Quick Sort

arr = [10, 80, 30, 90, 40]

def partition(arr, low, high):

pivot = arr[high]

i = low - 1

for j in range(low, high):

if arr[j] < pivot:

i += 1

arr[i], arr[j] = arr[j], arr[i]

arr[i + 1], arr[high] = arr[high], arr[i + 1]

return i + 1

def quick\_sort(arr, low, high):

if low < high:

pi = partition(arr, low, high)

quick\_sort(arr, low, pi - 1)

quick\_sort(arr, pi + 1, high)

quick\_sort(arr, 0, len(arr) - 1)

print(arr)

პირველი დაყოფა (ეს, რომ მორჩება შემდეგ ხდება რეკურსიულად გამოძახება და შემდეგ დაყოფაზე გადასვლა)

i = -1, j = 4

pivot = arr[4] = 40

ნაბიჯები

1. arr = [10, 80, 30, 90, 40]

10 < 40

i = 0, j = 0

arr[0], arr[0] = arr[0], arr[0] (არაფერი იცვლება)

1. arr = [10, 80, 30, 90, 40]

80 > 40 (არაფერი იცვლება)

i = 0, j = 1

1. arr = [10, 80, 30, 90, 40]

30 < 40

i = 1, j = 2

arr[1], arr[2] = arr[2], arr[1] (მივიღეთ arr = [10, 30, 80, 90, 40]

1. arr = [10, 30, 80, 90, 40]

90 > 40

i = 1, j = 3 (არაფერი იცვლება)

შემოვლა სრულდება, რადგან მივაღწიეთ პივოტამდე.

და პივოტი შეიცვლება - arr[2], arr[4] = arr[4], arr[2] (მივიღეთ arr = [10, 30, 40, 90, 80])

*ალგორითმის სირთულეები*

დროის სირთულე:

* საუკეთესო შემთხვევა: O(nlogn) - როცა პივოტი მასივს ჰყოფს თანაბარ ნაწილებად ყოველ გაყოფაზე
* უარესი შემთხვევა: O(nlogn) - როცა პივოტი არც ზედმეტად მცირეა და არც ზედმეტად დიდი
* საშუალო შემთხვევა: O(n^2) - როცა პივოტად შეირჩევა ყველაზე მცირე ან დიდი ელემენტი. ამ დროს ყოველ გაყოფისას მასივი მხოლოდ 1 ელემენტით მცირდება.

მეხსიერების სირთულე:

• Naive - O(n)

• Lomuto - O(logn)

• Hoare - O(logn)

საშუალოდ - O(logn)

Merge Sort

arr = [8, 4, 9, 5, 2, 5]

def merge\_sort(arr):

if len(arr) > 1:

mid = len(arr) // 2

left\_half = arr[:mid]

right\_half = arr[mid:]

merge\_sort(left\_half)

merge\_sort(right\_half)

i = j = k = 0

while i < len(left\_half) and j < len(right\_half):

if left\_half[i] < right\_half[j]:

arr[k] = left\_half[i]

i += 1

else:

arr[k] = right\_half[j]

j += 1

k += 1

while i < len(left\_half):

arr[k] = left\_half[i]

i += 1

k += 1

while j < len(right\_half):

arr[k] = right\_half[j]

j += 1

k += 1

return arr

print(merge\_sort(arr))

ეტაპი 1. ქვესიმრავლებად დაყოფა მანამ, სანამ არ გახდება 1 ელემენტიანი მასივი.

arr = [8, 4, 9, 5, 2, 5]

mid = len(arr) // 2 = 6 // 2 = 3

left\_half = arr[:3] 🡪 [8, 4, 9]

right\_half = arr[3:] 🡪 [5, 2, 5]

დავყოთ მარცხენა ნახევარი

[8, 4, 9] 🡪 [8], [4, 9] 🡪 [8], [4], [9]

დავყოთ მარჯვენა ნახევარი

[5, 2, 5] 🡪 [5], [2, 5] 🡪 [5], [2], [5]

ეტაპი 2. ქვემასივების გაერთიანება ელემენტების სწორ პოზიციაში ჩასმით. ვადარებთ და უმცირეს ელემენტებს ვსვამთ მარცხნივ.

[4] [9] 🡪 4 < 9 🡪 [4, 9]

[8] [4, 9] 🡪 8 > 4 (გავატარებთ); 8 < 9 (გავჩერდებით და ჩავსვამთ) 🡪 [4, 8, 9]

[2] [5] 🡪 2 < 5 🡪 [2, 5]

[5] [2, 5] 🡪 5 > 2 (გავატარებთ); 5 == 5 (გავჩერდებით და ჩავსვამთ) 🡪 [2, 5, 5]

საბოლოო გაერთიანება

[4, 8, 9] [2, 5, 5]

4 > 2 🡪 4 < 5 🡪 [2, 4, 5, 5]

8 > 2 🡪 8 > 4 🡪 8 > 5 🡪 8 > 5 🡪 [2, 4, 5, 5, 8]

9 > 2 🡪 9 > 4 🡪 9 > 5 🡪 9 > 5 🡪 9 > 8 🡪 [2, 4, 5, 5, 8, 9]

*ალგორითმის სირთულეები*

დროის სირთულე: O(nlogn), მასივის 2 ნაწილად გაყოფა ხდება რეკურსიულად და შემდგომ ქვემასივების გაერთიანება ხდება ელემენტების შედარებებით, რასაც O(n) დრო სჭირდება.

მეხსიერების სირთულე: O(n), დამხმარე მასივისთვის, რომელიც გაერთიანების პროცესში გამოიყენება.