Date: ..... Subject: .....

پاسخ تمرین چهارم

① راه‌حل اول: یک آرایه‌ی کمکی به نام visited در نظر می‌گیریم. به هر عنصر از آرایه‌ی اصلی که رسیدیم، بررسی می‌کنیم که آیا در visited هست یا نیست. اگر در visited بود، بدون انجام کار خاصی، به عنصر بعدی می‌رویم و اگر در visited نبود، آن را وارد visited می‌کنیم. بعد از اتمام پیمایش آرایه‌ی اصلی، طول visited، جواب سؤال است.

```
count_distincts(arr):
    visited = []
    for element in arr:                          → O(n)
        if element in visited:                   → O(logn) یا O(n)
            continue                             بسته به الگوریتم مورد استفاده
        else:
            visited.append(element)              → O(1)
    return len(visited)                          → O(n)
```

اگر برای بررسی این که عنصر در visited هست یا نیست، از سبت وجود خطی استفاده کنیم، هزینه‌ی سبت وجود $O(n)$ و هزینه‌ی کل الگوریتم، $O(n^2)$ می‌شود. اگر از سبت وجود دودویی استفاده کنیم، هزینه‌ی سبت وجود $O(\log n)$ و هزینه‌ی کل الگوریتم $O(n \log n)$ خواهد شد.

راه‌حل دوم: ابتدا آرایه را مرتب می‌کنیم. سپس به هر عنصر از آرایه که رسیدیم، آن را با عنصر قبلی‌اش مقایسه می‌کنیم. اگر با عنصر قبلی‌اش برابر نبود، تعداد عناصر متفاوت را یک واحد زیاد می‌کنیم. بعد از اتمام پیمایش آرایه، پاسخ را داریم.

Date: ..... Subject: .....

```
count_distincts (arr):
    distinct_counts = 0
    arr.sort()                          ---> O(nlogn)
    for i=0 to i=n-1:                   ---> O(n)
        if i == 0:
            distinct_counts ++
        else:
            if arr[i] != arr[i-1]:
                distinct_counts ++
    return distinct_counts
```

هزینه این الگوریتم برابر است با: $O(n\log n)+O(n) = O(n\log n)$

Date: ..... Subject: .....

(2) ابتدا آرایه را مرتب می‌کنیم ← هزینه این کار $O(n\log n)$ است. حال یک پیمایش روی آرایه انجام می‌دهیم و در این پیمایش تفاضل مثبت هر عنصر با عنصر بعدی‌اش را به دست می‌آوریم، اگر برای عنصری، تفاضل مثبت آن با عنصر بعدی‌اش، بزرگ‌تر از 1 بود، کار می‌کنیم که در صورت سؤال خواسته شده امکان‌پذیر نیست (چون با فرض این که آرایه را به شکل نانزولی مرتب کرده باشیم، اگر تفاضل مثبت یک عنصر با عنصر بعدی‌اش، بیشتر از 1 بود، باید عناصر بیشتر از 1 الت، پس برای این عنصر زوجی یافت نمی‌شود و در پایان عملیات حداقل 2 عنصر باقی می‌ماند) و False برمی‌گردانیم در غیر این صورت چیزی که در صورت سؤال خواسته شده امکان‌پذیر است و کافیست برای رسیدن به یک عنصر، از ابتدای آرایه مرتب شده شروع کنیم، هر عنصر را با عنصر بعدی‌اش بگیریم و همین آرایه نانزولی الت، خود عنصر حذف می‌شود. این کار را تا رسیدن به آخرین عنصر ادامه می‌دهیم و در آخر بزرگ‌ترین عنصر باقی می‌ماند. هزینه انجام این کار هم $O(n)$ است. هزینه پیمایش برای بررسی امکان‌پذیر بودن یا نبودن خواسته سؤال هم $O(n)$ است ← هزینه کل الگوریتم برابر است با:

$$O(n\log n) + O(n) + O(n) = O(n\log n)$$

```
fun (arr):
    arr.sort()  ——→ O(nlogn)
    n = len(arr)
    for i in range(n-1):  ——→ O(n)
        if arr[i+1] - arr[i] > 1:
            return False
    return True
```

Date: ..... Subject: .....

```
convert_to_single (arr):
    while len(arr) != 1 ——> O(n)
        arr.pop(0)

if fun(arr):
    convert_to_single(arr)
else:
    print('IMPOSSIBLE')
```

Date: ..... Subject: .....

