آمادگی برای آزمون میانی طراحی الگوریتم

۱) دنباله‌ی اعداد صحیح متمایز $a_1, \ldots, a_n$ را در نظر بگیرید. می‌گوییم زوج (a_i, a_j) بیش وارونگی دارند هر گاه $i < j$ و $a_i > 2a_j$. الگوریتمی طراحی کنید که تعداد بیش‌وارونگی‌های یک دنباله را در زمان $O(n \log n)$ مشخص کند.

الگوریتم را به صورت استقرایی بیان می‌کنیم. فرض می‌کنیم برای $n-1$ عدد بتوانیم: الف) به درستی تعداد بیش وارونگی‌ها بدست آوریم. ب) با پیچیدگی زمانی $O(\log n)$ تعداد اعداد بزرگ‌تر از یک عدد داده شده را از بین $n-1$ عدد ورودی بیابیم.

حال با داشتن n عدد، باید به صورت استقرایی تعداد بیش وارونگی‌ها را بیابیم و داده ساختاری که برای شمارش اعداد بزرگ‌تر از یک عدد به کار می‌رود را به روز کنیم. بیش‌وارونگی‌های n عدد، شامل بیش‌وارونگی‌های موجود در بین $n-1$ عدد اول (که شامل a_n نیستند) و بیش‌وارونگی‌هایی که a_n را در بر می‌گیرند، هستند. با استفاده از فرض استقرا تعداد بیش‌وارونگی‌ها بین $n-1$ عدد اول را بدست می‌آوریم. سپس با پیچیدگی زمانی $O(\log n)$ (با استفاده از فرض استقرا) تعداد اعداد موجود در بین $n-1$ عدد اول که از $2a_n$ بزرگ‌تر هستند را محاسبه می‌کنیم. با جمع این دو عدد تعداد بیش‌وارونگی‌های n عدد اول محاسبه خواهد شد. حال باید داده‌ساختار مناسبی مشخص کنیم که با پیچیدگی زمانی $O(\log n)$ بتوان تعداد اعداد بزرگ‌تر از یک عدد داده شده را یافت و با همین پیچیدگی زمانی بتوان یک عدد را در آن وارد کرد؛ درخت‌های قرمز-سیاه (Red-black) یا AVL این ویژگی‌ها را دارند. بنابراین دو فرض استقرا برای n عدد نیز محاسبه خواهند شد.

در شبه‌کد زیر این الگوریتم نشان داده شده است. Q یک درخت قرمز-سیاه است که عمل insert برای اضافه کردن یک عدد و bigger برای شمارش اعداد بزرگ‌تر از یک عدد داده شده را پشتیبانی می‌کند. با توجه به اینکه این عملیات دارای پیچیدگی زمانی $O(\log n)$ می‌باشند و در حلقه‌ی این الگوریتم n بار فراخوانی می‌شوند، الگوریتم از پیچیدگی زمانی $O(n \log n)$ می‌باشد.

```
# Calculate the number of inversions
count = 0                          # the number of inversions
Q = {}                             # a red-black tree
for i = 1 to n
      count = count + bigger(Q, a_i * 2)
      insert(Q, a_i)
return count
```

۲) تعداد n گل و n گلدان داریم که با شماره‌های ۱ تا n شماره‌گذاری شده‌اند. گل i در گلدان j به اندازه‌ی $b_{i,j}$ زیبایی دارد. این زیبای‌ها می‌توانند منفی باشند. می‌دانیم به علت حساس بودن این گل‌ها اگر گلی به شماره‌ی i در گلدان شماره‌ی j باشد، هیچ گلی با شماره‌ی کم‌تر از i نمی‌تواند در گلدانی به شماره‌ی بیش‌تر از j قرار بگیرد. می‌خواهیم تعداد از این گل‌ها را در گلدان‌ها بکاریم طوری که مجموع زیبایی آن‌ها بیشینه شود. الگوریتم کارایی برای این کار ارائه دهید.

مسئله را به زیر مسئله‌هایی تقسیم می‌کنیم: $O_{i,j}$ یعنی بیشترین زیبایی که از قرار دادن i گل اول در j گلدان اول حاصل می‌شود (بدیهی است که برخی از گل‌ها می‌توانند استفاده نشوند و برخی از گلدان‌ها خالی بمانند). در این صورت $O_{n,n}$ بیشترین زیبایی ممکن از قرار دادن گل‌ها در گلدان‌ها خواهد بود.

