

راهنمای گام به گام

به پروژه صفر درس ساختمان داده‌ها و الگوریتم‌ها خوش آمدید!

در این راهنمای قصد داریم قدم به قدم با هم پیش برویم و از ساده‌ترین روش، شروع به حل یک مسئله زیبا کنیم.

مطالعه‌ی راهنمای قبلاً از کارگاه خالی از لطف نیست و سعی کرده‌ایم راهنمای را طوری بنویسیم که به تنها یک خارج از کارگاه هم بتوانید پیش بروید. در حین کارگاه هم از روی همین راهنمای خواهیم رفت. همچنین دستیاران آموزشی در کارگاه با شما خواهند بود تا در ابهامات و اشکالات احتمالی همراهیتان کنند.

مقدمه

همگی ما با مبانی برنامه‌نویسی C آشنایی داریم و احتمالاً به واسطه درس برنامه‌سازی پیشرفتی با Java هم آشنا شده‌ایم. همچنین هیچ بعید نیست عده‌ای از شما برنامه‌نویسی به زبان Python را هم بلد باشید. در پروژه‌های این درس از زبان‌های C و یا C++ بهره خواهیم برداشت. (پایتون هم به پروژه صفر اضافه گردید).

می‌دانستیم که هر برنامه برای اجرا مقدار مشخصی زمان نیاز دارد که این زمان تقریباً با تعداد دستورهای اجرا شده در آن برنامه متناسب است. در درس «ساختمان داده‌ها و الگوریتم‌ها» با مفهوم نماد O و احتمالاً Ω و Θ آشنا شدیم و حالا در این پروژه می‌خواهیم دست به کد شویم و زمان مورد نیاز برای اجرای راه حل‌های متفاوت با اُردرهای زمانی متفاوت را به طور عملی ببینیم.

چاپ زمان اجرای کد

با یک سرج ساده‌ی «زمان اجرا زبان فلان» یا "execution time folan language" می‌توان به سادگی مطالب زیر را پیدا کرد که ما برای راحتی شما و سرعت بخشیدن به کارگاه این کار را برایتان از قبل کرده‌ایم. با کلیک بر روی هر کدام از جعبه‌های زیر نحوه چاپ زمان اجرا را در زبان مورد نظر ببینید و حتماً در سیستم خودتان هم تست کنید! خروجی این برنامه‌ها به میلی‌ثانیه است.

C++ ▼

```
1 #include <iostream>
2 #include <ctime>
3 using namespace std;
4
5 int main()
6 {
7     clock_t Start = clock();
8     //your code...
9     //int n;
10    //cin >> n;
11    //for(int i = 0; i < n; i++)
12    //    if(i%1000000 == 0)
13    //        cout << "";
14    clock_t End = clock();
15    cout << int((double(End - Start)/double(CLOCKS_PER_SEC))*1000) << "\n";
16 }
```

دقت کنید که ممکن است گاهی قسمت‌های زائدی از کد ما توسط کامپایلر حذف شود، یا بخش‌های ساده‌ای از کدامان توسط کامپایلر بهینه‌سازی شوند. مثلاً اگر یک حلقه خالی در این کد قرار بدهید مشاهده می‌کنید که هر چقدر شرط پایان حلقه بزرگ باشد باز هم کد در حدود ۰ میلی‌ثانیه اجرا می‌شود.

کد Python ▼

```
1 import time
2 Start = int(round(time.time() * 1000))
3 #your code...
4 #b = int(0)
5 #for a in range(0, 5000000):
6 #    b = b+a
7 End = int(round(time.time() * 1000))
8 print(End-Start)
```

(با اجرای کد کامنت شده، و تغییر عدد ثابت آن می‌توانید حدود تعداد عملیاتی که سیستم شما در ۱ ثانیه اجرا می‌کند بدست آورید).

صورت مسئله

دنباله‌ای از اعداد صحیح به طول n به ما داده شده است. به هر بازه‌ی پیوسته‌ی i تا j از دنباله یک «زیردنباله» می‌گوییم. پس n زیردنباله به طول ۱ داریم و همچنین $1 - n$ زیردنباله به طول ۲ داریم و ... حال تمامی زیردنباله‌های ممکن را تصور کنید. خواسته‌ی مسئله، یافتن بیشینه‌ی مجموع اعداد از بین تمام زیردنباله‌هاست.