(3)

$$\left[\begin{array}{ccc|c} 7 & 5 & -3 & 16 \\ 3 & -5 & 2 & -8 \\ 5 & 3 & -7 & 0 \end{array}\right] \quad \begin{array}{l} 7\times row_2 - 3\times row_1 \\ 7\times row_3 - 5\,row_1 \end{array}$$

$$\left[\begin{array}{ccc|c} 7 & 5 & -3 & 16 \\ 0 & -50 & 23 & -104 \\ 0 & -4 & -34 & -80 \end{array}\right] \quad 25\times row_3 - 2\,row_2$$

$$\left[\begin{array}{ccc|c} 7 & 5 & -3 & 16 \\ 0 & -50 & 23 & -104 \\ 0 & 0 & -896 & -1792 \end{array}\right] \Rightarrow \begin{cases} 7x + 5y - 3z = 16 \\ -50y + 23z = -104 \\ -896z = -1792 \end{cases}$$

$\Rightarrow \boxed{z = 2} \Rightarrow -50y + 46 = -104 \Rightarrow -50y = -150$

$\Rightarrow \boxed{y = 3}$

$7x + 15 - 6 = 16 \Rightarrow 7x = 7 \Rightarrow \boxed{x = 1}$

4) به طور کلی برای مشخص کردن این که یک نقطه دلخواه، داخل یک خم بسته هست یا خیر، می‌توانیم از winding number استفاده نماییم. اگر winding number یک منحنی حول یک نقطه دلخواه، 0 شود، آن نقطه، خارج از منحنی خواهد بود. در غیر این صورت نقطه مذکور، داخل منحنی است.

اگر یک خم بسته به شکل $\gamma=(\gamma_x,\gamma_y):[0,1]\to\mathbb{R}^2$ داشته باشیم، به طوری که $\gamma(0)=\gamma(1)$، آن گاه برای یک نقطه دلخواه (که روی منحنی نیست) مانند (x_0,y_0)، winding number از رابطه زیر به دست می‌آید:

$$\omega_\gamma(x_0,y_0)=\frac{1}{2\pi}\oint_\gamma d\theta=\frac{1}{2\pi}\oint_\gamma \frac{(\gamma_x(t)-x_0)\gamma_y'(t)-(\gamma_y(t)-y_0)\gamma_x'(t)}{(\gamma_x(t)-x_0)^2+(\gamma_y(t)-y_0)^2}\,dt$$

* اگر نقطه مذکور، به شکل پادساعتگرد توسط منحنی محصور شده باشد، winding number مثبت می‌شود و اگر به شکل ساعتگرد محصور شده باشد، winding number منفی خواهد شد.

در رابطه با این که این سؤال چه ارتباطی با مسائل دارای transform and conquer دارد، باید گفت که یک مسئله هندسی تبدیل به یک مسئله جبری شده است. پس می‌توان آن را جزو مسائل transform and conquer دانست.

Date: ..... Subject: .....

(5) الگوریتم هورنر برای محاسبه مقدار یک چند جمله‌ای در نقطه x به شکل زیر است:

```
p ← P[n]
for i ← n-1 downto 0 do
    p ← x * p + P[i]
return p
```

حال در چند جمله‌ای صورت سوال که درجه فرد است، توان هر جمله با جمله دیگر 2 واحد فرق می‌کند پس در حلقه for به جای این که در x ضرب کنیم، باید در x^2 ضرب نماییم. نقطه باقیمانده‌اش این باشد که وقتی به آخرین ضریب رسیدیم در x ضرب نماییم، چرا که در گام اول از x فاکتور گرفتیم و در باقی گام‌ها از x^2:

$$P_{2n+1}(x) = x\left(a_{2n+1}x^{2n} + a_{2n-1}x^{2n-2} + a_{2n-3}x^{2n-4} + \cdots + a_3x^2 + a_1\right) + 0 \rightarrow P[0]$$

گام اول فاکتور x

$$\Rightarrow P_{2n+1}(x) = x\left(x^2\left(a_{2n+1}x^{2n-2} + a_{2n-1}x^{2n-4} + a_{2n-3}x^{2n-6} + \cdots + a_3\right) + a_1\right) + 0$$

گام دوم فاکتور x^2 ... گام آخر

Date: ..... Subject: .....

=> الگوریتم هرنر به شکل زیر تغییر می کند:

لیستی حاوی ضرایب چندجمله‌ای ← P

$(P_0 = 0)$

```
n ← len(P)
p ← P[n]
for i ← n-1 downto 0 do
    if i != 0
        p ← x^2 * p + P[i]
    else
        p ← x * p + P[i]
```

* دقت شود که در صورت سوال، $P_0 = 0$ است و به معنی علامت نوشته شده $P_0 \geq 0$ و اگر $P_0 \neq 0$ باشد، نیازی به نوشتن $P_0 = 0$ نیست.

Date: ..... Subject: .....

