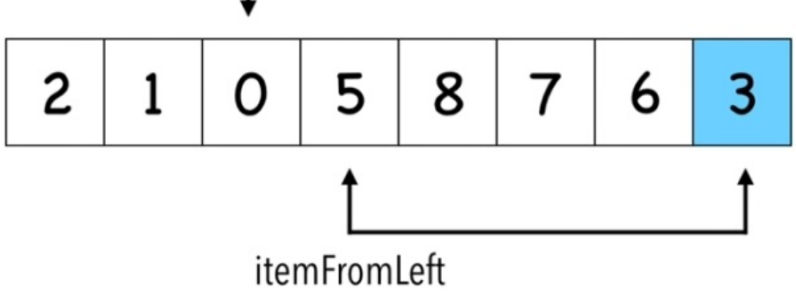
2. itemFromRight that is smaller than pivot





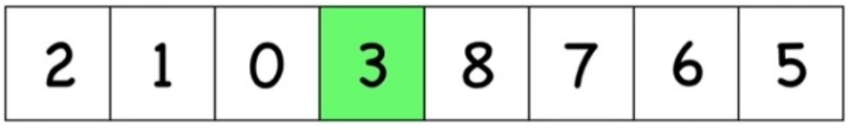


Stop when index of itemFromLeft > index of itemFromRight



Swap itemFromLeft and pivot

1. Correct position in final, sorted array
2. Items to the left are smaller
3. Items to the right are larger



and recursive

```
int partition (int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);

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

```
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

1 Quicksort

O algorytmie *Quicksort* wspomnieliśmy omawiając strategię dziel i zwyciężaj. Podany tam schemat

algorytmu można zapisać w następujący sposób:

```
procedure quicksort(A[1..n], p, r)
    if r − p jest małe then insert − sort(A[p..r])
    else choosepivot(A, p, r)
         q ← partition(A, p, r)
         quicksort(A, p, q)
         quicksort(A, q + 1, r)
```

pivot przemieszony na ostatnie miejsce w tablicy

Kluczowe znaczenie dla efektywności algorytmu mają wybór *pivota*, tj. elementu dzielącego, dokonywany w procedurze *choosepivot*, oraz implementacja procedury *partition* dokonującej przestawienia elementów tablicy A.

1.1 Implementacja procedury partition

Zakładamy, że w momencie wywołania $partition(A, p, r)$ pivot znajduje się w $A[p]$. Procedura przestawia elementy podtablicy $A[p..r]$ dokonując jej podziału na dwie części: w pierwszej – $A[p..q]$ – znajdą się elementy nie większe od pivota, w drugiej – $A[q+1, r]$ – elementy nie mniejsze od pivota. Granica tego podziału, wartość q, jest przekazywana jako wynik procedury.

```
procedure partition(A[1..n], p, r)
    x ← A[p]  pivot
    i ← p − 1
    j ← r + 1
    while i < j do
        repeat j ← j − 1 until A[j] ≤ x
        repeat i ← i + 1 until A[i] ≥ x
        if i < j then zamień A[i] i A[j] miejscami
                 else return j
```

Fakt 1 *Koszt procedury* $partition(A[1..n], p, r)$ *wynosi* $\Theta(r - p)$.