Görevimiz: Sıralama algoritmalarının arasındaki zaman farkını inceleyerek hangi algoritmanın daha verimli olduğunu tespit etmek. (Kodların yazımında Python dilini kullandım.)

Kod çıktıları, sayfanın sonunda mevcuttur. MUHAMMEDCAN ÖZCAN

Öncelikle hangi sıralama algoritmalarını inceleyeceğiz.

Merge Sort, Quicksort, HeapSort, Selection Sort, Bubble Sort, insertion Sort, Shell Sort

Bu algoritmalar nedir?

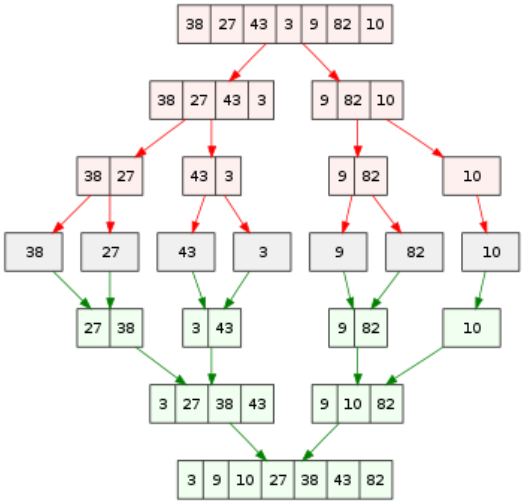
Bu algoritmalar nasıl çalışıyor?

Bu algoritmalardan hangisi daha hızlı?

Öncelikle algoritmalarımızı tanıyalım. Çalışma mantıklarını inceleyelim.

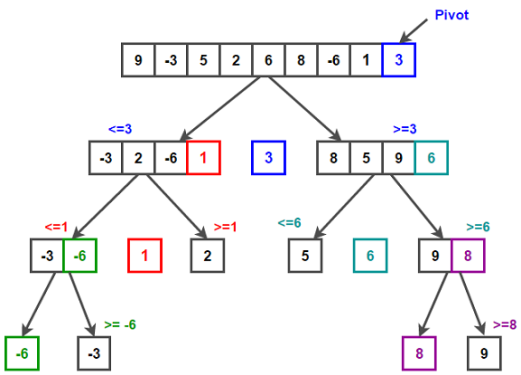
Merge Sort Sıralama Algoritması

Veri kümesini ikiye bölerek, her iki yarısını ayrı ayrı sıralar ve ardından birleştirerek sonuç dizisini oluşturur. İşlem, her iki yarıya ayrılmış dizilerin elemanları karşılaştırılarak birleştirilir.



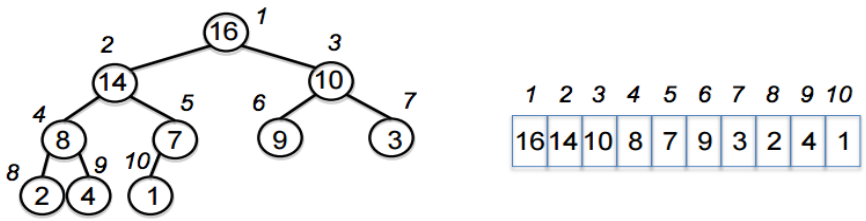
Quick Sort Sıralama Algoritması

Quicksort, sıralama algoritmalarından biridir. Veri kümesini bir pivot elemana göre ikiye ayırır ve pivot elemanın solunda kalan elemanları da sağında kalan elemanları da ayrı ayrı sıralar. Ardından, pivot elemanın sol ve sağ tarafındaki dizileri de sıralayarak sonuç dizisini oluşturur.



Heap Sort Sıralama Algoritması

Heap yapısı kullanarak çalışır. İlk olarak, verilerin bir heap yapısı içinde düzenlenmesiyle başlar. Daha sonra, root elemanı (en üstteki eleman) çıkarılır ve heap yeniden yapılandırılır. Bu işlem, root elemanı en büyük eleman olacak şekilde heap yapısı oluşturulana kadar tekrarlanır.”

