به نام خدا

تمرین شماره ۲ درس ساختمان داده‌ها و الگوریتم‌ها

چمران معینی : ۹۹۳۱۰۵۳

۱- الگوریتمی ارائه دهید که n عدد از بازه ۱ تا k میگیرد و با اعمال زمان O(n+k) ، پردازشی را انجام میدهد. سپس هرگاه از این الگوریتم پرسیده شود که چندتا از n عدد در بازه [a,b] قرار می‌گیرند، در زمان خطی پاسخ می‌دهد. (۱ نمره)

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.

۲- الگوریتمی طراحی کنید که آرایه n عضوی شامل اعدادی از بازه ۱ تا n^2 را در زمان O(n) مرتب کند. (۱ نمره)

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

می‌توانیم تمامِ اعداد را به این فرم بنویسیم:

$$a = (a_1)n + a_2$$

با توجه به این که هیچ یک از اعداد، بزرگ‌تر از n^2 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):

```python
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)
```

۳- روابط بازگشتی زیر را با روش جایگزینی حل کنید: (هر کدام ۱ نمره، در کل ۶ نمره)

a) $T(n) = T(n-1) + 2^n, T(1) = 1$

b) $T(n) = 3T\left(\frac{n}{2}\right) + n^2, T(n) = 1 \text{ for } n < 10$

c) $T(n) = T\left(\frac{3n}{7}\right) + T\left(\frac{n}{3}\right) + n, T(n) = 1 \text{ for } n < 10$

d) $T(n) = 2T\left(\frac{n}{2}\right) + n\lg^2 n, T(n) = 1 \text{ for } n < 10$

e) $T(n) = T(n-1) + n^2, T(n) = 1 \text{ for } n < 10$

f) $T(n) = 2T(n-1) + c, T(1) = 1$

a) $T(n) = T(n-1) + 2^n \to T(n) = (T(n-2) + 2^{n-1}) + 2^n \to \boldsymbol{T(n) = T(n-2) + 2^n\left(\frac{1}{2^0} + \frac{1}{2^1}\right)} \to T(n) = \left(T(n-3) + \frac{2^n}{2^2}\right) + 2^n\left(\frac{1}{2^0} + \frac{1}{2^1}\right) \to \boldsymbol{T(n) = T(n-3) + 2^n\left(\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2}\right)} \to T(n) = T\big(n-(n-1)\big) + 2^n\left(1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^{n-2}}\right) = T(1) + 2^n(2) \to$

$$T(n) = 1 + 2^{n+1}$$

b) $T(n) = 3T\left(\frac{n}{2}\right) + n^2 = 3\left(3T\left(\frac{n}{4}\right) + \frac{n^2}{4}\right) + n^2 \to \boldsymbol{T(n) = 3^2T\left(\frac{n}{4}\right) + n^2\left(1 + \frac{3}{4}\right)} \to T(n) = 3^2\left(3T\left(\frac{n}{8}\right) + \frac{n^2}{2^4}\right) + n^2\left(1 + \frac{3}{4}\right) \to T(n) = 3^3T\left(\frac{n}{2^3}\right) + n^2\left(\left(\frac{3}{4}\right)^0 + \left(\frac{3}{4}\right)^1 + \left(\frac{3}{4}\right)^2\right) = \boldsymbol{3^iT\left(\frac{n}{2^i}\right) + n^2\left(\left(\frac{3}{4}\right)^0 + \left(\frac{3}{4}\right)^1 + \left(\frac{3}{4}\right)^2 + \cdots + \left(\frac{3}{4}\right)^{i-1}\right)}$

$$\left(S = \left(\frac{3}{4}\right)^0 + \left(\frac{3}{4}\right)^1 + \left(\frac{3}{4}\right)^2 + \cdots \to \frac{3}{4}S = \left(\frac{3}{4}\right)^1 + \left(\frac{3}{4}\right)^2 + \cdots \to S - \frac{3}{4}S = \frac{1}{4}S = \left(\frac{3}{4}\right)^0 = 1 \to S = 4\right) \to$$

$$T(n) = 3^iT\left(\frac{n}{2^i}\right) + 4n^2 \to 1 \le \frac{n}{2^i} < 10$$

$$\to \frac{n}{2^i} < 10 \to \log_{10}\frac{n}{2^i} < 1 \to \log_{10}n - i\log_{10}2 < 1 \to \frac{(\log n - 1)}{\log 2} < i$$

$$\to 1 \le \frac{n}{2^i} \to 2^i \le n \to \log 2^i \le \log n \to i\log 2 \le \log n \to i \le \frac{\log n}{\log 2}$$

$$\to \frac{(\log n - 1)}{\log 2} < i \le \frac{\log n}{\log 2} \to 3^{\frac{(\log n - \log 10)}{\log 2}} < 3^i \le 3^{\frac{\log n}{\log 2}} \to 3^{\log_2\frac{n}{10}} < 3^i \le 3^{\log_2 n}$$

