به نام خدا

Python Sort

Maryam Fakhraei & Amirhossein Naseri

Sort Methods

در پایتون دو تابع `sorted(list)` و `list.sort()` وجود دارد که برای مرتب سازی از آنها استفاده می شود.

1. `sorted(list)`

- ***Syntax*** : `sorted(list, key=…, reverse=…)`

- یک لیست مرتب شده جدید را برمی گرداند و لیست اصلی را تغییر نمی دهد.

In [3]:

```
list = [5, 2, 3, 1, 4]
sorted_list = sorted(list)
print(sorted_list)
```

```
[1, 2, 3, 4, 5]
```

- روی هر شی قابل تکرار کار می کند.

In [4]:

```
string = "This is a test string"
sorted_string = sorted(string)
print(sorted_string)
```

```
[' ', ' ', ' ', ' ', 'T', 'a', 'e', 'g', 'h', 'i', 'i', 'i', 'n', 'r',
's', 's', 's', 's', 't', 't', 't']
```

2. list.sort()

- ***Syntax*** : `list.sort(key=…, reverse=…)`

- لیست اصلی را در جای خود تغییر می دهد.

In [5]:

```
list = [5, 2, 3, 1, 4]
list.sort()
print(list)
```

```
[1, 2, 3, 4, 5]
```

- فقط روی لیست ها کار می کند.

In [6]:

```
string = "This is a test string"
string.sort()
print(string)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call las
t)
Cell In[6], line 2
      1 string = "This is a test string"
----> 2 string.sort()
      3 print(string)

AttributeError: 'str' object has no attribute 'sort'
```

Parameters

- **key**

`key` تابعی مانند `str.lower` ، `int` ، `len` و ... است که به عنوان کلید برای مقایسه عمل می کند و تعیین می کند که ورودی بر چه اساسی مرتب شود.

In [1]:

```
string = "This is a test string"
sorted_string_list = sorted(string.split(), key=str.lower)
print(sorted_string_list)
```

```
['a', 'is', 'string', 'test', 'This']
```

- **reverse**

پارامتر `reverse` ترتیب را مشخص می کند. مقدار پیش فرض `False` است که به معنای مرتب سازی به ترتیب صعودی است. با استفاده از `reverse=True` لیست را به صورت نزولی مرتب می کنیم.

In [2]:

```
list = [5, 2, 3, 1, 4]
ascending_sorted_list = sorted(list)
print(ascending_sorted_list)
descending_sorted_list = sorted(list, reverse=True)
print(descending_sorted_list)
```

```
[1, 2, 3, 4, 5]
[5, 4, 3, 2, 1]
```

Sort Algorithm

برای مرتب سازی در پایتون اگر طول آرایه ورودی کمتر از *64* کارکتر باشد تابع sort به سراغ الگوریتم `Insertion Sort` می رود اما اگر طول آرایه ورودی بیشتر از *64* کارکتر باشد تابع sort به سراغ الگوریتم `Tim Sort` می رود که الگوریتم ترکیبی ای از `Insertion Sort` و `Merge Sort` است.

