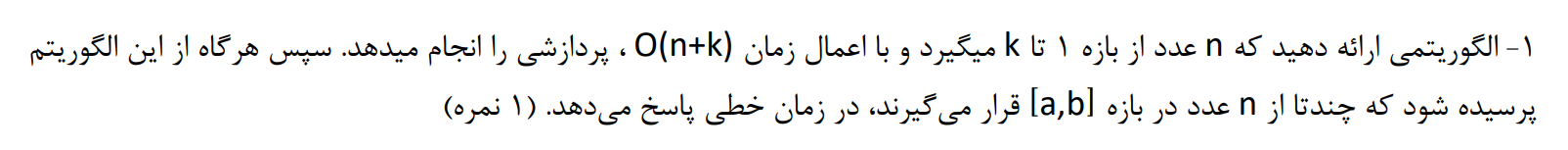
به نام خدا

تمرین شماره ۲ درس ساختمان داده‌ها و الگوریتم‌ها

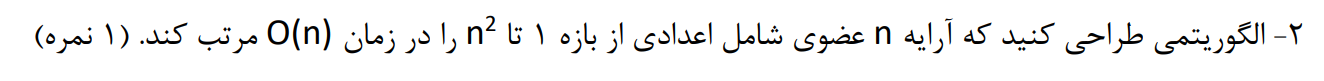


آرایه‌ی داده شده را input\_array می‌نامیم.

ابتدا مرحله‌ی اولِ counting sort را انجام می‌دهیم. به این شکل که یک آرایه به نام sorted\_array و به طولِ k + 1 تشکیل می‌دهیم و مقدار تمام عناصر آن را 0 در نظر می‌گیریم. سپس یک دور روی تمامِ عناصر input\_array پیمایش می‌کنیم و به هر عنصر که رسیدیم (اگر آن عنصر را i بنامیم)، یک واحد به sorted\_array[i] اضافه می‌کنیم.

این پردازش O(n+k) زمان می‌گیرد.

حال هر گاه از الگوریتم ما پرسیده شود که چندتا از n عدد در بازه‌ی [a, b] قرار می‌گیرند، جمعِ خانه‌های sorted\_array[a] تا sorted\_array[b] را محاسبه می‌کند و پاسخ می‌دهد.



این سوال را با ترکیبی از counting sort و Radix sort انجام می‌دهیم.

می‌توانیم تمامِ اعداد را به این فرم بنویسیم:

با توجه به این که هیچ یک از اعداد، بزرگ‌تر از نیست، مقدار a1 نمی‌تواند بزرگ‌تر از n باشد. از طرفی هم وقتی n را ماکسیمم مقدار مجاز خود بنویسیم، a2 برابر خواهد بود با a % n ، پس بدیهی‌ست که هرگز نتواند بزرگ‌تر از n باشد.

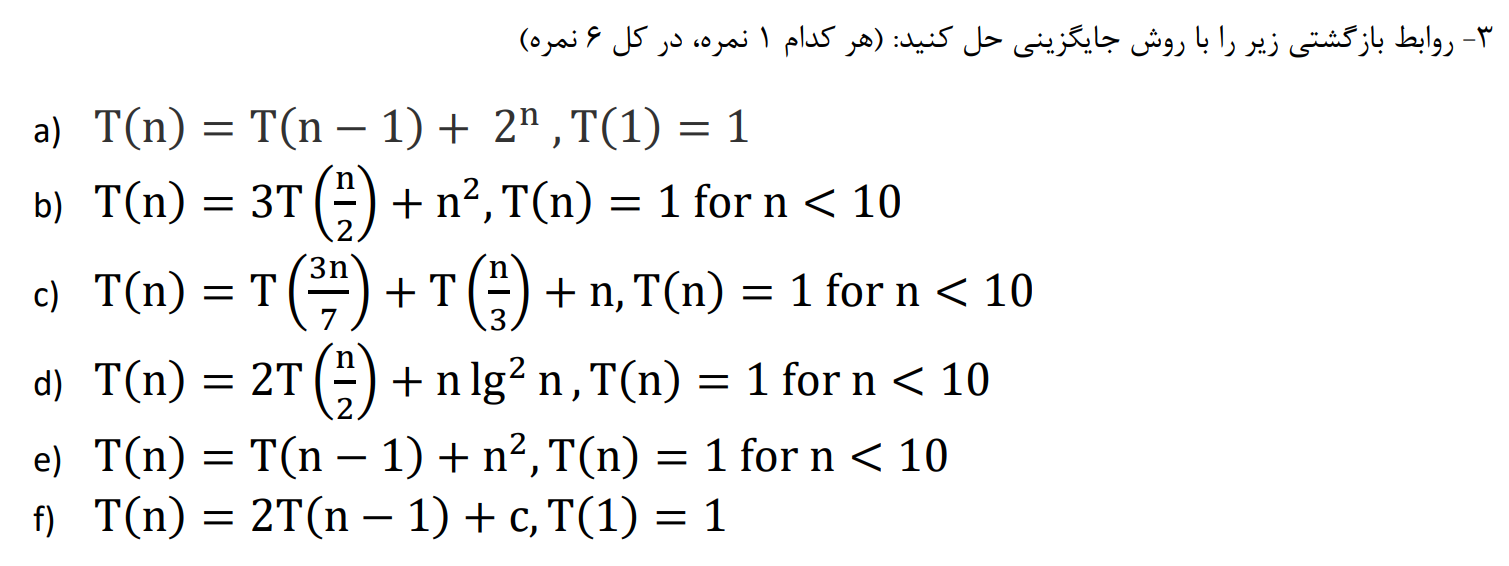
تا این‌جای کار، توانستیم هر یک از عناصرِ آرایه‌مان رو، تبدیل به دو عنصرِ دیگر کنیم که هیچ‌یک بزرگ‌تر از n نباشد. حالا می‌توانیم از counting sort استفاده کنیم، به این صورت که یک بار عناصر را بر حسب a1 هایشان، و بارِ دیگر آن‌ها را بر حساب a2 هایشان مرتب کنیم.

