به نام خداوند بخشنده مهربان

راه حل تمرین شماره یک

حسین بابایی

۵ اسفند ۱۳۹۶

۱. از روش جست و جوی دودویی استفاده می کنیم. برای پیدا کردن بزرگترین عنصر در آرایه $< a_1, ..., a_n >$، مقدار $a_{n/2}$ را مورد بررسی قرار می دهیم. اگر داشته باشیم $a_{n/2} > a_{n/2-1}, a_{n/2} > a_{n/2+1}$، آنگاه پاسخ مساله $a_{n/2}$ است. اگر داشته باشیم $a_{n/2} < a_{n/2+1}$ باشد آنگاه می دانیم که جواب در سمت چپ $a_{n/2}$ قرار دارد و برابر با بزرگترین عدد در آرایه $< a_1, ..., a_{n/2-1} >$ است که خود آرایه ای است که به صورت دایره‌ای شیفت پیدا کرده است و شبیه مساله اصلی می‌باشدو پاسخ آن را به صورت بازگشتی می‌توان یافت. و اگر داشته باشیم $a_{n/2} < a_{n/2+1}$ انگاه پاسخ مساله در سمت راست $a_{n/2}$ قرار دارد و برابر با بزرگترین عدد در آرایه $< a_{n/2+1}, ..., a_n >$ است که به صورت بازگشتی می توان این عدد را پیدا کرد.

زمان اجرای الگوریتم از رابطه بازگشتی $T(n) = T(n_{/2}) + O(1)$ پیروی می‌کند که با استفاده از قضیه اصلی در میابیم که زمان اجرای الگوریتم $O(logn)$ می باشد.

۲. آ) پیدا کردن $ceil(x)$:

ابتدا مقدار ceil را با ۱- مقدار دهی می کنیم. برای پیدا کردن $ceil(x)$ در آرایه‌ی $< a_1, ..., a_n >$، عنصر $a_{n/2}$ را مورد بررسی قرار می دهیم. اگر x برابر با این مقدار بود، $ceil(x)$ همان $a_{n/2}$ است و مقدار ceil را برابر با $a_{n/2}$ قرار می دهیم و الگوریتم به پایان می رسد. اگر $x < a_{n/2}$ بود، $ceil(x)$ در زیر آرایه $< a_1, ..., a_{n/2} >$ قرار دارد. مقدار ceil را برابر با $a_{n/2}$ قرار می دهیم و همین الگوریتم را برای زیرآرایه $< a1, ..., a_{n/2-1} >$ اجرا می کنیم و چون اعداد موجود در این آرایه نیز به صورت مرتب شده هستند، پس شبیه مساله اصلی می باشد و به صورت بازگشتی می توان این زیرمساله را شبیه مساله اصلی حل کرد. و اگر $x > a_{n/2}$ بود باید به دنبال $ceil(x)$ در زیر آرایه $< a_{n/2+1}, ..., a_n >$ گشت که به طریق مشابه می توان این کار را انجام داد.

ب) پیدا کردن $floor(x)$:

ابتدا مقدار floor را با ۱- مقدار دهی می کنیم . برای پیدا کردن $floor(x)$ در آرایه‌ی $< a_1, ..., a_n >$، عنصر $a_{n/2}$ را مورد بررسی قرار می دهیم. اگر x برابر با این مقدار بود، $floor(x)$ همان $a_{n/2}$ است و مقدار floor را برابر با $a_{n/2}$ قرار می دهیم و الگوریتم به پایان می رسد. اگر $x < a_{n/2}$ بود، $floor(x)$ در زیر آرایه $< a_1, ..., a_{n/2-1} >$ قرار دارد. همین

الگوریتم را برای زیرآرایه $< a1, ..., a_{n/2-1} >$ اجرا می کنیم و چون اعداد موجود در این آرایه نیز به صورت مرتب شده هستند، پس شبیه مساله اصلی می باشد و به صورت بازگشتی می توان این زیرمساله را شبیه مساله اصلی حل کرد. و اگر $x > a_{n/2}$ بود مقدار floor را برابر با $a_{n/2}$ قرار می دهیم و همین الگوریتم را برروی زیر آرایه $< a_{n/2+1}, ..., a_n >$ اجرا می کنیم و دوباره چون این زیرمساله شبیه مساله اصلی است پس به صورت بازگشتی می‌توان جواب آن را محاسبه کرد.

زمان اجرای الگوریتم برای هر دو حالت، از رابطه بازگشتی $T(n) = T(n/2) + O(1)$ پیروی می‌کند که با استفاده از قضیه اصلی در میابیم که زمان اجرای الگوریتم $O(logn)$ می باشد.