1. Insertion Sort

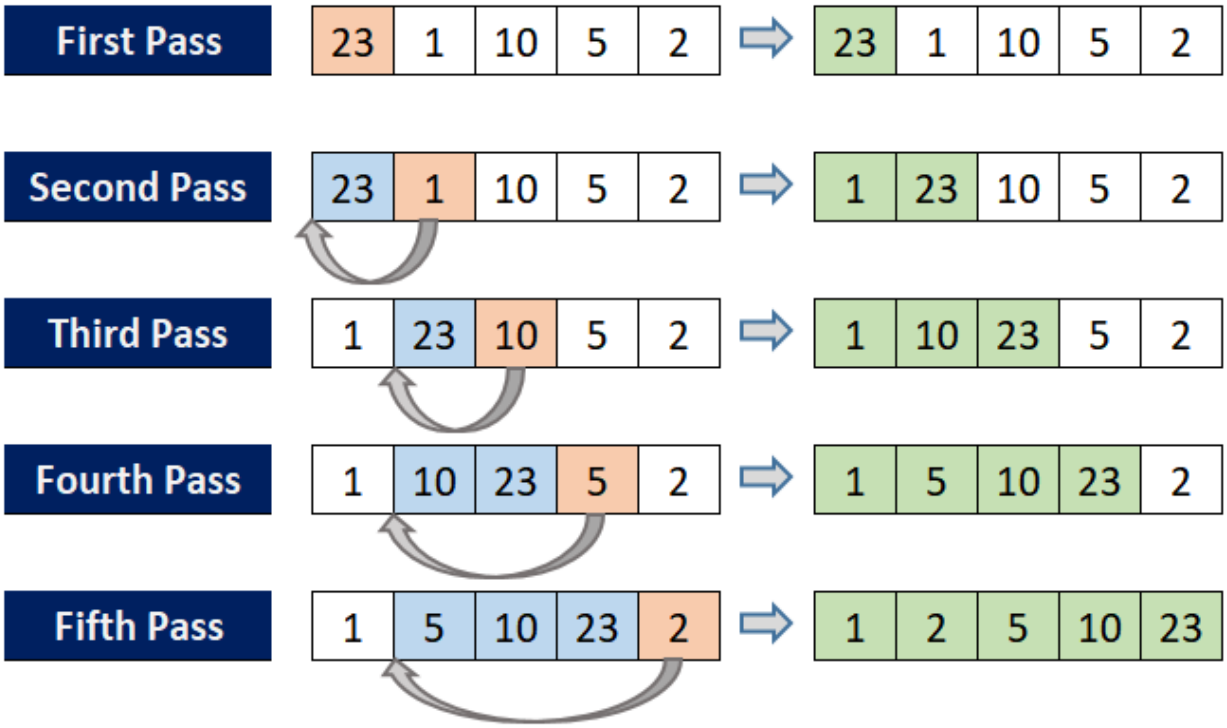
`Insertion Sort` یا مرتب سازی درجی یک الگوریتم مرتب سازی ساده است که با تقسیم آرایه ورودی به دو قسمت کار می کند: یک قسمت مرتب شده و یک قسمت مرتب نشده. الگوریتم روی هر عنصر در قسمت مرتب نشده آرایه تکرار می شود و آن را در موقعیت مناسب خود در قسمت مرتب شده آرایه قرار می دهد.

الگوریتم به صورت زیر کار میکند:

- فرض می شود اولین عنصر در آرایه از قبل مرتب شده است.
- برای درج هر عنصر، آن را با عناصر قبل از آن در قسمت مرتب شده آرایه مقایسه می کنیم.
- هر عنصر بزرگتر از آن را با یک موقعیت به سمت راست حرکت داده (shift right) تا فضا برای عنصر جدید باز شود.
- عنصر جدید را در جای مناسب خود در قسمت مرتب شده آرایه قرار داده و به تکرار این روند در قسمت مرتب نشده آرایه ادامه می دهیم تا زمانی که همه عناصر در قسمت مرتب شده آرایه درج شوند.
- در پایان الگوریتم، کل آرایه به ترتیب صعودی مرتب می شود.

مرتب سازی درجی دارای پیچیدگی زمانی $O(n^2)$ در بدترین حالت و پیچیدگی زمانی $O(n)$ در بهترین حالت است، و یک الگوریتم در محل (in-place) است، به این معنی که به حافظه اضافی نیاز ندارد. با وجود سادگی، مرتب سازی درجی می تواند سریع تر از الگوریتم های مرتب سازی پیچیده تر مانند `Quick sort` یا `Merge sort` برای آرایه های کوچک یا آرایه های جزئی مرتب شده باشد.

Best-Case Complexity	Average Complexity	Worst-Case Complexity
$O(n)$	$O(n^2)$	$O(n^2)$