برای مثال به نمونه زیر دقت کنید:

```
index :[0  1  2 3  4 5  6 7  8]
numbers: 5 -7 10 2 -4 5 -7 6 -2
maximum sum: ^.....^ = 10+2-4+5 = 13
```

بررسی کنید که آیا هیچ زیردنباله دیگری مجموع بیشتری نداشت؟!

فرمت ورودی و خروجی مسئله

در خط اول ورودی عدد n می‌آید که نشان‌دهنده اندازه دنباله ورودی است و در خط بعدی n عدد صحیح با فاصله از یکدیگر می‌آیند. در خروجی کافیست بزرگ‌ترین مجموع زیردنباله ممکن را چاپ کنید.

ورودی نمونه

```
9
5 -7 10 2 -4 5 -7 6 -2
```

خروجی نمونه

13

نحوه ارزیابی پروژه

همانطور که مشاهده می‌کنید پروژه بر اساس اینکه اندازه دنباله اولیه یعنی n تا چه اندازه بزرگ باشد به ۴ زیرمسئله تقسیم شده و همچنین در انتهای یک نمودار نیز از شما خواسته شده است.

ایده و راه حل هر ۴ زیرمسئله را در ادامه با هم می‌بینیم توضیحات نمودار خواسته شده هم در سوال خودش توضیح داده شده است. توصیه می‌شود کدهای هر ۴ زیرمسئله‌تان را نگهداری و با روش خواسته شده پیاده‌سازی کنید چرا که در رسم نمودار مجدداً به آن‌ها نیاز پیدا می‌کنید.

نمره‌ی زیرمسئله‌های یکم تا چهارم توسط داوری آنلاین کوئرا داده خواهد شد که در آن اولاً صحیح بودن خروجی کد شما در ازای تعدادی ورودی و ثانیاً مدت زمان اجرای کد شما برای هر ورودی سنجیده می‌شود. (که اگر توضیحات زیر را دنبال کنید، احتمالاً به مشکلی برخواهید خورد.)

نمره‌ی بخش «نمودار»، توسط دستیاران آموزشی داده خواهد شد. توضیحات بیشتر در مورد بخش «نمودار» در بخش خودش آورده شده است.

بر روی هر زیرمسئله فقط به راه حل‌های هم اُردر (هم تنا در واقع) با راهنمای آن نمره تعلق می‌گیرد. (یعنی ارسال راه حل زیرمسئله ۱ در زیرمسئله ۲ یا برعکس مجاز است، اما نمره‌ای به آن تعلق نمی‌گیرد.) این موضوع توسط دستیاران آموزشی بررسی خواهد شد. در صورت انجام این کار، هرچند داوری کوئرا به شما نمره‌ی کامل داده باشد، نمره‌ی آن زیرمسئله برای شما صفر در نظر گرفته خواهد شد. همچنین توجه داشته باشید که «ارسال نهایی» شما، تنها ارسال شما محسوب خواهد شد.

زیرمسئله یکم

اول از همه بباید ساده‌ترین راه حل ممکن را برای مسئله پیاده‌سازی کنیم. دو متغیر α و β در نظر بگیرید به کمک این دو و با استفاده از حلقه‌ی تودرتو تمام شروع و پایان‌های ممکن را برای زیردنباله در نظر بگیرید. حال به ازای هر حالت از α و β روی تمام عناصر با اندیس‌های i تا j فور بزنید و مقادیر آن‌ها را با هم جمع کنید تا مجموع عناصر این زیردنباله بدست بباید. حال این مجموع را بیشترین مجموعی که تا بحال به دست آورده‌ایم ماقسیم بگیرید و به سراغ α و β های بعدی بروید.

سعی کنید الگوریتم بالا را پیاده‌سازی و سپس تحلیل اُردر کنید!

