به نام خداوند بخشنده مهربان

پاسخنامه پیشنهادی تمرین شماره ۲ درس ساختمان داده‌ها و الگوریتم‌ها

گروه تدریسیاری دکتر شهرضا پاییز ۱۴۰۰

سوال ۱

با استفاده از الگوریتمی که در پایین مشاهده می‌کنید آرایه ای به نام c در زمان O(n+k) مقداردهی می‌شود.

با استفاده از آرایه c مشخص می‌شود که چه تعداد عدد کوچکتر از a و b وجود دارد و با کم کردن این دو مقدار از هم، در زمان ثابت به هر پرسش جواب خواهیم داد.

(این الگوریتم در واقع بخشی از الگوریتم مرتب سازی شمارشی است.)

```
CountSmaller(A)
  //A[]-- Initial Array
  //Complexity: O(k)
  for i = 0 to k do
      c[i] = 0

  //Storing Count of each element
  //Complexity: O(n)
  for j = 0 to n do
      c[A[j]] = c[A[j]] + 1

  //Complexity: O(k)
  for i = 1 to k do
      c[i] = c[i] + c[i-1]
  return c
end func
```

سوال ۲

با استفاده از الگوریتم مرتب سازی مبنایی(Radix Sort)

۱. ابتدا یک واحد از تمامی عناصرآرایه کم می‌کنیم تا به بازه تعریف شده در Radix sort برسیم.
۲. Radix sort را اعمال کرده و اعداد را مرتب می‌کنیم.
۳. یک واحد به اعداد مرتب شده اضافه می‌کنیم.

چگونه Radix Sort می‌تواند در زمان خطی این آرایه را مرتب‌کند؟ برای مطالعه به لینک زیر مراجعه کنید.

https://www.geeksforgeeks.org/sort-n-numbers-range-0-n2-1-linear-time/

سوال ۳

a) $T(n) = T(n-1) + 2^n$ ، $T(1) = 1$

حدس → $T(n) \in O(2^n)$ → پایه استقرا = $T(1) \le 2c \to c \ge \frac{1}{2}$

فرض استقرا = $T(k) \le c2^k$

گام استقرا → $T(k+1) \le 2^{k+1} \to T(k) + 2^{k+1} \le c2^k + 2^{k+1}$

$\to (2^{k+1})(1+\frac{c}{2}) \le 2^{k+1} \to c \ge 2$

، با توجه به این که c هر عددی می‌تواند باشد $\Longleftarrow$ $T(n) \in O(2^n)$

b) $T(n) = 3T(\frac{n}{2}) + n^2$ ، $T(n) = 1$ if $n < 1$.

حدس با توجه به قضیه اصلی → $T(n) \in O(n^2)$ → پایه استقرا = $1 \le c$ [redacted]

گام استقرا → $\forall m,k \mid 1 \le m \le k \to T(m) \le cm^2 \to T(k) \le ck^2$ and $c \ge 3$

$\to T(k) = 3T(\frac{k}{2}) + k^2 \Longrightarrow T(k) = 4k^2 \le 4k^2$

$\Longrightarrow T(k) \le ck^2 \longrightarrow T(n) \in O(n^2)$

c) $T(n) = T(\frac{3n}{7}) + T(\frac{n}{3}) + n$ → حدس → $T(n) \in O(n)$

پایه استقرا → $c \ge 1$ ، فرض استقرا → $\forall m,k \mid c \le 1 \le m \le k \to T(m) \le cm \to T(k) \le c$

گام → $T(k) = T(\frac{3k}{7}) + T(\frac{k}{3}) + k \le \frac{101}{21}k \le ck \to \frac{101}{21} \le c$

با توجه به این که c می‌تواند هر عددی باشد $\Longleftarrow$ $T(n) \in O(n)$

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

قابل تبدیل $\longrightarrow T(n) = 2T\left(\frac{n}{2}\right) + O(n^3)$

حدس $\longrightarrow n^3$

پایه استقرا $\rightarrow C \geq 1$ ، فرض استقرا $\rightarrow \forall m, k \quad 1 \leq m \leq k$

