Sorting
triediace algoritmy

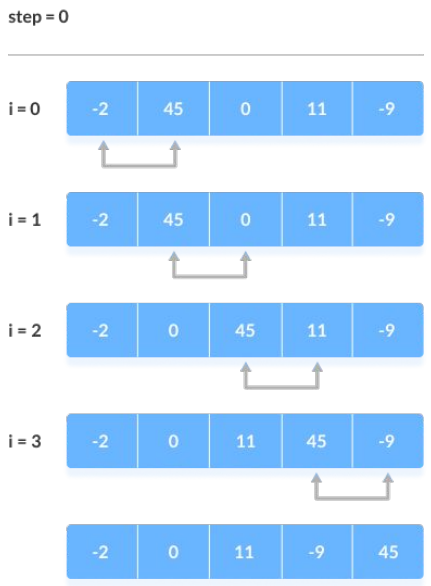
Dátové štruktúry a algoritmy 25/26 LS
Prednášky, garant: prof. Gabriel Juhás
Cvičenia: Milan Mladoniczky - milan.mladoniczky@paneurouni.com

Bubble Sort

- Triediaci algoritmus, ktorý porovnáva dva susediace prvky a prehodí ich kým nesiu v želanom poradí.
- Porovnávacia funkcia môže byť ľubovoľná.
- Pre triedenie nie je vytvorená ďalšia štruktúra a prvky sú triedené priamo v kontajnery (tzv. in-place).
- Algoritmus prechádza kontajner niekoľkokrát pokiaľ prvky nie sú zotriedené.
- Nie je vhodné použiť pre väčšie data sety. Časová zložitosť je $O(n^2)$ a priestorová náročnosť $O(1)$.

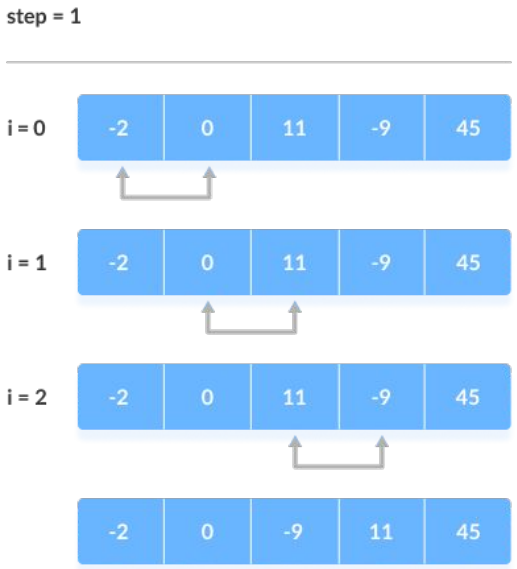
Bubble Sort

- Počnúc prvým indexom porovnáme prvý a druhý prvok.
- Ak je prvý prvok väčší ako druhý prvok, vymenia sa.
- Teraz porovnáme druhý a tretí prvok. Ak nie sú v rovnakom poradí, prehoďte ich.
- Uvedený postup pokračuje až po posledný prvok.



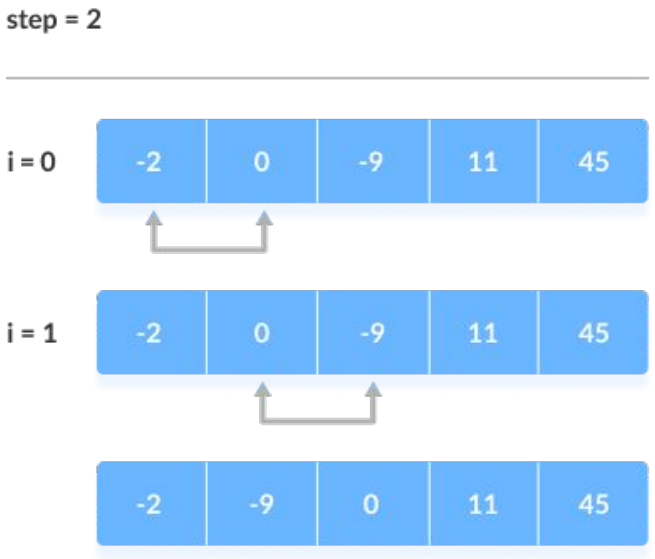
Bubble Sort

- Rovnaký postup pokračuje aj pri zvyšných iteráciách.
- Po každej iterácii sa na koniec umiestni najväčší prvok spomedzi netriedených prvkov.