▼ اُردر

این زیرمسئله ممکن است بدیهی باشد اما برای تقویت خودتان هم که شده هیچوقت قبل از فکر کردن کافی(!) جواب هیچ مسئله‌ای را نگاه نکنید.

▼ دیگه واقعاً اُردر

الگوریتم فوق با نظر به اینکه ۳ حلقه‌ی تودرتو دارد از $O(n^3)$ می‌باشد. نه تنها از O بلکه از $\Theta(n^3)$ می‌باشد.

محاسبه زمان تقریبی اجرا

محاسبه زمان تقریبی اجرای برنامه‌ها کار دشواری نیست. عموماً این موضوع به سیستمی که کد را اجرا می‌کند هم مربوط می‌شود اما یک استاندارد و حدود مشخصی دارد و در واقع می‌تواند نشان دهد که کدام مثلاً یک ساعت زمان برای اجرا نیاز ندارد و در حدود یک ثانیه یا کمتر به جواب می‌رسد.

استاندارد حدودی این‌گونه است: تعداد عملیات‌های برنامه را می‌شماریم، در یک برنامه به زبان C++ یا C فرض می‌کنیم که هر $10^8 * 2$ عملیات در حدود یک ثانیه اجرا می‌شود. این عدد به سخت‌افزار و قدرت پردازش سیستم هم بستگی دارد. (توی پرانتر بگم که رزو کردن حافظه در هنگام شروع اجرای برنامه هم تا حدی زمان نیاز دارد، مثلاً وقتی یک آرایه‌ی خیلی بزرگ تعریف می‌کنیم زمان اجرای برنامه هم زیاد می‌شود. فعلاً تا وقتی به مشکلش برخوردمید این پرانتر رو نادیده بگیرید.)

در این سوال ما برنامه‌ای با حدود n^3 عملیات نوشته‌یم. به سوال «زیرمسئله یکم» بروید و محدودیت n و به خصوص حداقل مقدارش را مشاهده کنید. آیا n^3 عملیات در کمتر از یک ثانیه انجام می‌شود؟

خب حالا با همین برنامه به سراغ «زیرمسئله دوم» بروید. سنگ مفت، گنجشک مفت، شاید اکسپت شد! البته به محدودیت n در این سوال هم گوشه چشمی داشته باشید. حدود n^3 عملیات با این n در یک ثانیه انجام می‌شود؟

قبل از رفتن به سراغ زیرمسئله دوم بهتر است «یکم» را اکسپت کرده باشید. اگر در کارگاه هستید و به مشکلی خوردمید با دستیارآموزشی مطرح کنید.

زیرمسئله دوم

در این زیرمسئله اندازه ورودی بزرگتر خواهد بود و الگوریتم قبلی ما جوابگوی حل آن در زمان مناسب نیست. چند دقیقه‌ای فکر کنید و سعی کنید یکی از حلقه‌ها را حذف کنید...

خب حالا که فکر کردید، فرض کنید θ را فیکس کرده باشیم و در حال فور زدن روی \mathcal{J} باشیم. فرض کنید مجموع زیردنباله θ تا $1 - \mathcal{J}$ را داشته‌ایم، آیا برای محاسبه مجموع زیردنباله θ تا \mathcal{J} فور زدن روی تمامی عناصرش نیاز است؟ کافی نیست که همان مجموع زیردنباله θ تا $1 - \mathcal{J}$ را با عنصر \mathcal{J}^m جمع کنیم؟ با توجه به این موضوع از \mathcal{J} حلقه تودرتوی ما یکی به سادگی حذف می‌شود. همچنین می‌توانید اُردر زمانی الگوریتم جدید را هم حساب کنید. حالا با این الگوریتم زیرمسئله دوم هم به سادگی اکسپت می‌شود و حدود n^2 عملیاتش در یک ثانیه می‌گنجد.

▼ اُردر

الگوریتم جدید با نظر به اینکه \mathcal{J} حلقه‌ی تودرتو دارد از $O(n^2)$ می‌باشد. نه تنها از O بلکه از $\Theta(n^2)$ می‌باشد.