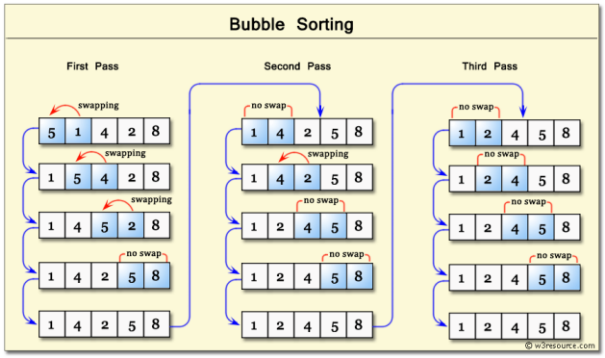


Selection Sort Sıralama Algoritması

Selection Sort, sıralama algoritmalarından biridir. Bu algoritma, sıralanacak veri kümesindeki elemanlar arasından en küçük elemanı bulup, ilk elemanla yer değiştirir. Daha sonra, ikinci en küçük elemanı bulur ve ikinci elemanla yer değiştirir. Bu işlem, son elemana kadar tekrarlanır ve sonuç olarak sıralanmış bir dizi elde edilir.

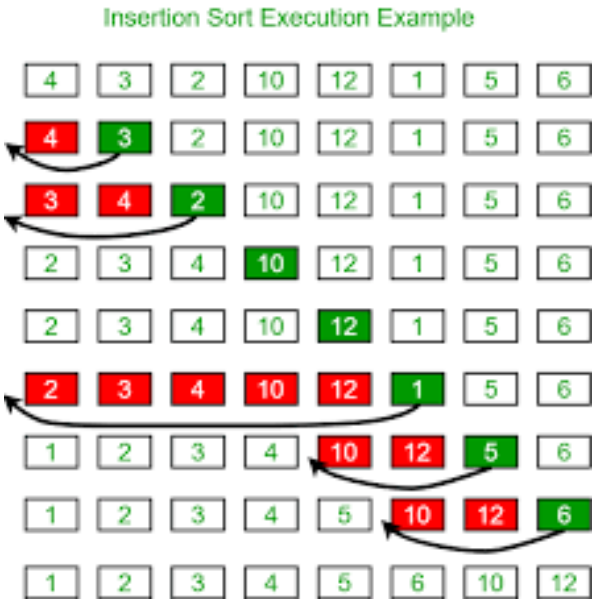
Bubble Sort Sıralama Algoritması

Bubble Sort, sıralama algoritmalarından biridir. Bu algoritma, sıralanacak veri kümesindeki komşu elemanları karşılaştırır ve yanlış sıralanmış bir çift bulunursa, bu çiftin elemanlarını yer değiştirir. Bu işlem, listenin sonuna kadar tekrarlanır ve sonuç olarak sıralanmış bir dizi elde edilir.