→ در اندازه‌های بزرگِ n →

$$3^i = 3^{\log_2 n}$$

$$T(n) = 3^iT\left(\frac{n}{2^i}\right) + 4n^2 = \left(3^{\log_2 n}\right)(1) + 4n^2 = n^{\log_2 3} + 4n^2$$

c) $T(n) = T\left(\frac{3n}{7}\right) + T\left(\frac{n}{3}\right) + n \to T(n) = \left(T\left(\left(\frac{3}{7}\right)^2 n\right) + T\left(\frac{n}{7}\right) + n\right) + \left(T\left(\frac{n}{7}\right) + T\left(\frac{n}{3^2}\right) + n\right) + n = T\left(\left(\frac{3}{7}\right)^2 n\right) + T\left(\frac{n}{7}\right) + T\left(\frac{n}{7}\right) + T\left(\frac{n}{3^2}\right)$

$$T(1) = 1, T(3) = 1$$

$$T(21) = T\left(\frac{(3)(21)}{7}\right) + T\left(\frac{21}{3}\right) + 21 = T(9) + T(7) + 2 = 1 + 1 + 2 = 4$$

d) $T(n) = 2T\left(\frac{n}{2}\right) + n\lg^2 n = 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\lg^2\frac{n}{2}\right) + n\lg^2 n = 2^2T\left(\frac{n}{2^2}\right) + \frac{n}{2}\lg^2\frac{n}{2} + n\lg^2 n =$
$2^2T\left(\frac{n}{2^2}\right) + \frac{n}{2}(\lg n - \lg 2)^2 + n\lg^2 n$

e) $T(n) = T(n-1) + n^2 = (T(n-2) + (n-1)^2) + n^2 = (T(n-3) + (n-2)^2) + n^2 +$
$(n-1)^2 = T(n-i) + n^2 + (n-1)^2 + (n-2)^2 + \cdots + (n-i+1)^2$
$\to [n - i = 1 \to i = n - 1] \to$
$$T(n) = T(1) + 2^2 + 3^2 + 4^2 + \cdots + n^2 = 1^2 + 2^2 + .. + n^2 = \frac{n(n+1)(2n+1)}{6}$$

f) $T(n) = 2T(n-1) + c = 2(2T(n-2) + c) + c = 2^2T(n-2) + 2c = 2^2(2T(n-3) + c) +$
$2c = 2^3T(n-3) + 3c = 2^iT(n-i) + ic$
$\to [n - i = 1 \to i = n - 1] \to$
$T(n) = 2^{n-1}T(1) + (n-1)c = 2^{n-1} + nc - c$

۴- آرایه A شامل a عضو و آرایه B شامل b عضو که مرتب شده‌اند را در اختیار داریم. الگوریتمی برای پیدا کردن عنصر kام در ترکیب مرتب شده این دو آرایه در زمان O(a+b) ارائه دهید. (۱ نمره)

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 بپیماییم و زمانمان برابر O(a + b) خواهد شد.

کدِ پایتونِ الگوریتم:

```python
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))
```


```python
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))
```

۵- یک آرایه از اعداد منفی و مثبت داده شده است. در زمان خطی و بدون هیچ حافظه اضافه ای، این دو آرایه را از هم تفکیک کنید به صورتی که اعداد منفی در ابتدا آرایه و اعداد مثبت در انتهای آرایه قرار بگیرند. الگوریتمی بدین منظور ارائه دهید. (۱ نمره)