پس از گرفتن اکسپت زیرمسئله دوم، کدتان را روی زیرمسئله سوم هم می‌توانید تست کنید. البته قبل از آن بد نیست محدودیت n را نگاه کنید.

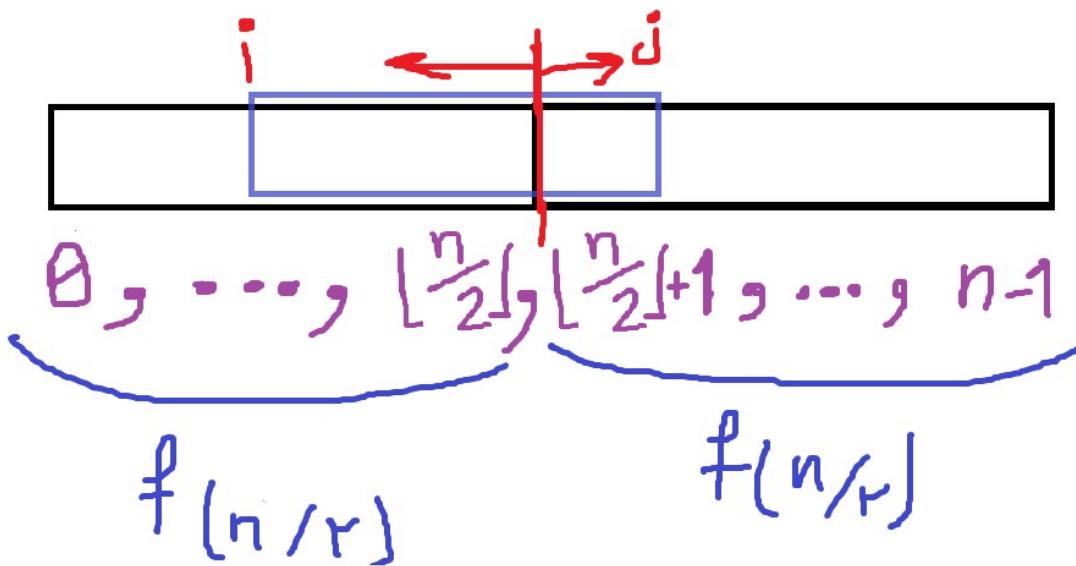
قبل از رفتن به سراغ زیرمسئله سوم بهتر است «دوم» را اکسپت کرده باشید. اگر در کارگاه هستید و به مشکلی خورده‌ید با دستیارآموزشی مطرح کنید.

زیرمسئله سوم

خب همانطور که می‌دانید اگر الگوریتم قبلی را روی زیرمسئله سوم ارسال کنید، با خطای محدودیت زمانی یا *Time Limit Exceeded* مواجه می‌شوید و اکسپت نمی‌شود. باید به سراغ الگوریتمی سریع‌تر برویم. اگر دوست داشتید کمی در این باره فکر کنید و سپس ادامه متن را مطالعه کنید. کمی هم با دانستن اینکه شاید یک الگوریتم بازگشتی بتوانیم بنویسیم به مسئله فکر کنید.

خب برویم سراغ راه حل سوم. فرض کنید حل کردن مسئله را به ازای k ‌های کوچک‌تر از n بلد هستیم. دنباله را به دو نیمه‌ی تقریباً مساوی تقسیم می‌کنیم. هر کدام از نیمه‌ها را طبق فرض حل می‌کنیم و بیشینه مجموع زیردنباله‌هایش را بدست می‌آوریم. جواب برای دنباله اصلی ما یا یکی از این دو عدد به دست آمده

است و یا مجموع عناصر زیردنباله‌ای است که بخشی از آن در نیمه سمت راست دنباله و بخشی از آن هم در نیمه سمت چپ دنباله است. (چرا که اگر کامل در یکی از این دو نیمه باشد هنگام حل آن نیمه حتماً مورد بررسی قرار گرفته است).



