Week 01 Questions

سوالات تکالیف هفته ۰۱

سوال اول (۱۰امتیاز):

عملیات جستجوی درِ گمشده در انبار مخفی 👁️🚪 !

سالی و **مایک وازوفسکی** در انبار عظیم کارخانه هیولاها گم شده‌اند. آن‌ها باید درِ اتاق **بو** را پیدا کنند، اما تمام درها به هم ریخته‌اند! روی هر در یک **کد ارتفاع** حک شده است. مایک متوجه شده که درِ اتاق بو، بلندترین در موجود است و برای رسیدن به آن، باید تمام درها را بر اساس ارتفاع مرتب کنند.

راندال (رقیب سالی) برای خرابکاری، سیستم جابجایی خودکار را غیرفعال کرده است. حالا سالی مجبور است از **الگوریتم مرتب‌سازی حبابی (Bubble Sort)** استفاده کند. او در هر مرحله دو درِ کنار هم را با هم مقایسه می‌کند و اگر درِ سمت چپی بلندتر بود، جای آن‌ها را با هم عوض می‌کند تا بلندترین درها مثل **حباب** به انتهای صف بروند.

وظیفه شما :به مایک و سالی کمک کنید تا تابعی به نام monster_door_sortبنویسند که:

۱. لیستی از ارتفاع درها را از ورودی بگیرد.
۲. با استفاده از الگوریتم **Bubble Sort**، آن‌ها را از کوچک به بزرگ مرتب کند.
۳. **بهینه‌سازی :**اگر در یک دور کامل(Pass) ، هیچ جابجایی صورت نگرفت، الگوریتم باید متوقف شود (یعنی لیست مرتب شده است).

ورودی نمونه:

درها: 120 85 150 70 110

خروجی مورد انتظار:

درهای مرتب شده : [70, 85, 110, 120, 150]

سوال دوم (۳۰امتیاز):

رمزگشایی ژورنال گمشده و طلسم بیل سایفر ⚠️ !

دیپر و **میبل** در زیرزمین معمای کلبه (Mystery Shack) یک صندوقچه قدیمی پیدا کرده‌اند که متعلق به نویسنده ژورنال‌هاست. داخل صندوق، صفحات پاره‌شده‌ای از یک فصل گمشده وجود دارد. روی هر صفحه یک **عدد جادویی** نوشته شده است.

ناگهان **بیل سایفر** ظاهر می‌شود و می‌گوید: "این صفحات حاوی طلسمی هستند که اگر به ترتیب **صعودی دقیق** (از کمترین قدرت به بیشترین) خوانده نشوند، کل شهر **گراویتی فالز** به بُعد کابوس‌ها منتقل می‌شود"!

دیپر که همیشه محتاط و دقیق است، پیشنهاد می‌دهد" :ما نباید ریسک کنیم! در روش‌های قبلی (مثل بابل سورت) مدام اعداد را جابجا می‌کردیم، اما اینجا هر جابجایی اشتباه ممکن است باعث انفجار جادویی شود. ما باید از استراتژی **مرتب‌سازی انتخابی (Selection Sort)** استفاده کنیم".

راهکار دیپر:

۱. کل لیست صفحات به‌هم‌ریخته را نگاه کن.

۲. با دقت فراوان **کوچکترین عدد** (کمترین قدرت جادویی) را در کل لیست پیدا کن.

۳. آن را با اولین صفحه لیست جابجا کن (تا جایگاهش تثبیت شود).

۴. حالا سراغ بقیه صفحات برو (از دومی تا آخر)، دوباره کوچکترینِ باقی‌مانده را پیدا کن و با صفحه دوم جابجا کن.

۵. این کار را تکرار کن تا تمام صفحات مرتب شوند.

وظیفه شما :به دیپر کمک کنید تا تابعی به نام gravity_falls_selectionبنویسد که:

۱. لیستی از اعداد جادویی (اعداد صحیح) را دریافت کند.

۲. با استفاده از الگوریتم **Selection Sort**آن‌ها را مرتب کند.

۳. برای اطمینان از عملکرد، در هر مرحله که یک جابجایی (Swap) انجام می‌شود، وضعیت لیست را چاپ کند تا میبل بتواند فرایند را چک کند.