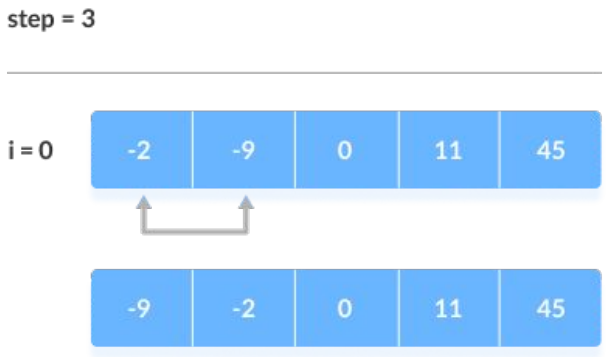
Bubble Sort

V každej iterácii sa porovnanie uskutoční až po posledný nezoradený prvok.



Bubble Sort

Pole je zoradené, keď sú všetky nezoradené prvky umiestnené na správnych pozíciách.



Bubble Sort

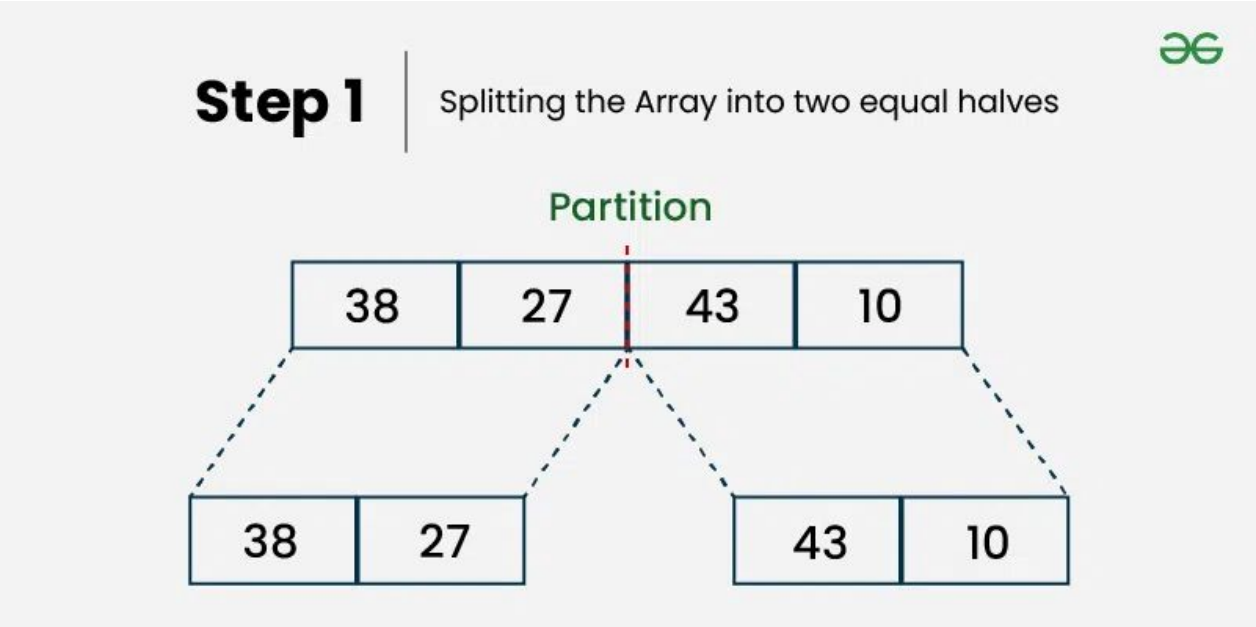
```
bubbleSort(array)
  for i <- 1 to sizeOfArray - 1
    for j <- 1 to sizeOfArray - 1 - i
      if leftElement > rightElement
        swap leftElement and rightElement
end bubbleSort
```