سعی کردیم برای فهم بهتر موضوع تصویری هم آماده کنیم. خب پس جواب یا در یکی از $(n/2)$ هاست و یا در زیردنباله‌ای مثل $\{z_j\}$. خب یک نکته‌ای که اینجا داریم این است تمام زیردنباله‌هایی که حالا دنبالشان هستیم از چسباندن دو زیردنباله که یکی حتما از $1 + \frac{n}{2}$ شروع می‌شود و یکی حتما به $\frac{n}{2}$ ختم می‌شود تشکیل می‌شود. پس کافیست یک حلقه بیشترین جمع برای زیردنباله‌هایی که از $1 + \frac{n}{2}$ شروع می‌شود را پیدا کنیم و با یک حلقه هم بیشترین جمع برای زیردنباله‌هایی که به $\frac{n}{2}$ ختم می‌شوند. مجموع این دو می‌شود بیشینه مد نظر ما که باید با $(n/2)f$ مقایسه شود و یکی از این 3 جواب نهایی است. هنوز فقط الگوریتم را گفته‌ایم و در ادامه پیاده‌سازی را هم خواهیم گفت. اما قبل از آن بباید تحلیل اُردر کنیم. ابتدا سعی کنید خودتان حل کنید و سپس به سراغ جعبه زیر بروید.

اُرڈر ▼

فرض کنید $T(n)$ تابع زمان اجرای برنامه‌مان باشد. می‌دانیم که

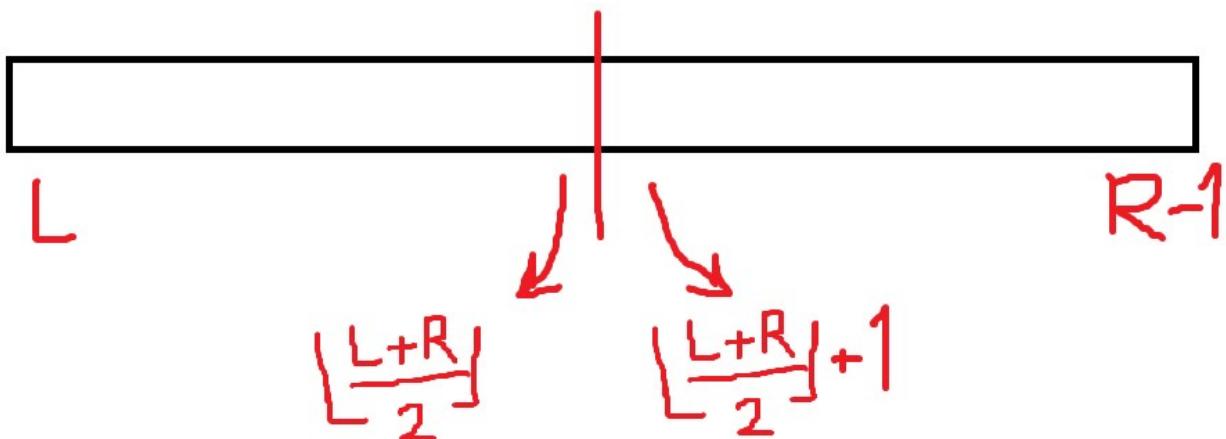
$$T(n) = 2 * T(n/2) + n/2 + n/2 = 2T(n/2) + O(n)$$

و اگر احتمالا مشایه این رابطه را دیده باشید می‌دانید که

$$T(n) = O(n \lg n)$$

اگر ندیده باشید در ادامه خواهید آموخت.

خب حالا الگوریتم را می‌دانیم. چطور پیاده‌سازی کنیم؟ آرایه اعداد را به صورت گلوبال و بالای کد تعریف می‌کنیم، یکتابع بازگشتی می‌نویسیم که ورودی‌اش، اندیس شروع و پایان بازه مدنظر را می‌گیرد و مسئله را برای آن زیردبالة حل می‌کند. اکثر اوقات بازه‌ها را به صورت بسته-باز می‌گیریم. یعنی ابتدای آن‌ها را بسته و انتهای آن‌ها را باز. یا به عبارتی به این صورت: $[start, end]$ خب حالا به تصویر زیر نگاه کنید:



در $F(L, R)$ چه کارهایی باید کنیم؟ در توابع بازگشتی ابتدا حالت پایه را می‌نویسیم. کافیست هرگاه اندازه بازه ۱ بود همان عدد $[L, R]$ را برگردانیم (البته اگر آن یک عدد منفی بود، صفر را برمی‌گردانیم).