$\longrightarrow T(k) \leq ck$

$$T(k) = 2T\left(\frac{k}{2}\right) + k\log^2 k$$

$$\rightarrow \frac{ck^3}{4} + k\log^2 k \leq ck^3$$

$$\rightarrow \frac{3}{4}ck^2 \geq k^2$$

$$\rightarrow c \geq \frac{4}{3}$$

پس دلیلی که c می‌تواند هر عددی باشد

$$\Rightarrow T(k) \leq ck^3$$

$$\Rightarrow T(n) \in O(n^3)$$

e) $T(n) = T(n-1) + n^{۲} \longrightarrow$ حدس $\rightarrow O(n^{۳})$

پایه $\rightarrow c \geqslant ۱$

فرض $\rightarrow \forall m \rightarrow ۱ \leqslant m \leqslant k \rightarrow T(m) \leqslant cm^{۳} \rightarrow T(k) \leqslant ck^{۳}$

گام $\rightarrow T_{(k-1)} + k^{۲} \leqslant c(k-1)^{۳} + k^{۲}$

$\xrightarrow{c=۳} ۳n^{۳} - ۳ - ۸n^{۲} + ۹n \leqslant ۳n^{۳}$

می دانیم که c می تواند هر عددی باشد.

$\Longrightarrow T(n) \in O(n^{۳})$

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

با کمی گسترش مسئله از روش درخت یا استقرا حدس می‌زنیم که $T(n)$ از $O(2^n)$ باشد. حال باید یک حد بالا برای آن حدس بزنیم که با توجه به حدس قبلی و این که $f(n)$ یک مقدار ثابت می‌باشد حدس می‌زنیم که $T(n) \leqslant k2^n - b$

حال دو مقدار ثابت k و b داریم که باید شرایط آن دو را پیدا کنیم.

$$\text{Base case} \rightarrow n=1 \rightarrow T(1) = 1 \leqslant k \times 2^1 - b = 2k - b$$

$$\Longrightarrow k \geqslant \frac{b+1}{2}$$

حال فرض می‌کنیم برای $n-1$ درست است. باید درستی را برای n ثابت کنیم.

$$T(n) = 2T(n-1) + c_1 \leqslant 2(k2^{n-1} - b) + c_1$$

$$= \underbrace{k2^n - 2b + c_1 \leqslant k2^n - b}_{} \rightarrow b \geqslant c_1$$

پس در شرط $b \geqslant c_1$ و $k \geqslant \frac{b+1}{2}$ داریم که با توجه به قید وجود دارد در اثبات ادر O بودن می‌دانیم هر مقداری برای این دو مقدار ثابت قابل انتخاب است. پس ثابت کردیم $\rightarrow T(n) \in O(2^n)$

سوال ۴

ایده الگوریتم به این صورت است که با مقایسه عنصر k/2 هر آرایه بتوانیم بخشی از عناصر یکی از آرایه‌ها را دور بریزیم و به دنبال بازه محدودتری برای پیدا کردن جواب بگردیم.

به طور مشخص‌تر، با مقایسه عنصر k/2 در هر آرایه دو حالت به وجود می‌آید:

۱. عنصر k/2 آرایه A، از عنصر k/2 آرایه B بزرگتر باشد:

در این صورت چون هر دو آرایه مرتب شده هستند، عناصر اول تا k/2 آرایه B نیز از عنصر k/2 آرایه A کوچکتر هستند و جواب ما در آن قسمت نیست.

۲. عنصر k/2 آرایه A، کوچکترمساوی عنصر k/2 در آرایه B باشد:

در این صورت چون هر دو آرایه مرتب شده هستند، عناصر اول تا k/2 آرایه A نیز از عنصر k/2 آرایه B کوچکتر هستند و جواب ما در آن قسمت نیست.

پس در هر مرحله k/2 از اعضای یکی از آرایه‌ها دور ریخته می‌شود تا به حالت پایه برسیم وجواب پیدا شود.