در مرحله‌ی اول، عناصر بر حسبِ باقیمانده‌شان بر n مرتب خواهند شد و در مرحله‌ی دوم، براساس بریده شده‌ی جوابِ تقسیم‌شان بر n ، به این ترتیب بعد از این دو مرحله آرایه‌مان کاملا مرتب خواهد بود.

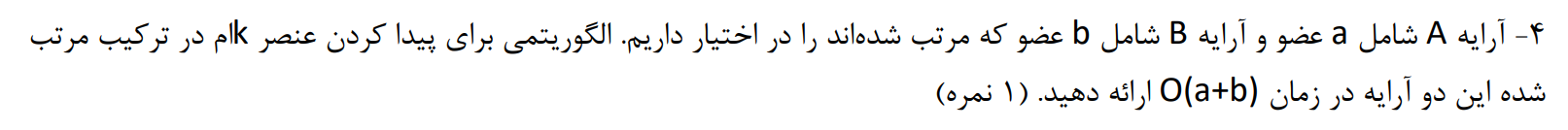
همچنین مجموعا الگوریتم‌مان O(2n) زمان خواهد برد که می‌دانیم معادل است با همان O(n) .

کدِ این الگوریتم به زبان پایتون نیز، به این شکل خواهد بود (در صفحه‌ی بعد):

import math  
  
numbers = [int(x) for x in input().split()]  
n = len(numbers)  
  
counting\_list = []  
sorted\_list = []  
  
for i in range(n):  
 counting\_list.append(0)  
 sorted\_list.append(0)  
  
# 1st counting sort: based on "a2"  
  
for number in numbers:  
 a2 = number % n  
 counting\_list[a2] += 1  
  
for i in range(1, n):  
 counting\_list[i] += counting\_list[i - 1]  
  
for i in range(n - 1, -1, -1):  
 a2 = numbers[i] % n  
 sorted\_list[counting\_list[a2] - 1] = numbers[i]  
 counting\_list[a2] -= 1  
  
for i in range(n):  
 numbers[i] = sorted\_list[i]  
  
# 2nd counting sort: based on a1  
  
for i in range(n):  
 counting\_list[i] = 0  
 sorted\_list[i] = 0  
  
for number in numbers:  
 a1 = (math.trunc(number / n)) % n  
 counting\_list[a1] += 1  
  
for i in range(1, n):  
 counting\_list[i] += counting\_list[i - 1]  
  
for i in range(n - 1, -1, -1):  
 a1 = (math.trunc(numbers[i] / n) % n)  
 sorted\_list[counting\_list[a1] - 1] = numbers[i]  
 counting\_list[a1] -= 1  
  
numbers = sorted\_list  
print(numbers)