بعد از مشخص کردن و درست کردن پایه باید

۱. نیمه اول را صدا بزنیم تا به صورت بازگشتی جوابش محاسبه شود.
۲. نیمه دوم را صدا بزنیم تا به صورت بازگشتی جوابش محاسبه شود.
۳. و در آخر با دو حلقه (یکی از وسط به آخر و دیگری از وسط به ابتدا) بیشینه‌ی مجموع زیردباله‌های بنفس گونه (در تصور اولی) را به دست آوریم. بیشینه‌ی این سه، خروجی تابع ما خواهد بود.

قبل از رفتن به سراغ زیرمسئله چهارم بهتر است «سوم» را اکسپت کرده باشید. اگر در کارگاه هستید و به مشکلی خوردید با دستیارآموزشی مطرح کنید.

زیرمسئله چهارم!

شاید تعجب کنید اما این مسئله از این هم سریع‌تر می‌تواند حل شود :) اگر علاقه و وقت داشتید جا دارد تا چند ساعت روی این مسئله فکر کنید اگر هم هر یک را نداشتید در ادامه به حل می‌پردازیم.

▼ ایده

فرض کنید یک آرایه به اسم p و به طول n بسازیم و عنصر i^* این آرایه برابر با مجموع عناصر 0 تا i آرایه ورودی باشد. ساختن آرایه p به سادگی و با یک حلقه در $O(n)$ قابل انجام است.

▼ فایده‌اش؟

دباله‌ی p را همان‌طور که در جعبه‌ی بالا توضیح داده شده بسازید. حال هر زیردباله از عناصر دبالتی اصلی را که در نظر بگیرید در $O(1)$ و فقط با یک تفربیق ساده بین دو عضو p ، مجموع تمام عناصرش را داریم. برای مثال برای محاسبه مجموع اعداد 0 تا j کافیست از $[j] p[i] - [i] p[1]$ مقدار را کم کنیم.
(حوالستان به اندیس 0 باشد!)

خب با این ایده هم کمی به مسئله فکر کنید.

برویم سراغ حل. زیردباله‌ای که جواب مسئله است، انتهاییش یا روی اندیس 1 یا ... و یا روی اندیس $1 - n$. برای حالت‌بندی روی این انتهای زیردباله، رویش فور می‌زنیم و این اندیس را j در نظر می‌گیریم. حال فرض کنید مینیمم $[i] p$ ‌ها به ازای $j < i \leq 0$ را داریم. در این صورت بیشینه زیردباله‌ای که به اندیس j ختم می‌شود برابر است با $(p[i] - min(p[i] - p[j]))$. اگر بخواهیم مقدار مینیمم $[i] p$ ‌ها را هر سری محاسبه کنیم حلقه تودرتو داریم و زمان اجرا از $O(n^2)$ خواهد بود. اما دوای این درد را قبلا هنگام تبدیل راه حل اول به دوم دیده‌ایم. اینجا هم درست به همان صورت وقتی j زیاد می‌شود نیاز نیست که ما مینیمم $[i] p$ ‌ها را هر بار از اول حساب کنیم. بلکه کافی است مقدار آن را با تک عنصر جدید که به محدوده اضافه شده است (یعنی $[1 - j] p$) مقدار مینیمم $[i] p$ ‌ها را به‌روزرسانی کنیم. به این ترتیب به الگوریتمی از چه اُردری می‌رسیم؟

▼ اُردر

$O(n)$

در صورت علاقه پس از گرفتن نمره کامل زیرمسئله، سعی کنید با تنها یک آرایه راه حل را پیاده‌سازی کنید.