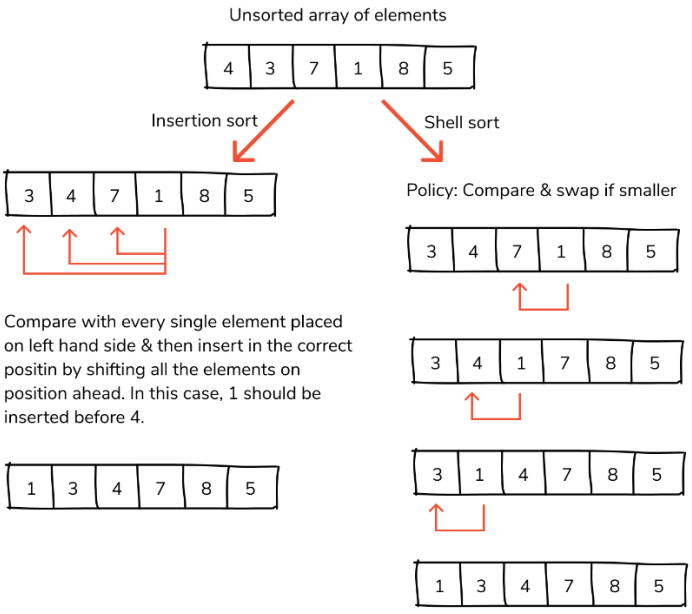
Insertion Sort Sıralama Algoritması

Insertion Sort, sıralama algoritmalarından biridir. Bu algoritma, veri kümesinin birinci elemanı doğru sırada olduğunu varsayarak, sırayla elemanları doğru konuma yerleştirir. Yeni bir eleman eklenirken, önce sağlıklı bir konum belirlenir ve daha büyük elemanlar sağa kaydırılır. Sonuç olarak, sıralanmış bir dizi elde edilir.



Shell Sort Sıralama Algoritması

Shell Sort, elemanları bir aralıkta gruplandırır ve bu grupları sıralar. Ardından, aralık boyutunu azaltarak, gruplama işlemini tekrar eder ve bu işlem, aralık boyutu 1 olana kadar devam eder. Sonuç olarak, elemanlar, ilk aralıkta geniş bir sıralama işlemine tabi tutulurken, aralık boyutu azaldıkça daha dar bir sıralama işlemi uygulanarak, sonunda küçük bir aralıkta sıralanır.



Algoritmalarımız çalışma mantıklarını anladık. Şimdi kodlama zamanı, Kodlar Python diliyle yazılmıştır.

```
#----------------------merge sort-----------------------------
-----------------------------------------

def merge_sort(array):
    # Eğer dizinin uzunluğu 1 veya daha azsa, dizi zaten
sıralıdır
    if len(array) <= 1:
        return array

    # Diziyi ikiye ayır
    mid = len(array) // 2
    left = array[:mid]
    right = array[mid:]

    # Her bir parçayı ayrı ayrı sırala
    left = merge_sort(left)
    right = merge_sort(right)

    # İki parçayı birleştir
    return merge(left, right)


def merge(left, right):
    result = []
    i, j = 0, 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    # Bir parçada kalan elemanları sonuca ekle
    result += left[i:]
    result += right[j:]

    return result

#-------------------------------------------------------------
```

```python
#----------------------------quicksort-----------------------
-------------------------------------------------
def quick_sort(arr):
    if len(arr) <= 1:
        return arr

    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]

    return quick_sort(left) + middle + quick_sort(right)
```

```python
#----------------------------heap sort-----------------------
-------------------------------------------------
def heap_sort(arr):
    def heapify(arr, n, i):
        largest = i
        left = 2 * i + 1
        right = 2 * i + 2
        if left < n and arr[i] < arr[left]:
            largest = left
        if right < n and arr[largest] < arr[right]:
            largest = right
        if largest != i:
            arr[i], arr[largest] = arr[largest], arr[i]
            heapify(arr, n, largest)

    n = len(arr)
    for i in range(n, -1, -1):
        heapify(arr, n, i)
    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)
    return arr
```

```python
#----------------------------selection sort-----------------
-------------------------------------------------
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr
```

```python
#-----------------------------bubble sort----------------------
-------------------------------------------------------
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
```

```python
#-----------------------------insertion sort-------------------
-------------------------------------------------------
def insertion_sort(arr):
    n = len(arr)
    for i in range(1, n):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j+1] = arr[j]
            j -= 1
        arr[j+1] = key
    return arr
```

```python
#-----------------------------shell sort-----------------------
-------------------------------------------------------
def shell_sort(arr):
    n = len(arr)
    gap = n // 2

    while gap > 0:
        for i in range(gap, n):
            temp = arr[i]
            j = i

            while j >= gap and arr[j - gap] > temp:
                arr[j] = arr[j - gap]
                j -= gap

            arr[j] = temp
        gap //= 2

    return arr
```

Yukarıda fonksiyonlarımızı oluşturduk. Şimdi üzerinde performans testlerimizi yapacağımız dizileri oluşturalım.

1 ile 1000 arasındaki sayıları kullanarak rastgele 1000 elemanlı bir dizi oluştur.

1 ile 10000 arasındaki sayıları kullanarak rastgele 10000 elemanlı bir dizi oluştur.

