# Quick Sorting Algoritması

Quick Sort Algoritması parçala fethet mantığıyla çalışır. Diziyi sıralarken her defasında parçalar ve sıralar, bu döngüyü dizideki elemanlar en küçükten en büyüğe doğru sıralanana kadar devam ettirir.

Dizide bulunan elemanlardan biri ‘pivot’ olarak seçilir. ‘Pivot’tan büyük olan elemanlar ‘pivot’un sağına, ‘pivot’tan küçük olan elemanlar ‘pivot’un soluna yerleştirilir. Daha sonra ‘pivot’un solundan ve sağından yeni ‘pivot’lar seçilir, bu yeni ‘pivot’lar kendi grupları içinde en başta olduğu gibi sıralanır. Bu ‘pivot’ seçimi ve sıralama işlemi dizi son halini alana kadar devam eder.

Örnek: [21, 33, 53, 97, 41, 10, 88]

1. Adım: Pivot 41 olsun; pivotun soluna pivottan küçükler, sağına pivottan büyükler dizideki sıralarına göre yerleşir.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 21 | 33 | 10 | 41 | 53 | 97 | 88 |

1. Adım: Küçükler arasındaki pivot 21, büyükler arasındaki pivot 88 olsun. Küçüklerin pivotuna göre sıralama kendi içinde son haline ulaşır, aynı durum büyüklerde de görülür. Sonuç olarak problem çözülür ve çıktı şu şekilde olur:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 21 | 33 | 41 | 53 | 88 | 97 |

Örnek Kod:

def quick\_sort(array): quick\_sort adlı fonksiyon tanımlanır

    if len(array) <= 1: fonksiyona verilen liste 0 veya 1 elemanlı ise

        return array girdide verilen listenin kendisi çıktı olarak döndürülür

    pivot = array[0] verilen listenin ilk elemanı “pivot” olarak seçilir

    lower = [] “pivot”tan küçükler için bir liste oluşturulur

    greater = [] “pivot”tan büyükler için bir liste oluşturulur

    for x in array: listedeki her eleman için

        if x < pivot: eleman “pivot”tan küçükse

            lower.append(x) lower adlı listeye eklenir

        elif x > pivot: eleman “pivot”tan büyükse

            greater.append(x) greater adlı listeye eklenir

    return quick\_sort(lower) + [pivot] + quick\_sort(greater) lower ve greater listeleri kendi içinde sıralanır ve “pivot” ile birleştirilir

number\_list = [7, 5, 2, 4, 0, 3, 9, 1, 6, 8]

print("Original List: ", number\_list)

sorted\_array = quick\_sort(number\_list)

print("Sorted List: ", sorted\_array)

# Selection Sorting Algoritması

Selection Sort Algoritması tüm listeyi tarar, en küçük sayıyı bulunca o sayı ile listenin en başındaki sayıyı yer değiştirir ve listeyi ikinci elemandan başlayacak biçimde böler. Tüm liste küçükten büyüğe doğru bu şekilde sıralanır.

Örnek: [-3, 5, 7, 1, 6, -9, 2, 8, 0]

1. Adım: Tüm liste tarandığında en küçük sayının -9 olduğu görülür, -9 ve -3 yer değiştirir ve artık bilgisayar listenin ilk elemanı olarak 5’i görür. Listenin yeni görünümü:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -9 | 5 | 7 | 1 | 6 | -3 | 2 | 8 | 0 |

1. Adım: Başlangıcı 5 olan listemiz tarandığında en küçük sayının -3 olduğu görülür, -3 ve 5 yer değiştirir ve bilgisayar 7’den başlayan yeni listeyi taramaya geçer. Listenin yeni görünümü:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -9 | -3 | 7 | 1 | 6 | 5 | 2 | 8 | 0 |

1. Adım: 7’den başlayan listenin en küçük elemanı 0’dır, 0 ve 7 yer değiştirir ve 1’den başlayan yeni liste taranır. Listenin yeni görünümü:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -9 | -3 | 0 | 1 | 6 | 5 | 2 | 8 | 7 |