زیرمسئله یکم

• محدودیت زمان: ۱ ثانیه

• محدودیت حافظه: ۲۵۶ مگابایت

بر روی هر زیرمسئله فقط به راه حل‌های هم اُردر با راهنمای آن نمره تعلق می‌گیرد. (یعنی ارسال راه حل زیرمسئله ۱ در زیرمسئله ۲ یا بر عکس مجاز است، اما نمره‌ای به آن تعلق نمی‌گیرد). این موضوع توسط دستیاران آموزشی بررسی خواهد شد. در صورت انجام این کار، هرچند داوری کوئٹرا به شما نمره‌ی کامل داده باشد، نمره‌ی آن زیرمسئله برای شما صفر در نظر گرفته خواهد شد.

ورودی

ورودی شامل دو خط است. در خط اول عدد طبیعی n آمده است و در خط بعدی n عدد صحیح با فاصله از هم آمده‌اند که دنباله اعداد است.

$$1 \leq n \leq 400$$

خروجی

خروجی برنامه‌ی شما باید تنها شامل یک عدد باشد که بیشینه مجموع زیردنباله‌هاست.

مثال

ورودی نمونه ۱

5

-1 -2 10 -2 3

خروجی نمونه ۱

11

ورودی نمونه ۲

6

200 -100 5 -1 2 -10

خروجی نمونه ۲

200

ورودی نمونه ۳

6

-200 -100 -5 -1 -2 -10

خروجی نمونه ۳

0

در چنین حالتی زیر丹باله تهی که مجموع عناصرش ° تعریف می‌شود را انتخاب می‌کنیم.

زیرمسئله دوم

• محدودیت زمان: ۱ ثانیه

• محدودیت حافظه: ۲۵۶ مگابایت

بر روی هر زیرمسئله فقط به راه حل‌های هم اُردر با راهنمای آن نمره تعلق می‌گیرد. (یعنی ارسال راه حل زیرمسئله ۱ در زیرمسئله ۲ یا برعکس مجاز است، اما نمره‌ای به آن تعلق نمی‌گیرد). این موضوع توسط دستیاران آموزشی بررسی خواهد شد. در صورت انجام این کار، هرچند داوری کوئٹرا به شما نمره‌ی کامل داده باشد، نمره‌ی آن زیرمسئله برای شما صفر در نظر گرفته خواهد شد.

ورودی

ورودی شامل دو خط است. در خط اول عدد طبیعی n آمده است و در خط بعدی n عدد صحیح با فاصله از هم آمده‌اند که دنباله اعداد است.

$$1 \leq n \leq 5\,000$$

خروجی

خروجی برنامه‌ی شما باید تنها شامل یک عدد باشد که بیشینه مجموع زیردنباله‌هاست.

مثال

ورودی نمونه ۱

5

-1 -2 10 -2 3

خروجی نمونه ۱

11

ورودی نمونه ۲

6

200 -100 5 -1 2 -10

خروجی نمونه ۲

200

ورودی نمونه ۳

6

-200 -100 -5 -1 -2 -10

خروجی نمونه ۳

0

در چنین حالتی زیر丹باله تهی که مجموع عناصرش ° تعریف می‌شود را انتخاب می‌کنیم.

زیرمسئله سوم

• محدودیت زمان: ۱ ثانیه

• محدودیت حافظه: ۲۵۶ مگابایت

بر روی هر زیرمسئله فقط به راه حل‌های هم اُردر با راهنمای آن نمره تعلق می‌گیرد. (یعنی ارسال راه حل زیرمسئله ۱ در زیرمسئله ۲ یا برعکس مجاز است، اما نمره‌ای به آن تعلق نمی‌گیرد). این موضوع توسط دستیاران آموزشی بررسی خواهد شد. در صورت انجام این کار، هرچند داوری کوئڑا به شما نمره‌ی کامل داده باشد، نمره‌ی آن زیرمسئله برای شما صفر در نظر گرفته خواهد شد.

ورودی

ورودی شامل دو خط است. در خط اول عدد طبیعی n آمده است و در خط بعدی n عدد صحیح با فاصله از هم آمده‌اند که دنباله اعداد است.

$$1 \leq n \leq 200\,000$$

خروجی

خروجی برنامه‌ی شما باید تنها شامل یک عدد باشد که بیشینه مجموع زیردنباله‌هاست.