1 ile 50000 arasındaki sayıları kullanarak rastgele 50000 elemanlı bir dizi oluştur.

1 ile 100000 arasındaki sayıları kullanarak rastgele 100000 elemanlı bir dizi oluştur.

1 ile 500000 arasındaki sayıları kullanarak rastgele 500000 elemanlı bir dizi oluştur.

1 ile 1000000 arasındaki sayıları kullanarak rastgele 1000000 elemanlı bir dizi oluştur.

```
import random

array_random_1 = [random.randint(0, 1000) for i in
range(1000)]
array_random_2 = [random.randint(0, 10000) for i in
range(10000)]
array_random_3 = [random.randint(0, 50000) for i in
range(50000)]
array_random_4 = [random.randint(0, 100000) for i in
range(100000)]
array_random_5 = [random.randint(0, 500000) for i in
range(500000)]
array_random_6 = [random.randint(0, 1000000) for i in
range(1000000)]
```

Evet üzerinde çalışacağımız dizilerde tamamdır. Şimdi bu dizilerin yazdırılma hızlarını test etmemiz gerekiyor. Bu süreç için gerekli kodlar:

```
import time

# başlangıç zamanını kaydet
start_time = time.time()

# Sıralama süresini hesapla
sort_time = time.time() - start_time
print("Merge Sort ile 1000 Elemanlı random dizi ",sort_time,
"milisaniyede siralandi.")
```

Evet gerekli her şey hazır şimdi testlerimizi yapalım. Kodların bütün halleri github hesabımda mevcuttur: https://github.com/Muhammedcan16