ورودی نمونه:

11 22 12 25 64

خروجی مورد انتظار:

[11, 12, 22, 25, 64]

سوال سوم(۵۰امتیاز):

فرار بزرگ از باغ‌وحش مرکزی نیویورک 🦁🦒

الکس (شیر)، مارتی (گورخر)، ملمن (زرافه) و گلوریا (اسب آبی) بالاخره تصمیم گرفته‌اند از باغ‌وحش فرار کنند. اما «اسکیپر»، رهبر پنگوئن‌ها، اعلام کرده که برای یک فرار بی‌نقص، باید تمام حیوانات بر اساس »کد اولویت خروج «مرتب شوند.

اسکیپر می‌گوید: «بچه‌ها، ما وقت نداریم تک‌تک حیوانات رو با هم مقایسه کنیم! چون کدهای اولویت همگی اعداد صحیح کوچکی هستند، باید از روشی استفاده کنیم که با شمارش تعداد تکرار هر اولویت، سریعاً صف رو مرتب کنه«.

پایداری (Stability) : اسکیپر تأکید دارد حیواناتی که اولویت یکسانی دارند، باید به همان ترتیبی که در صف اولیه بودند باقی بمانند (نظم صف به هم نخورد).

اعداد منفی :طبق محاسبات پنگوئن‌ها، برخی از حیوانات که از بخش‌های فوق‌سری یا زیرزمین باغ‌وحش می‌آیند، کد اولویت منفی دارند! همان‌طور که می‌دانید، Counting Sortبه صورت استاندارد با ایندکس‌های آرایه کار می‌کند و اعداد منفی را نمی‌پذیرد.

وظیفه شما:

به پنگوئن‌ها کمک کنید تا تابعی به نام madagascar_escape_sortدر پایتون بنویسند که:

۱. لیستی از کدهای اولویت (شامل اعداد مثبت و منفی) را دریافت کند.

۲. با استفاده از الگوریتم Counting Sort آن‌ها را مرتب کند.

۳. برای حل مشکل اعداد منفی، ابتدا کمترین مقدار (min_val) را پیدا کرده و از آن به عنوان Offsetاستفاده کند تا تمام اعداد موقتاً مثبت شوند و پس از مرتب‌سازی به حالت اول بازگردند.

راهنمایی:

یک آرایه برای شمارش (Count Array) به طول max - min + 1 بسازید.

برای حفظ پایداری، حتماً از آرایه کمکی برای محاسبه مجموع تجمعی (Cumulative Sum) استفاده کنید و پیمایش لیست اصلی را از **انتها به ابتدا** انجام دهید.

ورودی نمونه:

priorities = [4, -2, 0, 4, -1, -2, 1, 3]

خروجی مورد انتظار:

Sorted Priorities: [-2, -2, -1, 0, 1, 3, 4, 4]

سوال چهارم(۷۰ امتیاز):

پروتکل جعبه سیاه بانکی 💸 🏦

شما به عنوان مهندس ارشد به تیم امنیت بانک مرکزی پیوسته‌اید. سیستم قدیمی تشخیص کلاهبرداری (Anti-Fraud) که سال‌ها پیش توسط یک برنامه‌نویس نابغه (که اکنون ناپدید شده) نوشته شده بود، دچار اختلال شده است.
این سیستم تراکنش‌های مشکوک را دریافت می‌کرد و آن‌ها را برای بررسی توسط کارشناسان انسانی اولویت‌بندی (مرتب‌سازی) می‌کرد. متاسفانه سورس‌کد اصلی پاک شده و تنها چیزی که باقی مانده، یک مستند فنی ناقص (سوخته) و لاگ‌های سرور است.

وظیفه شما بازنویسی تابع fraud_detection_sort است که دقیقاً همان رفتار سیستم قبلی را با استفاده از Bucket Sort شبیه‌سازی کند.

