به نام خداوند وجد و سرور پدید اور عشق و احساس و شور

دانشکده مهندسی برق و کامپیوتر دانشگاه تهران

طراحی الگوریتم
نیم‌سال دوم سال ۹۸-۹۹
دستیاران : مجید دلیری

راه حل کلی تمرین کامپیوتری اول | **تقسیم و حل** | **تاریخ : ۲۳ ام اسفند ماه ۱۳۹۸**

خب واضحا چون label سوال divide and conquer هستش پس یک دید خوبی از نحوه حل کردن مساله داریم. کلن توی این نوع مسائل ما باید یک داستانی رو به چند قسمت بشکونیم (معمولا دو قسمت) و هر تیکه رو جداگانه حل کنیم و در نهایت این دوتا تیکه رو به اصطلاح merge کنیم.

کلا هم اینو بگم که یکی از روش های حل مساله های الگوریتمی وقتی نمی‌دونیم تگ یا ایده سوال چیه اینه که ایده های مختلف رو روش تست می‌کنیم. می‌ایم می‌بینیم دی‌پی خوره سوال؟، یا مساله تقسیم و حله؟، یا ... که روش حل بر اساس این متد نیاز به این داره که شما تعداد زیادی مساله دیده باشید و به خوبی با ایده های خاص الگوریتم های متفاوت آشنا باشید . خب چون این ایده های خاص محدوده یعنی کل ایده های متفاوت مثلا برای تقسیم و حل کلن سرجمع شاید ۱۰ تا نشه و هر مساله که الان داره توی دنیا طرح میشه نمی‌گم بعیده می‌گم خیلی احتمالش کمه که ایده حلش خارج از این pool ایده ها خارج باشه، خیلی راحت می‌توانید به این مهارت دست پیدا کنید و از این متد استفاده کنید. من هم خواستم با دادن این تمرین ها که ایده هاش شاید یه مقداری جدیدتر باشه pool شما رو یه مقداری گسترش بدم.

مساله‌ی ۱. پشمک فروشی در پشمکستان

من می‌خواهم با همین دید بالا برم روی مساله. خب اول (چه چیزایی رو میشه نصف کرد؟) به گمونم تنها چیزی که میشه نصفش کرد صفه که خب اگر این کارو بکنیم موقع مرج کردن ما به این نیاز داریم که خوشحالی جلو و عقب هر فرد رو بدونیم (که این باید با یه preprocess در بیاد) و اگر اینم داشته باشیم، برای مرج، می‌خواهیم توی خوشحالی های عقب تیکه چپی اونایی که از خوشحالی جلو تیکه راست بزرگتر اند رو پیدا کنیم. اینکه دوتا دوتا بخوایم با هم چک کنیم پیچیدگی زمانی ما رو بد می‌کنه . پس باید از یه trick خاصی استفاده کنیم که خب اینجا اصل کار همینه.
ایده این بود که بیایم خوشحالی جلو تیکه راستی و همچنین خوشحالی عقب تیکه چپی رو سورت کنیم و این کمکی که می‌کنه اینه که مقایسه کردن دو تا عنصر از دو آرایه سورت شده به ما دید میده نسبت به سایر عناصر (نسبت به بزرگترهای عنصر بزرگتر به طور واضح می‌تونیم بگوییم که بزرگتر از عنصر کوچکتر هستند و همچنین نسبت به عناصر کوچکتر از عنصر کوچک از عنصر بزرگ کوچکتر اند) و به صورت زیر این تعداد مد نظر را بدست می‌آوریم.

```
backward_happiness.sort()
forward_happiness.sort()
i,j = 0,0
while i < left_len and j < right_len:
        if backward_happiness [i] <= forward_happiness [j]:
            i = i + 1
        else :
            inversion_count += left_len - i
            j = j + 1
```

پیچیدگی زمانی این الگوریتم شما هم اگر تحلیل کنید به صورت زیر میشه که با توجه به محدودیت های مساله جواب میده.

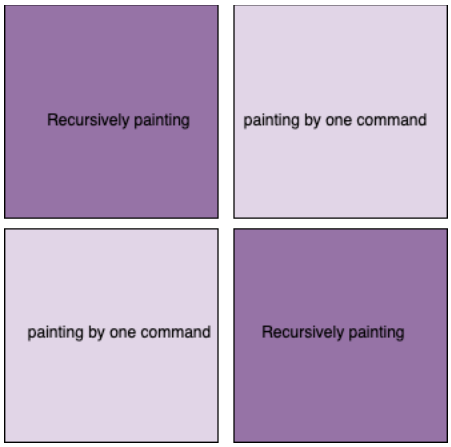
$$T(n) = 2 * T\left(\frac{n}{2}\right) + O(n \log n) + O(n) = O(n \log^2 n)$$

$$n < 10^5 \Longrightarrow T(n) < 10^5 * \log^2 10^5 = 10^5 * 16^2 < 10^8$$

که خب این زیر یک ثانیه انجام میشه خوشبختانه .

مساله‌ی ۲. رنگ زدن دیوار

این سوال بنظر من ساده ترین سوال‌تون بود که ایده اصلی حل سوال به این صورت است که جدول رو از سطر و از ستون به دو نیم می‌کنیم یعنی در حقیقت به ۴ قسمت جدول رو تقسیم می‌کنیم. حالا همانطور که در تصویر می‌بینید دو تا مربع بالا راست و پایین چپ رو هر کدوم‌شون رو با یک دستور که جمعا ۲ تا دستور میشه رنگ کرد (کافیه کل ستون های هر تیکه رو توی دستور بیاریم). حالا با این دو تا دستور فقط مربع بالا چپ و پایین راست فقط رنگ نشده‌اند که کافیه اینا رو به صورت ریکرسیو رنگ کنیم و حالا با توجه به اینکه دستورات‌شون (مربع بالا چپ و پایین راست) روی هم دیگه تاثیر نمی‌گذاره، می‌تونیم دستورات‌شون رو با هم مرج کنیم.



$$T(n) = T\left(\left\lceil\frac{n}{2}\right\rceil\right) + 2 = 2 * \lceil \log n \rceil$$

$$n \leq 1000 \Longrightarrow T(n) \leq 20$$

پس در کل تعداد دستورات‌مون بصورت فوق می‌شه .

