Recursion & Sorting

Sadaf Sadeghian

روش حل مسائل بازگشتی

برای حل مسائل بازگشتی لازم است دو چیز را در مسئله پیدا کنیم:

۱) **شرط خاتمه** : جایی از مسئله که به علت وجود شرایطی خاص می‌توانیم مسئله را حل کنیم و در واقع حالت پایه مسئله ماست.

۲) **قدم** : تبدیل مسئله به تعدادی زیر مسئله مشابه و کوچکتر

بررسی Palindrome بودن یک عدد

یک عدد در صورتی Palindrome است که عدد را از راست به چپ و از چپ به راست بخوانیم در هر دو صورت یک مقدار باشد.

حال کدی بزنید که عددی را از ورودی بگیرد و در صورت palindrome بودن عدد به عنوان جواب **Yes** را چاپ کند و در غیر این صورت **No** را چاپ کند.

مثال)

- 12321 عددی palindrome است.
- 54325 عدد palindrome نیست.

بررسی Palindrome بودن یک عدد

شرط خاتمه :

- عدد یک رقمی، همیشه Palindrome است.
- در صورتی که تمامی ارقام بررسی شده باشد و رقمی برای بررسی باقی نمانده باشد(تعداد ارقام عدد زوج باشد) عدد Palindrome است.

قدم :

کافی است هر بار دو رقم، یکی از ابتدا و دیگری از انتها را بررسی کنیم:

- در صورتی که برابر نباشند، عدد Palindrome نیست.
- در صورتی که برابر باشند، لازم است ارقام بعدی بررسی شوند.

بررسی Palindrome بودن یک عدد

```python
def is_palindrome(num):
    if len(num)==0 or len(num)==1:
        return True
    if not num[0]==num[-1]:
        return False
    return is_palindrome(num[1:len(num)-1])

num = input()
if(is_palindrome(num)):
    print("Yes")
else:
    print("No")
```

بررسی k-palindrome بودن یک رشته

یک رشته k-palindrome است اگر با حذف حداکثر k کاراکتر از رشته، به رشته‌ای palindrome برسیم.

حال کدی بزنید که یک رشته و عدد k را از ورودی بگیرد و در صورت k-palindrome بودن رشته **Yes** را چاپ کند و در غیر این صورت **No** را چاپ کند.

مثال)

- رشته ABCDBA با k=1 خروجی Yes را می‌دهد.
- رشته ABCDECA با k=1 خروجی No را می‌دهد.

بررسی k-palindrome بودن یک رشته

شرط خاتمه :

به رشته‌ای به طول ۰ یا ۱ رسیده باشیم و تا به الان حداکثر k حرف را حذف کرده‌ایم.

قدم :

دو کاراکتر ابتدا و انتهای رشته را مقایسه می‌کنیم:

- اگر برابر بودند، همین تابع را با عدد k و رشته از کاراکتر دوم تا یکی مانده به آخر صدا می‌زنیم.
- اگر بربر نبودند، همین تابع را با عدد k-1 و رشته از کاراکتر دوم تا آخر و یکبار دیگر با k-1 و رشته از کاراکتر اول تا یکی مانده به آخر صدا می‌زنیم.

بررسی k-palindrome بودن یک رشته

```python
def is_k_palindrome(string, k):
    if len(string)==0 or len(string)==1:
        if k>=0:
            return True
        else:
            return False
    if string[0]==string[-1]:
        return is_k_palindrome(string[1:len(string)-1], k)
    else:
        if is_k_palindrome(string[1:len(string)], k-1):
            return True
        elif is_k_palindrome(string[0:len(string)-1], k-1):
            return True
        return False

string = input()
k = int(input())
if(is_k_palindrome(string, k)):
    print("Yes")
else:
    print("No")
```

اعداد n رقمی با مجموع ارقام دلخواه

با گرفتن n و sum به عنوان ورودی، تمامی اعداد n رقمی را بیابید که مجموع ارقام آن اعداد برابر با s باشد.

$$1 \leq n \leq 9$$

$$1 \leq s \leq 81$$

مثال)

برای مثال با ورودی‌های n=3 و s=6 خروجی اعداد زیر خواهند بود:

105 114 123 132 141 150 204 213 222 231 240
303 312 321 330 402 411 420 501 510 600

اعداد n رقمی با مجموع ارقام دلخواه

شرط خاتمه :

در عدد ساخته شده تعداد ارقام برابر با n باشد و مجموع ارقام برابر با s

قدم :

از سمت چپ یک رقم را مقدار دهی کرده (مثلا i) و حالا لازم است یک عدد n-1 رقمی با مجموع ارقام s - i را بسازیم.

(دقت کنید سمت چپ ترین رقم نمی‌تواند ۰ باشد)

اعداد n رقمی با مجموع ارقام دلخواه

```python
def print_num(digits):
    res = ""
    for digit in digits:
        res += str(digit)
    print(res)

def find_n_digits_nums_withsum_s(n, s, index, num):
    if index==n and s==0:
        print_num(num)
    if index<n and s>=0:
        for d in range(0, 10):
            if index==0 and d==0:
                continue
            num[index] = d
            find_n_digits_nums_withsum_s(n, s-d, index+1, num)

n = int(input())
s = int(input())
num = [0] * n
find_n_digits_nums_withsum_s(n, s, 0, num)
```

یافتن تعداد نابه جایی‌ها در آرایه اعداد

فرض کنید آرایه‌ای از اعداد به شکل A1, A2, A3, …, An داریم. تعداد نابه‌جایی‌ها برابر با تعداد جفت‌های (i, j) ای هستند که $1 \leq i < j \leq n$ و $A[i] > A[j]$

شما باید در خط اول n خط بعدی n عدد که عناصر آرایه به ترتیب هستند را بگیرید و تعداد نابه‌جایی ها را چاپ کنید.