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)
```


```
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)
```

۶- در الگوریتم مرتب سازی سریع اگر n عنصر مقادیر متفاوت داشته باشند، بزرگ ترین عنصر حداکثر چند بار جا به جا می شود؟ (۱ نمره)

بزرگ‌ترین عنصر را، چپ‌ترین عنصر فرض می‌کنیم، باید طوری پیش برویم که این عنصر در هر حرکت، به کم‌ترین مقدارِ ممکن به سمت راست حرکت کند.

فرض می‌کنیم که یک بازه‌ی سه تایی برای مرتب کردن داشته باشیم، اگر بخواهیم که در هر حرکت این عنصر تنها یک خانه به سمت راست برود، ناچاریم که آن را به عنوان محور در نظر بگیریم، از آن‌جا که محور در نظر گرفتنِ آن باعث می‌شود که به جایگاه نهایی خود برود، پس ناچاریم که حرکاتمان را طوری در نظر بگیریم که بزرگ‌ترین عنصر، همیشه عضو چسبیدهِ به راستِ عنصرِ محوری باشد و بعد از جابه‌جایی، به خانه‌ی چسبیده‌ی سمت راستیِ محور برود، یعنی دو خانه جابه‌جا شود. با این حساب، با حرکاتِ دوخانه‌دوخانه‌ی بزرگ‌ترین عنصر، در صورتی که تعداد عناصر زوج باشد، حداکثر به $\frac{n}{2}$ جابه‌جایی و در صورتی که عنصر فرد باشد به $\frac{n-1}{2}$ جابه‌جایی نیاز خواهیم داشت.

۷- فرض کنید آرایه ای به طول N داریم. فرض کنید در مرتب سازی سریع، میانه عناصر زیر را به عنوان محور در نظر می‌گیریم:

عنصر ابتدای آرایه(k=0)

عنصر انتهای آرایه(k=N-1)

عنصر وسط آرایه(k=N/2)

عنصر وسط قسمت راست آرایه(k=N/4)

عنصر وسط قسمت چپ آرایه(k=3N/4)

در این صورت، زمان اجرای مرتب‌سازی در بدترین حالت چه خواهد بود؟ (۱ نمره)

حتی اگر ما تمام این موارد را هم برای انتخاب محور در نظر بگیریم، به هرحال هررر عنصری با هر مقداری ممکن است در مکان‌های گفته شده قرار گرفته باشد و نهایتا بدترین حالت همان $O(n^2)$ خواهد بود.

۸- الگوریتم مرتب‌سازی زیر را برای مرتب کردن مجموعه A که شامل n عدد صحیح مثبت کوچکتر از k است را در نظر بگیرید:

```
Assignment-Sort(A, B, k)
   for i <- 0 to k
       do C[i] <- 0
   for j <- 1 to length[A]
       do C[A[j]] <- c[A[j]] + 1
   for i <- 1 to k
       do C[i] <- C[i] + C[i-1]
   for j <- 1 to length[A]
      do B[C[A[j]]] <- A[j]
      do C[A[j]] <- C[A[j]] - 1
```

الف) ثابت کنید که این الگوریتم اعداد را به درستی مرتب می‌کند. (۱ نمره)

ب) آیا این الگوریتم پایدار است؟ چرا؟ (۱ نمره)

الف)

این الگوریتم مشابه counting sort عمل می‌کند.

در حلقه‌ی اول، یک آرایه‌ی جدید تعریف می‌کنیم که تمام عناصرِ آن صفر است.

در حلقه‌ی دوم به ازای هر عنصر از A با مقدار A[i] ، یک واحد به خانه‌ی C[A[i]] اضافه می‌کنیم، به نوعی آرایه‌ی C مانندِ ظرفی‌ست که خانه‌ی i ام از آن، برای شمارشِ تعداد i های موجود در آرایه‌ی A است.

سپس در حلقه‌ی سوم از عضو یکم شروع می‌کنیم هر عضو را، با تمامِ مقادیرِ خانه‌های قبلی‌اش جمع می‌کنیم، با این کار مطمئن می‌شویم که همواره عضو n ام از آرایه‌ی C ، دارای C[n] عضوِ کوچک‌تر یا مساویِ n است و در نتیجه عضو C[n] ام از آرایه‌ی مرتب شده برابر با n است. برای مثال اگر مقدار خانه‌ی هشتم از آرایه‌ی C برابر با ۱۱ باشد، می‌دانیم که یازدهِ عضوِ کوچک‌تر مساویِ هشت در آرایه‌مان وجود دارد، یعنی یازدهمین عضو از آرایه‌ی مرتب شده‌مان هشت است.

به سراغ آخرین خلقه می‌رویم. در این بخش تفاوتی با counting sort دیده می‌شود و آن هم در این است که از آخرین عضوِ A شروع نمی‌کنیم که این مورد هم مشکلی ایجاد نمی‌کند چون می‌دانیم که همواره عضو C[A[j]] ام از آرایه‌ی مرتب شده، باید برابر با A[j] باشد. در این حلقه یکی یکی به سراغ اعضای A[j] می‌رویم، به کمک C[A[j]] خانه‌ی صحیحِ آن را می‌یابیم، آن را سرِ جای خود می‌گذاریم و یک خانه از C[A[j]] هم کم می‌کنیم چون یک خانه شمرده شده است.

ب)

می‌دانیم که counting sort یک الگوریتم پایدار است، پس کافی‌ست مرحله‌ی آخرِ این الگوریتم را، که با counting sort متفاوت است را بررسی بکنیم.

با توجه به این که «همواره عضو n ام از آرایه‌ی C ، دارای C[n] عضوِ کوچک‌تر یا مساویِ n است»، در مرحله‌ی آخرِ الگوریتم، هر عنصر در آخرین خانه‌ی مجازِ خالی قرار می‌گیرد. پس اگر از انتهای آرایه‌ی اول شروع به جایگذاری کرده باشیم الگوریتم‌مان پایدار خواهد بود، اما در این‌جا دقیقا برعکس است، یعنی ترتیبِ تمامِ عناصری که کلیدهای یکسان دارند دقیقا برعکس خواهد شد.