In [6]:

```
def InsertionSort(A):
    for k in range(1, len(A)):
        item = A[k]
        i = k
        while i > 0 and A[i-1] > item:
            A[i] = A[i-1]
            i -= 1
        A[i] = item
    return A

A = [6, 5, 3, 1, 8, 7, 2, 4]
print(InsertionSort(A))
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

6 5 3 1 8 7 2 4

2. Merge Sort

Merge Sort یا مرتب سازی ادغامی یکی از مؤثرترین تکنیک های مرتب سازی برمبنای تکنیک «تقسیم وحل» (Divide and Conquer) است.

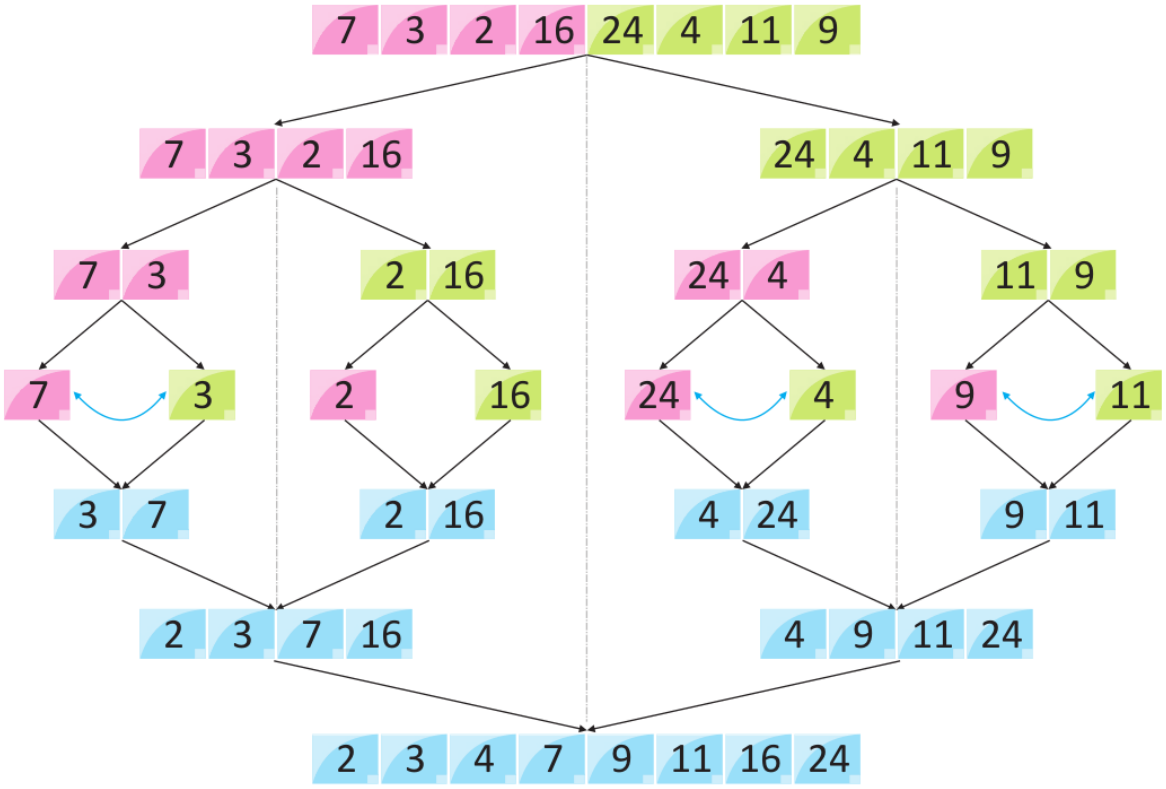
مرتب سازی ادغامی یک الگوریتم «تقسیم و حل» است که در آن ابتدا مسئله به مسائل فرعی تقسیم می شود. زمانی که راه حل ها برای مسائل فرعی آماده شد، مجدداً آن ها را با هم ترکیب می کنیم تا راه حل نهایی برای مسئله اصلی به دست آید.

Merge Sort یکی از الگوریتم هایی است که با استفاده از «بازگشت» (Recursion) به سادگی پیاده سازی می شود، چون به جای مسئله اصلی با مسائل فرعی سر و کار داریم.

الگوریتم آن را می توان به صورت فرایند دو مرحله ای زیر توصیف کرد:

- تقسیم: در این مرحله آرایه ورودی به دو نیمه تقسیم می شود. محور تقسیم نقطه میانی آرایه است. این مرحله به صورت بازگشتی روی همه آرایه های نیمه انجام می یابد تا این که دیگر نیمه آرایه ای برای تقسیم وجود نداشته باشد.
- حل: در این مرحله باید آرایه های تقسیم شده را مرتب سازی و ادغام کنیم و این کار از بخش زیرین به سمت بالا برای به دست آوردن آرایه مرتب انجام می یابد.

Best-Case Complexity	Average Complexity	Worst-Case Complexity
$O(n \lg(n))$	$O(n \lg(n))$	$O(n \lg(n))$



In [5]:

```
def MergeSort(A):
    if len(A) < 2:
        return A
    mid = len(A) // 2
    B = A[:mid]
    C = A[mid:]
    B = MergeSort(B)
    C = MergeSort(C)
    i = j = 0
    A = []
    while i < len(B) and j < len(C):
        if B[i] <= C[j]:
            A += [B[i]]
            i += 1
        else:
            A += [C[j]]
            j += 1
    A += B[i:] + C[j:]
    return A

A = [6, 5, 3, 1, 8, 7, 2, 4]
print(MergeSort(A))
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

3. Tim Sort

در پایتون، تابع sort داخلی از یک الگوریتم مرتب سازی ترکیبی به نام `Tim sort` استفاده می کند.

`Timsort` یک الگوریتم مرتب سازی ترکیبی است که از `Merge Sort` و `Insertion Sort` گرفته شده است و با تقسیم لیست ورودی به زیر لیست های کوچک تر و سپس مرتب سازی لیست های فرعی با استفاده از مرتب سازی درجی، و در نهایت ادغام لیست های فرعی مرتب شده با یکدیگر با استفاده از مرتب سازی ادغامی کار می کند.

- معمولاً روی داده هایی که دارای نظم یا ساختار ذاتی هستند، مانند داده های طبقه بندی شده یا داده هایی با مقادیر منحصر به فرد کم، بهترین عملکرد را دارد.
- در سال 2002 توسط Tim Peters برای استفاده در زبان برنامه نویسی پایتون اختراع شد.
- دلیل استفاده از `Tim sort` در تابع sort پایتون این است که عملکرد سریعی در حالت میانگین را ارائه می کند، در حالی که همچنان پایدار(یعنی ترتیب نسبی عناصر برابر را در لیست ورودی حفظ می کند) و تطبیقی(یعنی می تواند رفتار خود را بر اساس ویژگی های داده های ورودی تنظیم کند)است.
- علاوه بر این، نشان داده شده است که Tim sort بر روی طیف گسترده ای از مجموعه داده های دنیای واقعی موثر است، و آن را به یک انتخاب همه جانبه خوب برای تابع sort در پایتون تبدیل می کند.

برخلاف سایر الگوریتم های مرتب سازی مانند Quick Sort یا Heap Sort که دارای پیچیدگی های زمانی $O(n^2)$ در بدترین حالت هستند، Tim sort دارای پیچیدگی زمانی $O(n \lg(n))$ در بدترین حالت و $O(n)$ در بهترین حالت است.

Best-Case Complexity	Average Complexity	Worst-Case Complexity
$O(n)$	$O(n \lg(n))$	$O(n \lg(n))$

In [7]:

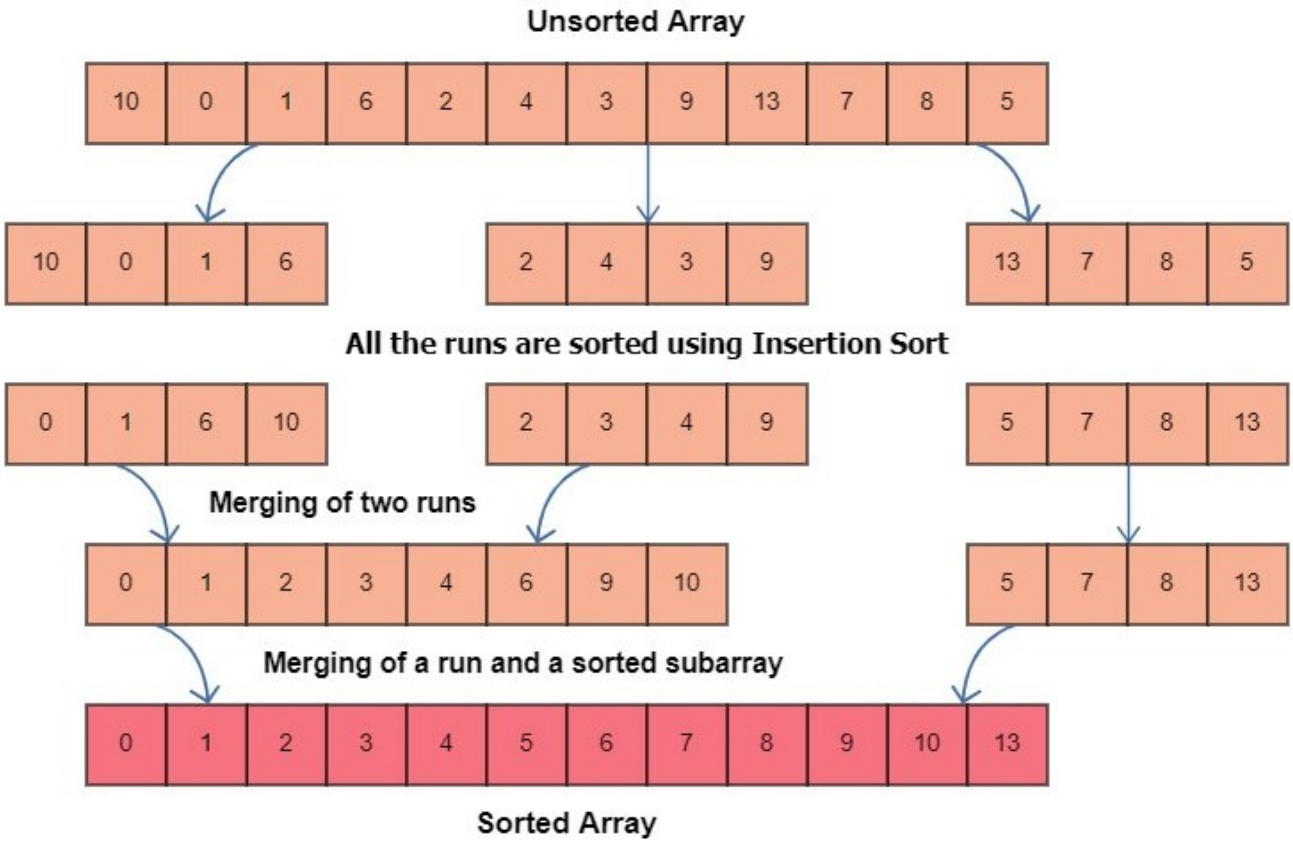
```
def tim_sort(arr):
    minrun = 32
    n = len(arr)
    for i in range(0, n, minrun):
        insertion_sort(arr, i, min((i+minrun-1), n-1))
    size = minrun
    while size < n:
        for start in range(0, n, size*2):
            mid = start + size - 1
            end = min((start + size*2 - 1), (n-1))
            merged_arr = merge_sort(arr, start, mid, end)
            arr[start:start+len(merged_arr)] = merged_arr
        size *= 2
    return arr

def insertion_sort(arr, left, right):
    for i in range(left+1, right+1):
        key_item = arr[i]
        j = i - 1
        while j >= left and arr[j] > key_item:
            arr[j+1] = arr[j]
            j -= 1
        arr[j+1] = key_item

def merge_sort(arr, start, mid, end):
    merged_arr = []
    left = arr[start:mid+1]
    right = arr[mid+1:end+1]
    while left and right:
        if left[0] < right[0]:
            merged_arr.append(left.pop(0))
        else:
            merged_arr.append(right.pop(0))
    merged_arr.extend(left)
    merged_arr.extend(right)
    return merged_arr

A = [10, 0, 1, 6, 2, 4, 3, 13, 7, 8, 5]
print(tim_sort(A))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 13]
```


Unsorted Array
10 0 1 6 2 4 3 9 13 7 8 5
10 0 1 6
2 4 3 9
13 7 8 5
All the runs are sorted using Insertion Sort
0 1 6 10
2 3 4 9
5 7 8 13
Merging of two runs
0 1 2 3 4 6 9 10
5 7 8 13
Merging of a run and a sorted subarray
0 1 2 3 4 5 6 7 8 9 10 13
Sorted Array