* در واقع تعداد نابه‌جایی‌ها نشان می‌دهد آرایه چقدر با آرایه مرتب شده فاصله دارد. برای مثال در یک آرایه مرتب تعداد نابه‌جایی‌ها برابر با 0 و در آرایه کاملا برعکس و نزولی تعداد نابه‌جایی ماکزیمم است.

مثال)

برای ورودی n=5 و آرایه 4 5 1 3 2 خروجی برابر با 3 می‌باشد.

یافتن تعداد نابه جایی‌ها در آرایه اعداد

فرض کنید آرایه را به دو زیر آرایه، یکی از عنصر ۱ تا n/2 و دیگری از n/2+1 تا آخر تقسیم کنیم.

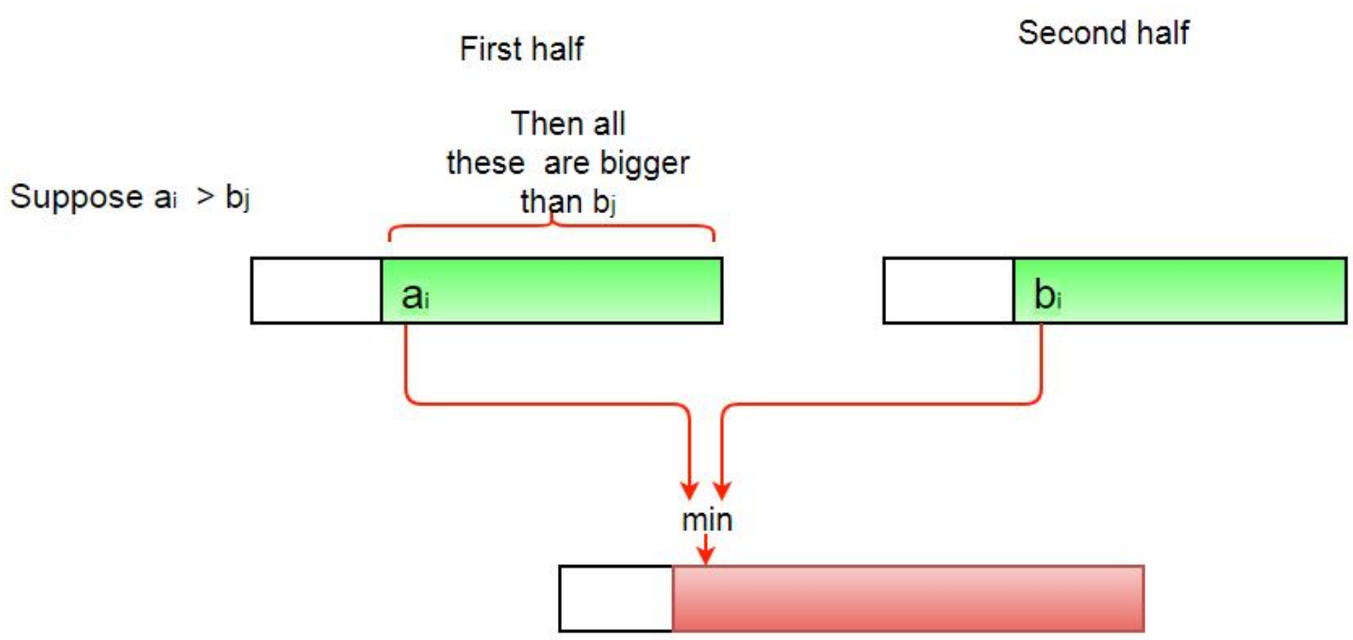
با فرض داشتن تعداد نابه‌جایی‌های دو زیر آرایه با مقادیر inv_r برای زیر آرایه سمت راست و inv_l برای زیر آرایه سمت چپ، تعداد کل نابه‌جایی‌های آرایه اولیه برابر با :

inv_l + inv_r + تعداد نابه‌جایی‌هایی که لازم است مرحله merge دو زیر آرایه شمرده شود.

یافتن تعداد نابه جایی‌ها در آرایه اعداد

شمارش تعداد نابه‌جایی‌ها در مرحله merge:

اگر از i برای ایندکس زیر آرایه چپ و از j برای ایندکس زیر آرایه راست استفاده کنیم. در هر مرحله از merge زمانی که A[i] > A[j] در این مرحله mid - i به تعداد نابه‌جایی‌هایمان اضافه می‌شود. چرا که هر دو زیر آرایه مرتب هستند پس اگر A[i] بزرگتر از A[j] است پس تمامی عناصر بعدی در زیر آرایه چپ شامل A[mid] ,…, A[i+2], A[i+1] همگی از A[j] بزرگتر هستند.



یافتن تعداد نابه جایی‌ها در آرایه اعداد

```python
def count_inversions(arr):
    if len(arr) == 1:
        return arr, 0


    else:
        a = arr[ : len(arr)//2]
        b = arr[len(arr)//2 : ]
        a, ai = count_inversions(a)
        b, bi = count_inversions(b)
        c = []
        i = 0
        j = 0
        inversions = ai + bi

    while i < len(a) and j < len(b):
        if a[i] <= b[j]:
            c.append(a[i])
            i += 1
        else:
            c.append(b[j])
            j += 1
            inversions += (len(a)-i)

    c += a[i:]
    c += b[j:]
    return c, inversions

n = int(input())
arr = list(map(int,input().split()))
print(count_inversions(arr)[1])
```