Merge Sort

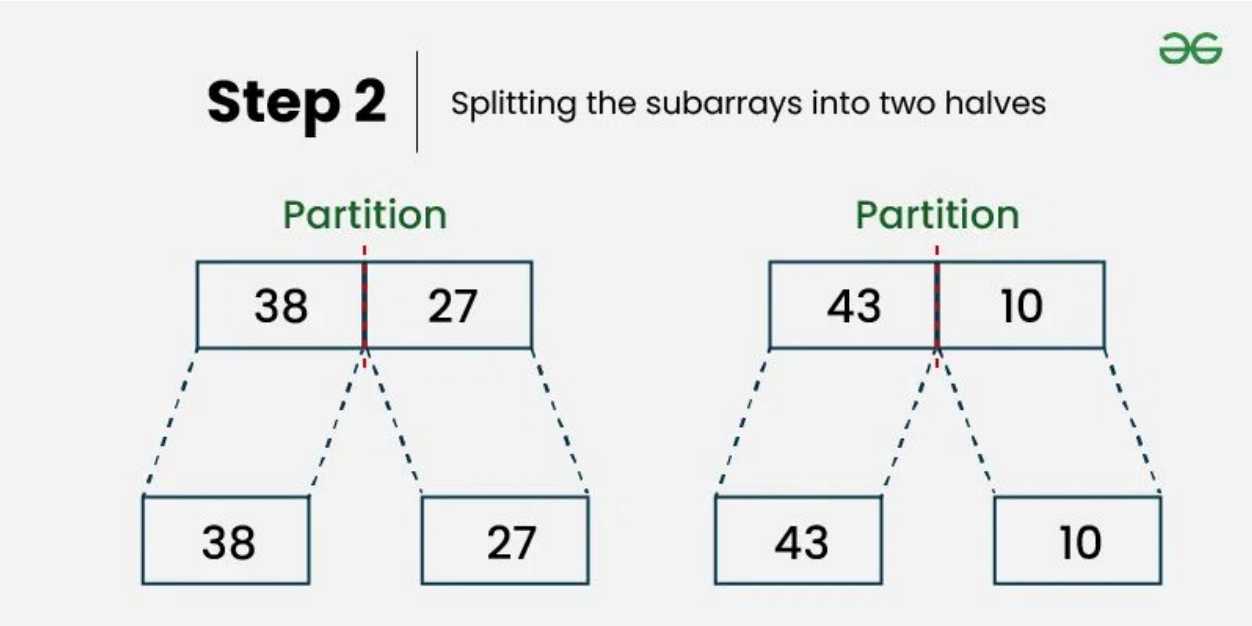
- Algoritmus sa riadi princípom rozdeľ a panuj.
- Rekurzívne delí kontajner na menšie a menšie podmnožiny kým sa dajú, zotriedi ich a následne ich naspäť spojí do kompletného kontajneru.
- Stabilný a efektívny algoritmus.
- Časová zložitosť O(n log n), priestorová náročnosť O(n).

- **Rezdel**: Rozdeľ kontajner rekurzívne na dve polovice, kým sa už nedá rozdeliť.
- **Panuj**: Každé čiastkové pole sa zoradí samostatne pomocou platného porovnania.
- **Zlúč**: Zoradené čiastkové polia sa opäť spoja do jedného v zoradenom poradí.
- Proces pokračuje, kým sa nezlúčia všetky prvky z oboch čiastkových polí.