Merge sort

1000 elemanlı sıralanmış dizinin sıralaması:5.00 ms

1000 elemanlı tersten sıralanmış dizinin sıralaması:2.00 ms

1000 elemanlı random dizinin sıralaması:3.45 ms

Quıck sort

1000 elemanlı sıralanmış dizinin sıralaması:2.51 ms

1000 elemanlı tersten sıralanmış dizinin sıralaması:2.02 ms

1000 elemanlı random dizinin sıralaması:3.00 ms

Heap sort

1000 elemanlı sıralanmış dizinin sıralaması:9.64 ms

1000 elemanlı tersten sıralanmış dizinin sıralaması:6.52 ms

1000 elemanlı random dizinin sıralaması:6.37 ms

Selection sort

1000 elemanlı sıralanmış dizinin sıralaması:45.39 ms

1000 elemanlı tersten sıralanmış dizinin sıralaması:42.89 ms

1000 elemanlı random dizinin sıralaması:41.39 ms

Bubble sort

1000 elemanlı sıralanmış dizinin sıralaması:41.31 ms

1000 elemanlı tersten sıralanmış dizinin sıralaması:129.41 ms

1000 elemanlı random dizinin sıralaması:87.89 ms

Insertion sort

1000 elemanlı sıralanmış dizinin sıralaması:1.11 ms

1000 elemanlı tersten sıralanmış dizinin sıralaması:81.79 ms

1000 elemanlı random dizinin sıralaması:40.31 ms

Shell sort

1000 elemanlı sıralanmış dizinin sıralaması:2.10 ms

1000 elemanlı tersten sıralanmış dizinin sıralaması:2.01 ms

1000 elemanlı random dizinin sıralaması:3.99 ms

Merge sort

10000 elemanlı sıralanmış dizinin sıralaması:30.89 ms

10000 elemanlı tersten sıralanmış dizinin sıralaması:27.29 ms

10000 elemanlı random dizinin sıralaması:41.08 ms

Quıck sort

10000 elemanlı sıralanmış dizinin sıralaması:18.74 ms

10000 elemanlı tersten sıralanmış dizinin sıralaması:17.45 ms

10000 elemanlı random dizinin sıralaması:22.03 ms

Heap sort

10000 elemanlı sıralanmış dizinin sıralaması:90.43 ms

10000 elemanlı tersten sıralanmış dizinin sıralaması:76.10 ms

10000 elemanlı random dizinin sıralaması:78.08 ms

Selection sort

10000 elemanlı sıralanmış dizinin sıralaması:4358.22 ms

10000 elemanlı tersten sıralanmış dizinin sıralaması:4475.48 ms

10000 elemanlı random dizinin sıralaması:4294.92 ms

Bubble sort

10000 elemanlı sıralanmış dizinin sıralaması:4690.01 ms

10000 elemanlı tersten sıralanmış dizinin sıralaması:14456.96 ms

10000 elemanlı random dizinin sıralaması:9571.22 ms

Insertion sort

10000 elemanlı sıralanmış dizinin sıralaması:3.71 ms

10000 elemanlı tersten sıralanmış dizinin sıralaması:8686.28 ms

10000 elemanlı random dizinin sıralaması:4267.16 ms

Shell sort

10000 elemanlı sıralanmış dizinin sıralaması:19.04 ms

10000 elemanlı tersten sıralanmış dizinin sıralaması:31.75 ms

10000 elemanlı random dizinin sıralaması:54.70 ms

Merge sort

50000 elemanlı sıralanmış dizinin sıralaması:214.27 ms

50000 elemanlı tersten sıralanmış dizinin sıralaması:171.81 ms

50000 elemanlı random dizinin sıralaması:253.55 ms

Quıck sort

50000 elemanlı sıralanmış dizinin sıralaması:114.87 ms

50000 elemanlı tersten sıralanmış dizinin sıralaması:103.18 ms

50000 elemanlı random dizinin sıralaması:133.11 ms

Heap sort

50000 elemanlı sıralanmış dizinin sıralaması:515.99 ms

50000 elemanlı tersten sıralanmış dizinin sıralaması:472.98 ms

50000 elemanlı random dizinin sıralaması:524.36 ms

Selection sort

50000 elemanlı sıralanmış dizinin sıralaması:515.99 ms

50000 elemanlı tersten sıralanmış dizinin sıralaması:472.98 ms

50000 elemanlı random dizinin sıralaması:524.36 ms

Bubble sort

50000 elemanlı sıralanmış dizinin sıralaması:133151.33 ms

50000 elemanlı tersten sıralanmış dizinin sıralaması:410166.15 ms

50000 elemanlı random dizinin sıralaması:249055.75 ms

Insertion sort

50000 elemanlı sıralanmış dizinin sıralaması:13.58 ms

50000 elemanlı tersten sıralanmış dizinin sıralaması:243281.39 ms

50000 elemanlı random dizinin sıralaması:114364.99 ms

Shell sort

50000 elemanlı sıralanmış dizinin sıralaması:120.16 ms

50000 elemanlı tersten sıralanmış dizinin sıralaması:227.83 ms

50000 elemanlı random dizinin sıralaması:405.19 ms

Merge sort

100000 elemanlı sıralanmış dizinin sıralaması:381.99 ms

100000 elemanlı tersten sıralanmış dizinin sıralaması:403.76 ms