مثال

ورودی نمونه ۱

5

-1 -2 10 -2 3

خروجی نمونه ۱

11

ورودی نمونه ۲

6

200 -100 5 -1 2 -10

خروجی نمونه ۲

200

ورودی نمونه ۳

6

-200 -100 -5 -1 -2 -10

خروجی نمونه ۳

0

در چنین حالتی زیر丹باله تهی که مجموع عناصرش ° تعریف می‌شود را انتخاب می‌کنیم.

زیرمسئله چهارم

• محدودیت زمان: ۱ ثانیه

• محدودیت حافظه: ۲۵۶ مگابایت

بر روی هر زیرمسئله فقط به راه حل‌های هم اُردر با راهنمای آن نمره تعلق می‌گیرد. (یعنی ارسال راه حل زیرمسئله ۱ در زیرمسئله ۲ یا برعکس مجاز است، اما نمره‌ای به آن تعلق نمی‌گیرد). این موضوع توسط دستیاران آموزشی بررسی خواهد شد. در صورت انجام این کار، هرچند داوری کوئڑا به شما نمره‌ی کامل داده باشد، نمره‌ی آن زیرمسئله برای شما صفر در نظر گرفته خواهد شد.

ورودی

ورودی شامل دو خط است. در خط اول عدد طبیعی n آمده است و در خط بعدی n عدد صحیح با فاصله از هم آمده‌اند که دنباله اعداد است.

$$1 \leq n \leq 2 * 10^6$$

خروجی

خروجی برنامه‌ی شما باید تنها شامل یک عدد باشد که بیشینه مجموع زیردنباله‌هاست.

مثال

ورودی نمونه ۱

5

-1 -2 10 -2 3

خروجی نمونه ۱

11

ورودی نمونه ۲

6

200 -100 5 -1 2 -10

خروجی نمونه ۲

200

ورودی نمونه ۳

6

-200 -100 -5 -1 -2 -10

خروجی نمونه ۳

0

در چنین حالتی زیر丹باله تهی که مجموع عناصرش ° تعریف می‌شود را انتخاب می‌کنیم.

نمودار

برای این بخش، کدهایی که در زیرمسئله‌های یکم تا چهارم زدید را روی تست‌کیس‌هایی که در کانال قرار گرفته‌اند اجرا کرده و هر بار زمان اجرای کد را اندازه بگیرید. سپس برای هر زیرمسئله، منحنی زمان اجرای برنامه نسبت به n رارسم کنید. منحنی‌ها را در یک نمودار (که با هم قابل مقایسه باشند) و در قالب یک فایل زیپ یا pdf آپلود کنید.

نمودار امتیازی

این بار از شما می‌خواهیم روی هر کدام از راه حل‌هاییتان تست‌ها (همان تست‌های سوال قبلی) را اجرا کنید و چهار نمودار جدا بدست بیاورید که محور افقی آن n (اندازه دنباله ورودی) و محور عمودی آن «تعداد عملیات‌های کد» باشد.

برای شمارش تعداد عملیات‌های کد، فرض کنید کافیست تعداد عملیات‌های جمع، ضرب، تفریق و تقسیمی که برنامه شما انجام می‌دهد را بشماریم. برای مثال در کدهای زیر متغیر `cnt` این تعداد را می‌شمارد:

```
1 #include <stdio.h>
2 int cnt = 0;
3 int main(){
4     int a = 2*5 + 12 - 14;
5     cnt += 3;
6
7     int b = (a-3)/2;
8     cnt += 2;
9
10    int n;
11    scanf("%d", &n);
12    for(int i = 0; i < n; i++){//no need to count loop operations like i++
13        b = a-3;
14        cnt += 1
15    }
16    printf("%d\n", cnt);
17 }
```

```
1 cnt = 0
2
3 a = 2*5 + 12 - 14
4 cnt += 3
5
6 b = (a-3)/2
7 cnt += 2
8
```

```
n = input()
for i in range(n):
    b = a-3
    cnt += 1

print(cnt)
```

(به طور پیشفرض فایل ارسالی شما محدودیت حجم ۱۰ مگابایتی دارد. اگر بیش این نیاز بود در گروه درس مطرح بفرمایید.)