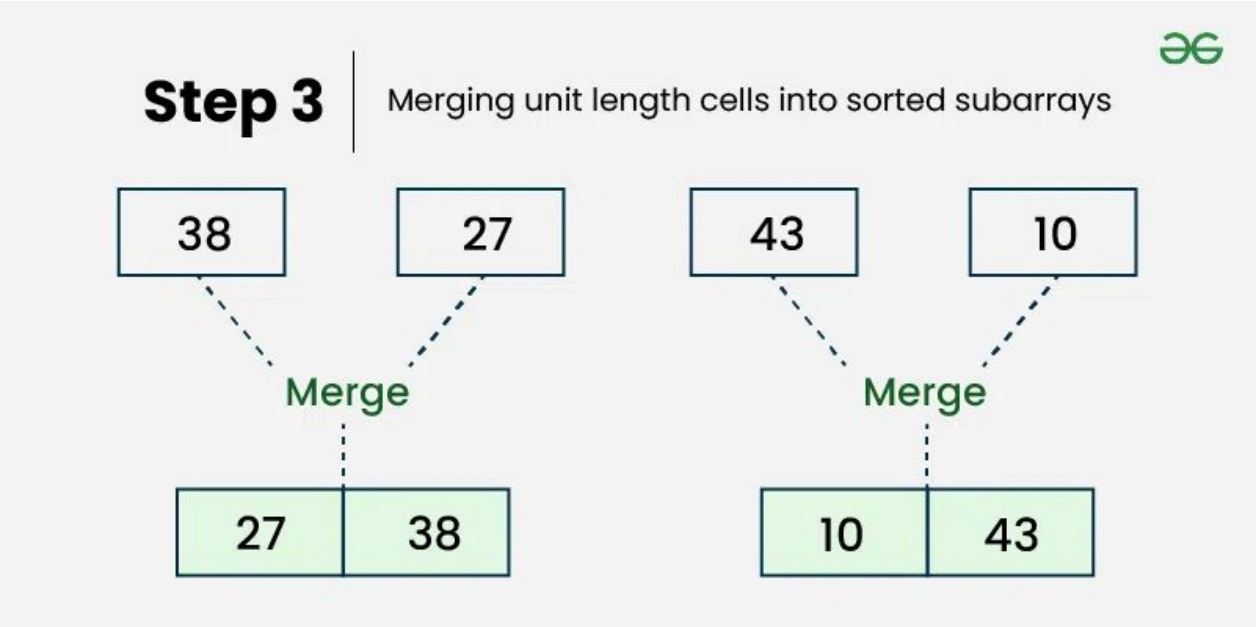
Merge Sort – pole [38, 27, 43, 10]



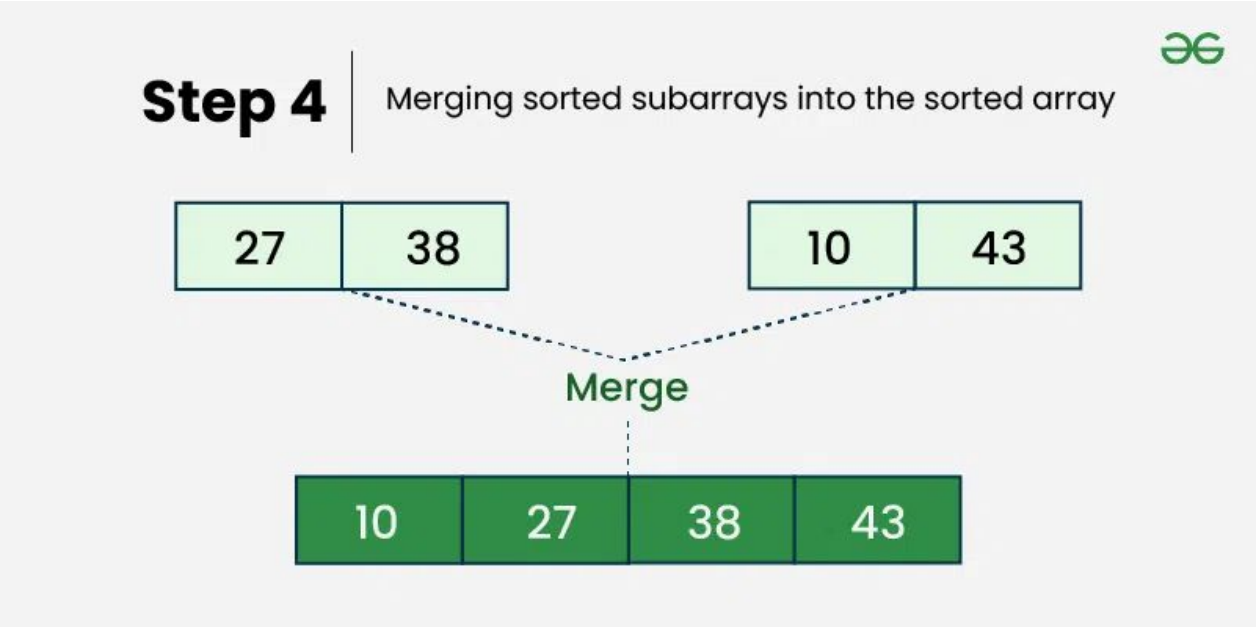
Merge Sort – pole [38, 27, 43, 10]