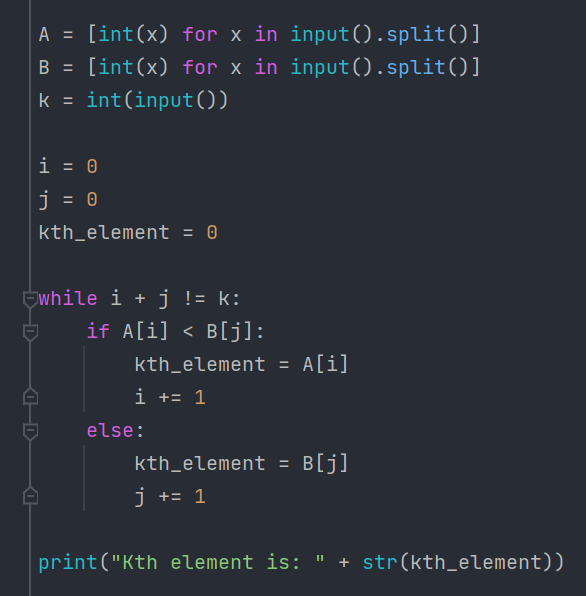
*در اندازه‌های بزرگِ*



برای پاسخ‌دهی، همزمان هر دو آرایه را می‌پیماییم. از عنصرِ اولِ هردو آرایه شروع می‌کنیم و دو عنصر را با یکدیگر مقایسه می‌کنیم. عنصرِ کوچک‌تر را رد می‌کنیم و به عنصرِ بعدیِ آن آرایه می‌رویم. آن‌قدر این روند را ادامه می‌دهیم که k عنصر را پیموده باشیم، در این هنگام به عنصرِ k امِ ترکیب مرتب‌شده‌ی دو آرایه می‌رسیم.

بدترین زمان پاسخ‌گوییِ این الگوریتم زمانی‌ست که عضو آخر از ما خواسته شده باشد، در این هنگام باید کلِ عناصر، یعنی a + b عنصر را بیپماییم و زمانمان برابر O(a + b) خواهد شد.

کدِ پایتونِ الگوریتم:

A = [int(x) for x in input().split()]

B = [int(x) for x in input().split()]

k = int(input())

i = 0

j = 0

kth\_element = 0

while i + j != k:

if A[i] < B[j]:

kth\_element = A[i]

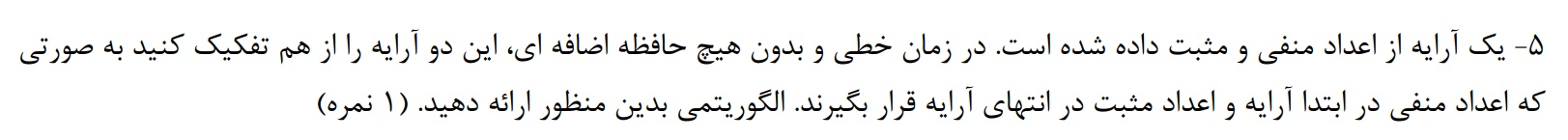
i += 1

else:

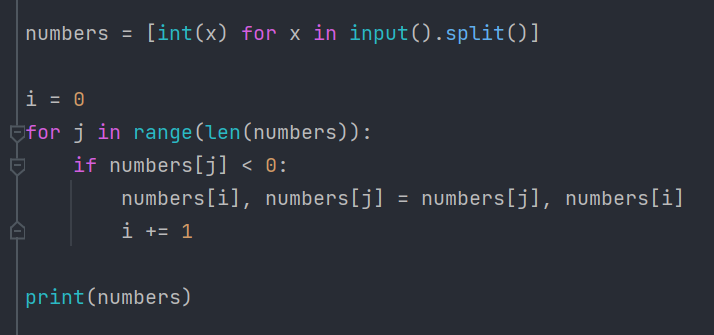
kth\_element = B[j]

j += 1

print("Kth element is: " + str(kth\_element))



کافی‌ست از عنصر اول شروع به پیمایش کنیم. اگر عنصرمان مثبت بود از آن می‌گذریم، اگر منفی بود آن را با عنصر شماره‌ی ۰ جابه‌جا می‌کنیم. عنصرِ بعدیِ منفی‌مان را با عنصر شماره‌ی ۱ جابه‌جا می‌کنیم، عنصر منفیِ بعدی را با عنصر شماره‌ی ۲ و...



numbers = [int(x) for x in input().split()]

i = 0

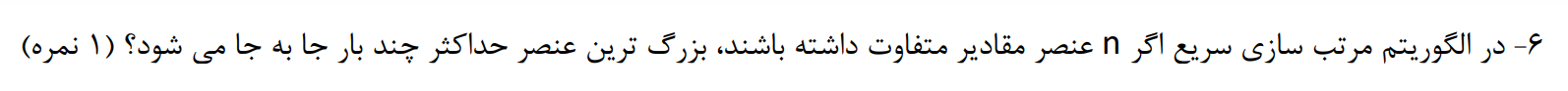
for j in range(len(numbers)):

if numbers[j] < 0:

numbers[i], numbers[j] = numbers[j], numbers[i]

i += 1

print(numbers)



می‌دانیم که هر یک از اعضا، هنگامی که به عنوان عنصر محوری انتخاب می‌شود به جایگاه نهاییِ خود خواهد رفت و دیگر جابه‌جا نخواهد شد، پس باید سعی کنیم عنصر مذکور، تا جای ممکن به عنوان عضو محوری انتخاب نشود.