(6) اگر فرض کنیم که اندیس های یک آرایه از 1 شروع می شوند، آنگاه اگر اندیس یک عنصر، i باشد، اندیس پدر آن $\lfloor \frac{i}{2} \rfloor$، اندیس فرزند چپ آن، $2*i$ و اندیس فرزند راست آن، $2*i+1$ خواهد بود. اگر آرایه‌ای شرایط بالا را داشته باشد و هر عنصر پدر، از فرزندانش بزرگ‌تر باشد، آرایه یک هرم بیشینه است و اگر هر عنصر پدر از فرزندانش کوچک‌تر باشد، آرایه، هرم کمینه است. در این جا راه‌حل بررسی هرم بیشینه بودن را ارائه می‌دهیم. برای بررسی هرم کمینه بودن، روش همین است. فقط باید چک کرد که پدر، کوچک‌تر از فرزندانش باشد.

راه‌حل: راه‌حل این است که به ازای تمام عناصر پدر بررسی نماییم که پدر از فرزندانش بزرگ‌تر است یا خیر.

```
is_heap(arr):
    n = len(arr)
    end_ind = n//2
    for i in range(1, end_ind+1):          ← بار ⌊n/2⌋
        if arr[2*i-1] > arr[i-1]:
            return False
        if 2*i < n and arr[2*i] > arr[i-1]:
            return False
    return True
```

2 مقایسه

از آن جایی که حلقه for، $\lfloor \frac{n}{2} \rfloor$ بار اجرا می‌شود و در هر بار اجرای آن دو مقایسه برای فرزندان چپ و راست داریم، پس راه‌حل از $O(n)$ است.

از لحاظ حافظه هم گفتیم حافظه‌ای اشغال نمی‌کند. پس از $O(1)$ است.

Date: ..... Subject: .....

(7) محدودیت مسأله در میزان تولید روزانه هر محصول است که آن هم به وسیله ساعت کاری هر واحد محدود شده است. آن چیزهم با بیشتر بیشینه باشد، سود حاصل از فروش است. اگر فرض کنیم که پیراهن را با S، دامن را با L و شلوار را با P نشان می دهیم، آن چیز باید بیشینه شود به شکل زیر است و محدودیت هایی که در ادامه اشاره نوشته شده اند را دارد:

maximize $6S + 4L + 8P$ ⟶ سود به دلار

such that: $0.03S + 0.02L + 0.02P \leq 12$

$0.01S + 0.02L + 0.03P \leq 14$

$0.02S + 0.03L + 0.04P \leq 16$

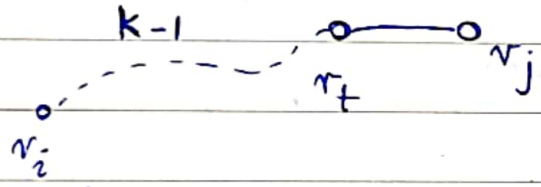
$S, L, P \in W$

Date: ..... Subject: .....

(8) مسأله را با استقرا روی k حل می‌کنیم:

پایهٔ استقرا ← $k=1$ به وضوح برقرار است، چرا که $k=1$ نیز مسیری به طول ۱ است. مسیر به طول ۱ بین دو رأس، معادل این است که آن دو رأس همسایه باشند. و این موضوع به وضوح در ماتریس مجاورت گراف برقرار است چرا که اگر دو رأس مجاور باشند، درایهٔ نظیر آن دو، ۱ می‌شود.

حال می‌توان گفت هر مسیر به طول k از رأس v_i به رأس v_j، از یک مسیر به طول $k-1$ از رأس v_i به یک رأس میانی مانند t و یک یال از رأس t به رأس v_j تشکیل شده است:



از طرفی طبق فرض استقرا تعداد مسیرهای به طول $k-1$ از رأس v_i به رأس t برابر است با درایهٔ (i,t) از ماتریس A^{k-1}. حال اگر فرض کنیم درایهٔ (i,t) از ماتریس A^{k-1}، برابر با b_{it} باشد، چون رأس میانی t می‌تواند هریک از رئوس 1 تا n باشد، طبق اصل ضرب تعداد مسیرهای به طول k که از رأس t می‌گذرند، برابر است با:

$$\sum_{t=1}^{n} b_{it}\, a_{tj}$$

(a_{tj}، همان درایهٔ (t,j) از ماتریس A است.)

$\sum$ فوق همان تعریف درایهٔ (i,j) در ماتریس A^k است ($A^k = A^{k-1} \times A$)

$\Rightarrow A^k$ مسیرهای به طول k بین هر دو رأس دلخواه گراف را نشان می‌دهد

و حکم استقرا اثبات شد.