Merge Sort – pole [38, 27, 43, 10]



Merge Sort – pole [38, 27, 43, 10]



Merge Sort

- Stabilný - udržiava relatívne poradie rovnakých prvkov.
- Efektívny časovo aj na väčších data setoch avšak väčšia náročnosť na úložisko.
- Možné jednoducho paralelizovať.
- Reálne sa používa v programovacích jazykoch Python, Java, Swift.
- Jednoduché použitie aj pri zložitejších objektoch prvkov.
- Preferovaný pre zreťazené zoznamy.

Quick Sort

- Algoritmus založený na metóde Rozdeľuj a Panuj.
- Vyberie prvok ako pivot a rozdelí dané pole okolo vybraného pivotu umiestnením pivotu na správnu pozíciu v zoradenom poli.
- Časová zložitosť je O(n log n), priestorová náročnosť je O(log n).

- **Vyber pivot**: Vyberieme prvok z poľa ako pivot. Výber pivotu môže byť rôzny (napr. prvý prvok, posledný prvok, náhodný prvok alebo medián).
- **Rozdeľ**: Zmeníme usporiadanie poľa okolo pivotu. Po rozdelení budú všetky prvky menšie ako pivot na jeho ľavej strane a všetky prvky väčšie ako pivot budú na jeho pravej strane.Pivot je potom na správnej pozícii a získame index pivotu.
- **Rekurzívne volanie**: Rekurzívne aplikujeme rovnaký postup na dve rozdelené čiastkové polia (vľavo a vpravo od pivotu).
- **Základ**: Rekurzia sa zastaví, keď v podmnožine zostane len jeden prvok, pretože jeden prvok je už zoradený.

Quick Sort - Vyber pivot

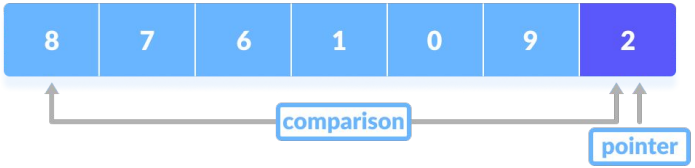
Existujú rôzne varianty quicksortu, pri ktorých sa pivot vyberá z rôznych pozícií. V tomto prípade budeme ako pivot vyberať najpravejší prvok poľa.

Quick Sort - Zotrieď prvky

Teraz sa prvky poľa preskupia tak, že prvky menšie ako pivot sa umiestnia na ľavú stranu a prvky väčšie ako pivot sa umiestnia na pravú stranu.

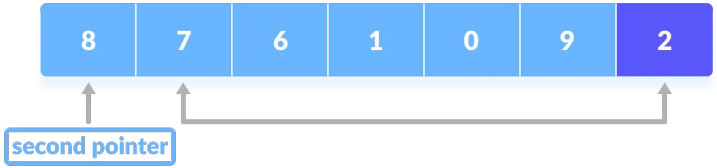
Quick Sort - Zotrieď prvky

Na pivot je upevnený ukazovateľ. Pivot sa porovnáva s prvkami začínajúcimi od prvého indexu.