🧩 مستندات فنی باقی‌مانده (بخش‌های خوانا)

سیستم توزیع بار (Load Balancer):

تراکنش‌ها باید در ۱۰ سطل (Bucket) شماره‌های ۰ تا ۹ دسته‌بندی شوند. معیار انتخاب سطل وابسته به "مجموع ارقام (Digit Sum) "شناسه تراکنش است:

1. **اگر مجموع ارقام تراکنش زوج باشد: تراکنش بر اساس ..... (ناخوانا: مربوط به ابتدای عدد)به سطل هدایت می‌شود.**
2. **اگر مجموع ارقام تراکنش فرد باشد: تراکنش بر اساس ..... (ناخوانا: مربوط به انتهای عدد)به سطل هدایت می‌شود.**

پروتکل اولویت‌بندی:(Sorting Protocol)

- **سطل‌های کم‌خطر (۰ تا ۴) باید به صورت صعودی مرتب شوند تا تراکنش‌های کوچک‌تر زودتر بررسی شوند.**
- **سطل‌های پرخطر (۵ تا ۹) باید به صورت) .....ناخوانا (مرتب شوند (وضعیت اضطراری).**

فرمت داده‌ها:

- **شناسه تراکنش‌ها رشته‌های عددی هستند.**
- **هشدار :شناسه 007با شناسه 7تفاوت دارد و هویت آن باید حفظ شود.**

🕵️‍♀️ داده‌های لاگ سرور (تنها منبع حقیقت)

شما باید با تحلیل ورودی و خروجی زیر، جاهای خالی مستندات را پر کرده و الگوریتم را کشف کنید:

ورودی نمونه:

912 040 312 800 005 521 119 021 700

خروجی نمونه:

040 700 021 312 521 005 800 912 119

سوال پنجم (۱۰۰ امتیاز):

طراحی سامانه مرتب‌سازی هوشمند تراکنش‌های کلان

در سامانه‌های بانکی، مرتب‌سازی شناسه‌های تراکنش (Transaction IDs) به دلیل حجم بسیار بالا و تعداد ارقام زیاد، با روش‌های سنتی) مانند (Quick Sort با چالش کندی مواجه می‌شود. شما ماموریت دارید یک موتور مرتب‌سازی مبتنی بر **Radix Sort (LSD)** طراحی کنید که نه تنها اعداد را مرتب می‌کند، بلکه فرآیند انتخاب مبنا را به صورت هوشمند مدیریت نماید.

الف) نیازمندی‌های پیاده‌سازی (الزامی):

۱ .انتخاب هوشمند مبنا :(Adaptive Base) الگوریتم شما باید بر اساس تعداد ارقامِ بزرگترین عدد ورودی، مبنای مرتب‌سازی را طبق جدول زیر انتخاب کند:

- اگر ارقام $4 \leq$:مبنای ۱۰ (بررسی ۱ رقم در هر مرحله)
- اگر ارقام بین ۵ تا ۸: مبنای ۱۰۰ (بررسی ۲ رقم در هر مرحله)
- اگر ارقام بین ۹ تا ۱۲: مبنای ۱۰۰۰ (بررسی ۳ رقم در هر مرحله)
- اگر ارقام $12 <$:مبنای ۱۰,۰۰۰ (بررسی ۴ رقم در هر مرحله)

۲ .زیرروال پایداری (Counting Sort): برای توزیع اعداد در هر مرحله، حتماً باید از Counting Sortاستفاده شود تا پایداری (Stability) الگوریتم حفظ شود. استفاده از تابع ()sortداخلی پایتون در هر بخش از کد ممنوع است.

۳ .ردیابی وضعیت میانی :سامانه باید قادر باشد وضعیت آرایه را پس از مرحله دوم (Pass 2) ذخیره کرده و «میانه (Median) «آرایه را در آن لحظه محاسبه کند.

ب) قالب ورودی و خروجی:

- ورودی :در یک سطر، مجموعه‌ای از اعداد صحیح مثبت (شناسه‌های تراکنش) که با فاصله از هم جدا شده‌اند دریافت می‌شود.
- خروجی:

۱ .در سطر اول، لیست مرتب شده اعداد را چاپ کنید.

۲ .در سطر دوم به (صورت کامنت) با علامت #، مبنای انتخاب شده و تعداد مراحل (Passes) را نمایش دهید.

۳ .در سطر سوم (به صورت کامنت)، مقدار میانه آرایه را دقیقاً پس از اتمام مرحله دوم مرتب‌سازی چاپ کنید.

ورودی:

9876543210 1029384756 5555566666 1234567890 9999999999 1010101010 4545454545 8888877777 1234123412

خروجی نمونه:

1010101010 1029384756 1234123412 1234567890 4545454545 5555566666 8888877777 9876543210 9999999999

Base: 1000, Passes: 4

#Median after pass 2: 9876543210