جواب این مسئله را به صورت استقرایی بیان می‌کنیم: در جوابی که به زیبایی $O_{i,j}$ برای قرار دادن i گل اول در j گلدان اول دست می‌یابد (به ازای $i \geq ۱$ و $j \geq ۱$) یا گل i-ام در گلدان j-ام قرار می‌گیرد (حالت اول) یا گل i-ام در یکی از گلدان‌های قبل از آن قرار می‌گیرد (حالت دوم) یا گل i-ام در هیچ یک از گلدان‌ها قرار نمی‌گیرد (حالت سوم). در حالت اول، $O_{i,j}$ برابر $O_{i-۱,j-۱} + b_{i,j}$ می‌باشد (در صورتی که گل i در گلدان j قرار بگیرد زیبایی $b_{i,j}$ از این جایدهی حاصل می‌شود و قرار دادن سایر گل‌ها در سایر گلدان‌ها به حداکثر زیبایی $O_{i-۱,j-۱}$ دست می‌یابد). در حالت دوم، با استدلال مشابهی $O_{i,j}$ برابر $O_{i,j-۱}$ می‌باشد و در حالت سوم $O_{i,j}$ برابر $O_{i-۱,j}$ می‌باشد. چون $O_{i,j}$ بیشترین زیبایی ممکن برای این زیر مسئله را مشخص می‌کند، حداکثر مقدار بدست آمده برای این سه حالت برابر $O_{i,j}$ خواهد بود.

الگوریتم زیر مقدار $O_{i,j}$ را محاسبه می‌نماید. O آرایه‌ای $n \times n$ است که مقادیر $O_{i,j}$ را در خود ذخیره می‌کند. چون بدنه‌های حلقه‌های این الگوریتم $O(n^2)$ بار اجرا می‌شوند، الگوریتم از پیچیدگی زمانی $O(n^2)$ می‌باشد. بهترین جواب مسئله در $O[n, n]$ قرار می‌گیرد.

```
# b[i, j]: the value of placing flower i in flowerpot j
# O[i, j]: the optimal value of placing the first i flowers
#          in the first j flowerpots
for i = 1 to n:
        O[i, 0] = 0;
for i = 1 to n:
        O[0, j] = 0;
for i = 1 to n:
        for j = 1 to n:
                o1 = O[i - 1, j - 1] + b[i, j];
                o2 = O[i, j - 1];
                o3 = O[i - 1, j];
                O[i, j] = max(o1, o2, o3);
```

با فراخوانی print_placement(n, n)، رویه‌ی زیر به صورت بازگشتی بهترین چینش گل‌ها را با توجه به مقدار-های آرایه‌ی O چاپ می‌کند.

```
print_placement(i, j):
        if i == 0 or j == 0:
                return;
        o1 = O[i - 1, j - 1] + b[i, j];
        o2 = O[i, j - 1];
        o3 = O[i - 1, j];
        if O[i, j] == o1:
                print i, "->", j;
                print_placement(i - 1, j - 1);
        else if O[i, j] == o2:
                print_placement(i, j - 1);
        else if O[i, j] == o3:
                print_placement(i - 1, j - 1);
```

۳) الگوریتم بـزرگ‌ترین زیـر دنباله‌ی مشترک (یـا کوله‌پشتـی، کد هافمـن، تراز دنبـاله‌ها، ضرب اعداد بـزرگ، یا سایـر مسئله‌های مطرح شده در کلاس) را روی ورودی‌های زیر، با نشان دادن وضعیت نهایـی داده ساختارهـای به کـار رفته، اجرا نمایید.

برای حل این دسته از مسائل لازم است الگوریتم مسئله‌ی نام برده را بدانید یا بتوانید در زمان امتحان به خاطر آوریـد. برای الگوریتم‌های برنامه‌ریزی پویا جدول زیر مسئله‌ها و بـرای الگوریتم‌هـای حریصانه یـا استقرایی، ساختـار جواب را نمایش دهید.