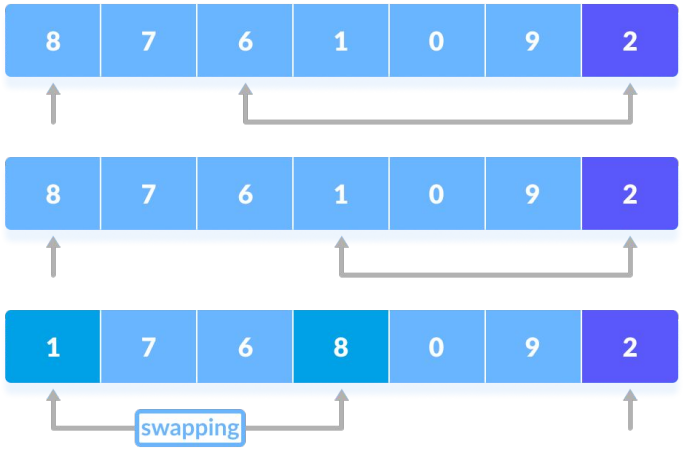
Quick Sort - Zotrieď prvky

Ak je prvok väčší ako pivot, druhý ukazovateľ je nastavený na tento prvok.



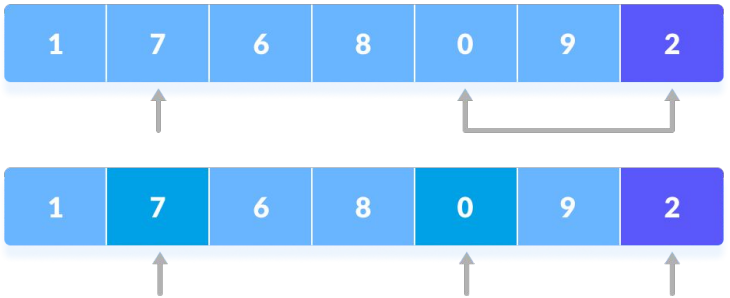
Quick Sort - Zotrieď prvky

Teraz sa pivot porovnáva s ostatnými prvkami. Ak sa dosiahne prvok menší ako pivot, menší prvok sa vymení za väčší prvok nájdený skôr (druhý ukazovateľ).



Quick Sort - Zotrieď prvky

Znovu sa process opakuje na nastavenie ďalšie väčšieho prvku od pivotu ako druhý ukazovateľ a následne je vymenený s iným menším prvkom.

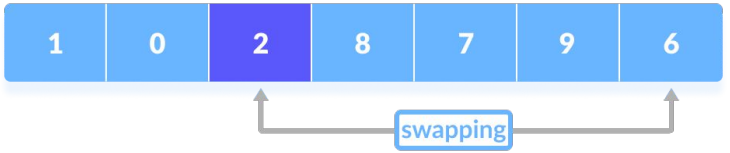


Quick Sort - Zotrieď prvky

Proces pokračuje až kým nenarazíme na prvok susediaci s pivotom.

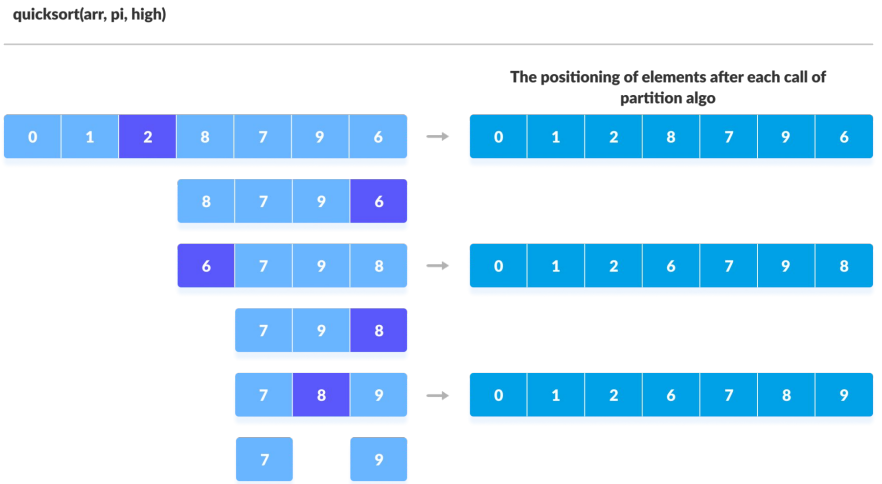
Quick Sort - Zotrieď prvky

Nakoniec je pivot prehodený s druhým ukazovateľom.