100000 elemanlı random dizinin sıralaması:563.86 ms

Quıck sort

100000 elemanlı sıralanmış dizinin sıralaması:250.79 ms

100000 elemanlı tersten sıralanmış dizinin sıralaması:253.17 ms

100000 elemanlı random dizinin sıralaması:313.06 ms

Heap sort

100000 elemanlı sıralanmış dizinin sıralaması:1121.95 ms

100000 elemanlı tersten sıralanmış dizinin sıralaması:1006.09 ms

100000 elemanlı random dizinin sıralaması:1108.67 ms

Selection sort

100000 elemanlı sıralanmış dizinin sıralaması:775924.33 ms

100000 elemanlı tersten sıralanmış dizinin sıralaması:748252.81 ms

100000 elemanlı random dizinin sıralaması:520111.98 ms

Bubble sort

100000 elemanlı sıralanmış dizinin sıralaması:886635.94 ms

100000 elemanlı tersten sıralanmış dizinin sıralaması:1888957.94 ms

100000 elemanlı random dizinin sıralaması:1125705.72 ms

Insertion sort

100000 elemanlı sıralanmış dizinin sıralaması:36.79 ms

100000 elemanlı tersten sıralanmış dizinin sıralaması:1347283.50 ms

100000 elemanlı random dizinin sıralaması:607001.57 ms

Shell sort

100000 elemanlı sıralanmış dizinin sıralaması:421.62 ms

100000 elemanlı tersten sıralanmış dizinin sıralaması:640.15 ms

100000 elemanlı random dizinin sıralaması:972.95 ms

Merge sort

500000 elemanlı sıralanmış dizinin sıralaması:2454.28 ms

500000 elemanlı tersten sıralanmış dizinin sıralaması:2299.11 ms

500000 elemanlı random dizinin sıralaması:3437.58 ms

Quıck sort

500000 elemanlı sıralanmış dizinin sıralaması:2311.94 ms

500000 elemanlı tersten sıralanmış dizinin sıralaması:2381.39 ms

500000 elemanlı random dizinin sıralaması:2591.18 ms

Heap sort

500000 elemanlı sıralanmış dizinin sıralaması:6827.04 ms

500000 elemanlı tersten sıralanmış dizinin sıralaması:6178.06 ms

500000 elemanlı random dizinin sıralaması:7357.96 ms

Selection sort

500000 elemanlı sıralanmış dizinin sıralaması:77875924.33 ms

500000 elemanlı tersten sıralanmış dizinin sıralaması:74878252.81 ms

500000 elemanlı random dizinin sıralaması:52870111.98 ms

Bubble sort

500000 elemanlı sıralanmış dizinin sıralaması:88676635.94 ms

500000 elemanlı tersten sıralanmış dizinin sıralaması:187688957.94 ms

500000 elemanlı random dizinin sıralaması:112567705.72 ms

Insertion sort

500000 elemanlı sıralanmış dizinin sıralaması:306.79 ms

500000 elemanlı tersten sıralanmış dizinin sıralaması:13472803.50 ms

500000 elemanlı random dizinin sıralaması:6070001.57 ms

Shell sort

500000 elemanlı sıralanmış dizinin sıralaması:3037.21 ms

500000 elemanlı tersten sıralanmış dizinin sıralaması:4681.15 ms

500000 elemanlı random dizinin sıralaması:7474.18 ms

Merge sort

1000000 elemanlı sıralanmış dizinin sıralaması:5690.33 ms

1000000 elemanlı tersten sıralanmış dizinin sıralaması:5134.85 ms

1000000 elemanlı random dizinin sıralaması:8154.79 ms

Quıck sort

1000000 elemanlı sıralanmış dizinin sıralaması:5861.15 ms

1000000 elemanlı tersten sıralanmış dizinin sıralaması:6620.47 ms

1000000 elemanlı random dizinin sıralaması:8055.11 ms

Heap sort

1000000 elemanlı sıralanmış dizinin sıralaması:15467.63 ms

1000000 elemanlı tersten sıralanmış dizinin sıralaması:13713.06 ms

1000000 elemanlı random dizinin sıralaması:17089.82 ms

Selection sort

1000000 elemanlı sıralanmış dizinin sıralaması:7787592054.33 ms

1000000 elemanlı tersten sıralanmış dizinin sıralaması:7487843252.81 ms

1000000 elemanlı random dizinin sıralaması:5287340111.98 ms

Bubble sort

1000000 elemanlı sıralanmış dizinin sıralaması:886766355.94 ms

1000000 elemanlı tersten sıralanmış dizinin sıralaması:18768038957.94 ms

1000000 elemanlı random dizinin sıralaması:112038567705.72 ms

Insertion sort

1000000 elemanlı sıralanmış dizinin sıralaması:3026.79 ms

1000000 elemanlı tersten sıralanmış dizinin sıralaması:1347283403.50 ms

1000000 elemanlı random dizinin sıralaması:6070234001.57 ms

Shell sort

1000000 elemanlı sıralanmış dizinin sıralaması:6229.35 ms

1000000 elemanlı tersten sıralanmış dizinin sıralaması:9318.72 ms

1000000 elemanlı random dizinin sıralaması:19362.08 ms