1. Adım: 1’le başlayan listenin en küçüğü 1’dir dolayısıyla listenin görünümü değişmez. Bilgisayar 6’dan başlayan yeni listeyi taramaya geçer.
2. Adım: 6’dan başlayan yeni listenin en küçük elemanı 2’dir, 2 ve 6 yer değiştirir ve bilgisayar 5 ile başlayan yeni listeyi kontrol eder. Listenin yeni görünümü:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -9 | -3 | 0 | 1 | 2 | 5 | 6 | 8 | 7 |

1. Adım: 5 ile başlayan listenin en küçüğü 5’tir ve listenin görünümü değişmez. Bilgisayar 6 ile başlayan yeni listeyi tarar.
2. Adım: 6 ile başlayan listenin en küçüğü 6’dır ve listenin görünümü değişmez. Bilgisayar 8’den başlayan son listeyi tarar.
3. Adım: 8’den başlayan son taramada 7 ve 8 yer değiştirir liste son halini alır ve bilgisayar çıktı olarak bu listeyi verir:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -9 | -3 | 0 | 1 | 2 | 5 | 6 | 7 | 8 |

Örnek Kod:

def selection\_sort(array): selection\_sort adlı fonksiyon oluşturulur

    for i in range(len(array)-1): her bir liste elemanının indexi için (indexler 0’dan başlar listedeki eleman sayısının 1 eksiğinde sona erer)

        min\_index = i minimum index i olarak belirlenir

        for j in range(i+1, len(array)): i+1 indexli elemandan listenin sonuna kadar olan her j elemanı için

            if array[j] < array[min\_index]: listenin j indexli elemanı minimum indexli elemanından küçükse

                min\_index = j minimum index j olur

        array[i], array[min\_index] = array[min\_index], array[i] i indexli eleman ve minimum indexli eleman yer değiştirir

    return array liste tekrar okunur

number\_list = [0, 6, 4, -8, 5, 1, -7, 9, 3, -2]

print("Original List: ", number\_list)

sorted\_list = selection\_sort(number\_list)

print("Sorted List: ", sorted\_list)

# Bubble Sorting Algoritması

Bubble Sorting Algoritması listede ilk elemandan başlayarak ve her elemanı bir yanındaki elemanla kıyaslayarak sırlama yapan algoritmadır. Kıyaslanan elemanlardan büyük olan sağda yer alır, bu algoritmada adım sayısı ile listenin eleman sayısı aynıdır.

Örnek: [2, 5, 1, 3, 0, 4]

1. Adım: Karşılaştırma 2’den başlar; 2, 5’ten küçük olduğu için yer değiştirmezler ve karşılaştırma sırası 5’e geçer.
2. Adım: 5 listedeki en büyük eleman olduğu için listenin sonuna dek ilerler. Listenin yeni görünümü:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 1 | 3 | 0 | 4 | 5 |

1. Adım: 2, 1’den büyük olduğu için bir sağa kayar ama 3’ten küçük olduğu için sıra 3’e geçer. Listenin yeni görünümü:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 0 | 4 | 5 |

1. Adım: 3, 0’dan büyük olduğu için bir sağa kayar ve daha fazla ilerleyemez. Listenin yeni görünümü:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 2 | 0 | 3 | 4 | 5 |

1. Adım: 1, 2’den küçük olduğu için 2 ile kıyasa başlarız ve 2 ile 0 yer değiştirir. Listenin yeni görünümü:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | 0 | 2 | 3 | 4 | 5 |

1. Adım: Son karşılaştırmada 1 ve 0 yer değiştirir sonuç olarak bilgisayar bize çıktıyı verir:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 |

Örnek Kod:

def bubble\_sort(array): bubble\_sort adlı fonksiyon tanımlanır

    for \_ in range(len(array)): listedeki her eleman için (kodda bir daha kullanılmadığı için harf yerine \_ işareti koymak da bir seçenektir)

        for i in range(len(array)-1): (i index olacağı için) liste uzunluğundan bir eksik i değerleri için

            if array[i] > array[i+1]: i indexli eleman i+1 indexli elemandan büyükse

                array[i], array[i+1] = array[i+1], array[i] i indexli elemanla i+1 indexli eleman yer değiştirir

    return array liste tekrar okunur

number\_list = [7, 3, 8, 2, 0, 4, 1, 6, 9, 5]

print("Original List: ", number\_list)

sorted\_list = bubble\_sort(number\_list)

print("Sorted List: ", sorted\_list)