Quick Sort - Rozdeľ podpolia

Pole je rozdelené na dve časti podľa pivota. Nové pivoty sú určené na v ľavom a pravom podpoli a je opäť spustený proces **Zotrieď prvky**.



Quick Sort

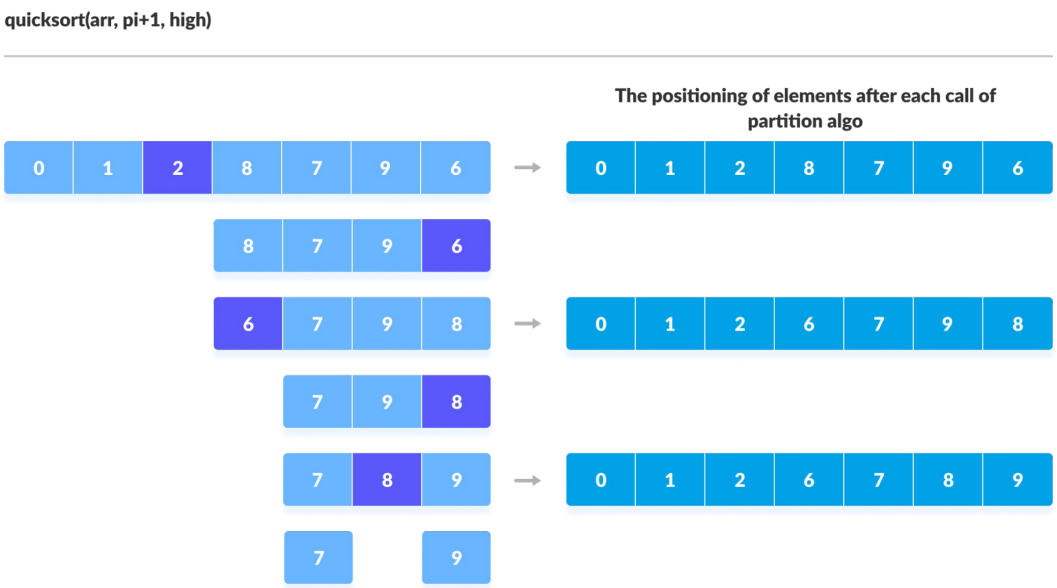
```
quickSort(array, leftmostIndex, rightmostIndex)
  if (leftmostIndex < rightmostIndex)
    pivotIndex <- partition(array,leftmostIndex,
rightmostIndex)
    quickSort(array, leftmostIndex, pivotIndex - 1)
    quickSort(array, pivotIndex, rightmostIndex)
```

```
partition(array, leftmostIndex, rightmostIndex)
  set rightmostIndex as pivotIndex
  storeIndex <- leftmostIndex - 1
  for i <- leftmostIndex + 1 to rightmostIndex
  if element[i] < pivotElement
    swap element[i] and element[storeIndex]
    storeIndex++
  swap pivotElement and element[storeIndex+1]
return storeIndex + 1
```

Quick Sort


quicksort(arr, low, pi-1)
8 7 6 1 0 9 2
1 0 2 8 7 9 6
1 0
0 1
1
The positioning of elements after each call of partition algo
1 0 2 8 7 9 6
0 1 2 8 7 9 6

Quick Sort


quicksort(arr, pi+1, high)
The positioning of elements after each call of partition algo
0 1 2 8 7 9 6
0 1 2 8 7 9 6
8 7 9 6
6 7 9 8
0 1 2 6 7 9 8
7 9 8
7 8 9
0 1 2 6 7 8 9
7
9

https://dsa.interes.group/exercises/exercise-5