پس زمان اجرا الگوریتم به صورت زیر می‌شود:

$$T(n) = O(\log(k))$$

$$k \ is \ lower \ than \ a + b \ \rightarrow T(n) = O(\log(a + b))$$

کد الگوریتم بیان‌شده به صورت زیر است:

```
def kthElement(k, A, a, B, b):
    if k > a + b:
        return -1

    if a == 0:
        return B[k-1]
    if b == 0:
        return A[k-1]
    if k == 1:
        return min(A[0], B[0])

    i = min(k//2, a)
    j = min(k//2, b)

    if A[i-1] > B[j-1]:
        return kthElement(k-j, A, a, B[j:b], b-j)
    else:
        return kthElement(k-i, A[i:a], a-i, B, b)
```

سوال ۵

ایده حل استفاده از الگوریتم افراز کردن درجا (In-place Partition) در مرتب سازی سریع است.

بدین صورت که اگر عدد صفر به عنوان عنصر محوری به این الگوریتم داده شود، اعدداد مثبت در زیر آرایه سمت راست و اعداد منفی در زیرآرایه سمت چپ قرار می‌گیرند و اعداد مثبت و منفی از یکدیگر تفکیک می‌شوند.

سودوکد افراز درجا را در اسلاید استاد مطالعه کنید.

سوال ۶

پاسخ $N-1$ است.

از آنجایی که بزرگترین عنصر همواره به سمت راست منتقل می شود، پس حداکثر به تعداد $N-1$ بار میتواند جابجا شود.

اگر $A[1]$ بزرگترین عنصر و $A[n]$ دومین عنصر بزرگ آرایه باشد، بزرگترین عنصر در هر مرحله جا به جا خواهد شد و در نهایت با عنصر محور $swap$ می شود.

مثلا:

4 1 2 3	
1 4 2 3	1 swap
1 2 4 3	1 swap
1 2 3 4	1 swap

سوال ۷

برای اینکه بدترین حالت را بدست آوریم، کافی است فرض کنیم در هر مرحله ۵ عنصر بزرگ آرایه در این خانه ها قرار دارند. در اینصورت اگر در هر مرحله میانه این عناصر را به عنوان محور انتخاب کنیم، آرایه به دو زیر آرایه به طول 2 و $n-2$ تقسیم می شود. اگر درخت بازگشتی آن را رسم کنیم، معادل رابطه زیر خواهد بود:

$$T(n) = T(n-2) + T(2) + n$$

که در آن $T(n-2)$ برای زیر آرایه به طول $n-2$ و $T(2)$ برای زیر آرایه به طول 2 و n هزینه عمل $partition$ است.

با حل این رابطه بازگشتی در می یابیم که در بدترین حالت این روند $O(n^2)$ زمان خواهد برد.

سوال ۸

الف) اگر دقت کنید متوجه میشوید که این الگوریتم در واقع همان الگوریتم مرتب سازی شمارشی می‌باشد که در خط ۸ به جای 1 to Length(A) ، Length(A) to 1 قرار داده شده است. همچنین میدانیم که ترتیب در پیمایش آرایه A و جایگذاری درست عناصر آن اهمیتی ندارد بنابرین همانطور که مرتب سازی شمارشی به درستی عمل می‌کند، این الگوریتم نیز به درستی آرایه را مرتب خواهدکرد.

ب) با توجه به اینکه ترتیب پیمایش A برعکس شده است، و ما بر اساس تعداد تکرار یک key موقعیت آنرا پیدا می‌کنیم، برعکس شدن پیمایش در این الگوریتم نسبت به مرتب سازی شمارشی باعث می‌شود که برخلاف مرتب سازی شمارشی، این الگوریتم پایدار نباشد.

سوال ۹(تمرین عملی)

```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    cout.tie(nullptr);
    int test;
    cin >> test;
    while (test--){
        int n;
        cin >> n;
        map<int, int> m;
        for (int i = 0; i < n; i++){
            int temp;
            cin >> temp;
            m[temp]++;
        }
        int total = n;
        for (auto itr = m.begin(); itr != m.end(); itr++){
            total = total + itr->second *(itr->second - 1) / 2;
        }
        cout << total << "\n";
    }
    return 0;
}
```