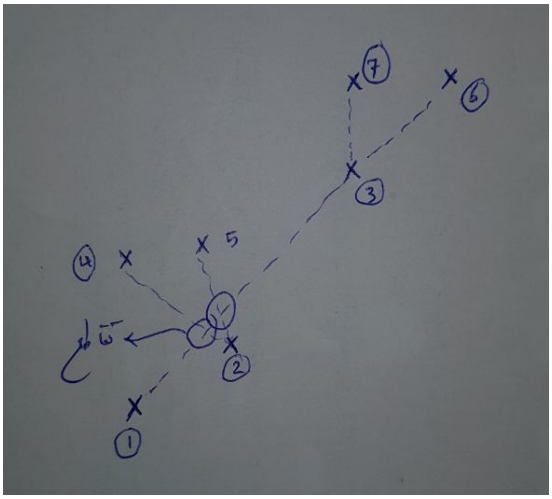
توجه : من قطر اصلی رو به صورت ریاضی گرفتم اما اگر قطر رو برعکس بگیرید مشکلی ایجاد نمی‌کند و تعداد دستورات نیز به همین صورت می‌باشد .

مساله‌ی ۳. نقاشی

خب من این سوال رو دادم که ببینید مسائلی که به نظر پیچیده میان و اصلا فضاشون شاید به فضای الگوریتم نخوره رو هم می‌تونیم با ساده ترین الگوریتم ها حل کنیم. برای حل این سوال شما کافی بود مساله رو روی کاغذ ببرید و باهاش ور برید.

ایده اصلی اینه که ما root رو یه نقطه ای بگیریم که یک طرف اون هیچ نقطه دیگه ای نباشه مثلا راست ترین نقطه یا بالا ترین نقطه (حالا برای چی این‌کار رو می‌کنیم چون این root ای که الان انتخاب میکنیم توی حد فاصل این root و root قبلی هیچ نقطه این نباشه) که من برای حل این سوال پایین ترین نقطه رو به عنوان root قرار دادم. خب حالا که نقطه root رو فیکس کردیم باید بقیه نقاط رو برای دو زیردرخت فرزند چپ و فرزند راست به دو قسمت تقسیم کنیم چون این دو زیر درخت مستقل از هم دیگه هستند پس دو تکه کردن باید یه جوری باشه که دو نیمه نقاط مشترک نداشته باشند و در ضمن یال های آنها تلاقی نداشته باشد.

پس حالا ایده هایی که اولش ممکنه به ذهنمون برسه برای این دو نیم کردن (با توجه به اینکه root پایین ترین نقطه در مجموعه نقاطمون هست) مثلا میتونه این باشه برحسب x نقاط رو سورت و سپس نصف‌شون کنیم این خوبی که داره اینه که دو دسته هیچ ارتباطی با هم ندارند و یال های درون دو دسته با هم هیچ برخوردی نداره اما این مثال نقص داره که به شکل زیره چون پایین ترین نقطه، نقطه ۱ هست پس اگر برحسب x سورت کنیم، نقاط ۲و۴و۵ توی یه دسته می‌افتند و سایر توی یه دسته دیگه پس یال ۱ به ۳ با یال های ۲ به ۴ و ۲ به ۵ تقاطع داره که مشاهده می‌فرمایید .



با این نوع نصف کردن تنها جاهایی که ممکنه یال ها تقاطع داشته باشن یال هایی است که از root ما می‌گذره.

خب ما اگر بیایم روش دو نیم کردن رو یه جوری انجام بدیم که اولا دو بخش هیچ تقاطعی نداشته باشند و ثانیا یال هایی که از root می‌گذرند کامل درون دسته خودش قرار بگیرد مشکل فوق رفع خواهد شد.

برای این هدف مون اگر بیایم بجای x برحسب زاویه خط واصل نقاط با root و خط افق سورت کنیم و نقاط رو نصف کنیم، در واقع اومدیم یه خطی از root کشیدیم و نقاط رو به دو دسته تقسیم کردیم و هر سمت این خط، مربوط به یک ناحیه است. دو طرف این خط هیچ اشتراکی ندارند و یال های گذرنده از root نیز در سمت مربوط به آن فرزند قرار می‌گیرد و با دسته دیگه ارتباطی ندارد پس مشکلات روش فوق رفع می‌گردد.

برای سورت کردن بر حسب زاویه نیازی به بدست آوردن زاویه های خط واصل نقاط با root نیست، چون root پایین ترین نقطه است، زاویه ها از ۰ تا ۱۸۰ درجه می‌باشند و چون در این بازه تابع کسینوس نزولی است پس کافی است بر حسب کسینوس زوایا نقاط را سورت کرده و در نتیجه لیست سورت شده بر حسب زاویه وارونه شده لیست سورت شده بر حسب کسینوس می باشد .

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

$$n \leq 1500 \implies 1500 * 11^2 < 10^8$$

پیچیدگی زمانی این الگوریتم به مانند مساله اول است که با توجه به محدودیت های مشابه زیر ۱ ثانیه قابل اجرا روی کامپیوتر معمولی است .