۳. این مساله، مساله پیدا کردن inversion numbers است. اگر رشته P را در نظر بگیریم و به حروف آن اعداد ۱ تا n را نسبت دهیم، از آنجایی که رشته P نامرتب شده رشته S است بنابراین می‌توان بین حروف این دو رشته یک تناظر یک به یک برقرار کرد و عدد نسبت داده شده به هر حرف رشته P را به حرف متناظر در رشته S نسبت داد. از آنجا که در یک آرایه از اعداد که inversion داریم حتما دو عدد متوالی وجود دارند که عنصر سمت چپ از عنصر سمت راست برزگتر است (در غیر این صورت اگر تمام دو عدد های متوالی به گونه ای باشند که عدد سمت چپ کوچکتر از عدد سمت راست باشد در واقع اعداد به صورت مرتب شده خواهند بود و inversion نخواهیم داشت.)، بنابراین swap کردن حروف متناظر این دوو عدد در رشته S یک inversion را برطرف می‌کند. بنابراین تعداد swap های لازم برای تبدیل این دو رشته، همان تعداد inversion ها در آرایه اعداد متناظر با رشته P است که با اشتفاده از روش تقسیم و غلبه با هزینه زمانی $O(nlogn)$ می توان جواب را محاسبه کرد.

۴. فرض کنیم که اعداد نوشته شده روی هر کارت یک آرایه‌ای از اعداد به صورت $< a_1, ..., a_n >$، را تشکیل می‌دهند. در واقع ما به دنبال عددی از بین این اعداد هستیم که بیش از $n/2$ بار تکرار شده است. می دانیم که در این آرایه نمی توانیم دو عدد مختلف داشته باشیم که هردو بیش از $n/2$ بار تکرار شده باشند.

آ) شبیه merge sort عمل می کنیم و در هر مرحله آرایه ورودی را به دو قسمت می شکنیم و تابع را روی ورودی جدید صدا می زنیم تا زمانی که به یک آرایه با یک عنصر برسیم.در یک آرایه با یک عنصر، همان عنصر جواب می باشد و به عنوان جواب برگردانده می شود. در هنگام merge کردن دو زیرآرایه نیز چهار حالت پیش می آید که به صورت زیر عمل می کنیم:

حالت اول: هیچ کدام از دو زیر آرایه عنصری ندارند که بیش از $n/2$ بار تکرار شده باشد و آرایه حاصل از merge شدن این دو زیر آرایه نیز چنین عنصری نخواهد داشت.

حالت دوم: زیرآرایه سمت چپ چنین عنصری دارد ولی زیر آرایه سمت راست ندارد. در این حالت روی تمامی عناصر هر دو زیر آرایه پیمایش انجام میدهیم و تعداد تکرار عنصری را که به عنوان عنصر پر تکرار در زیرآرایه سمت چپ بود را به دست می آوریم. اگر تعداد تکرار این عنصر بزرگتر از نصف طول آرایه حاصل از merge شدن بود، مقدار آن عنصر به عنوان عنصر پر تکرار برگردانده می شود. در غیز این صورت عنصر پر تکرار نداریم.

حالت سوم: زیرآرایه سمت راست چنین عنصری دارد ولی زیر آرایه سمت چپ ندارد که شبیه حالت دوم عمل می کنیم.

حالت چهارم: هر دو زیر آرایه عنصر پر تکرار دارند که در این صورت تعداد تکرار هر دو عنصر را در

آرایه حاصل از merge شدن دو زیر آرایه به دست می آوریم و اگر یکی از آن ها تعداد تکرارش بیشتر از نصف طول آرایه حاصل از merge شدن بود، به عنوان عنصر پرتکرار برگردانده میشود(می دانیم که یک آرایه حداکثر یک عنصر پر تکرار دارد) و در غیر این صورت عنصر پر تکرار نداریم.

* توجه کنید که منظور از عنصر پر تکرار، عنصری است که بیشتر از $n/2$ بار در یک آرایه به طول n تکرار شده است.

زمان اجرای الگوریتم از رابطه بازگشتی $T(n) = 2T(n/2) + O(n)$ پیروی می‌کند که با استفاده از قضیه اصلی در میابیم که زمان اجرای الگوریتم $O(nlogn)$ می باشد.

ب) فرض کنیم که دو متغیر majority-index و count داریم که به ترتیب با مقدار های 0 و 1 مقدار دهی شده اند. روی عناصر آرایه و با شروع از عنصر دوم(a_2) شروع به پیمایش می کنیم و به هر عنصر با اندیس i که می رسیم مقدار آن عنصر را با مقدار $a_{majority-index}$ مقایسه می کنیم. اگر با هم برابر بودند مقدار count را یک واحد زیاد می کنیم و به سراغ عنصر بعدی می رویم و در غیر اینصورت مقدار count را یک واحد کم می کنیم و اگر مقدار count صفر شد، majority-index را برابر با i و مقدار count را برابر با 1 می کنیم. بعد از اتمام این پیمایش، تعداد تکرار $a_{majority-index}$ را در آرایه اعدادی که داریم، می شماریم. اگر این مقدار بزرگتر از $n/2$ بود به عنوان عنصر پرتکرار شناخته می شود و در غیر این صورت عنصر پرتکرار نداریم.

۵. اگر بانوی شماره i را با a_i و همسرش را با b_i نشان دهیم یک آرایه به صورت $< a_1, a_2, ..., a_n, b_1, b_2, ..., b_n >$ خواهیم داشت که می خواهیم این آرایه را به صورت $< a_1, b_1, 1_2, b_2, ..., a_n, b_n >$ در آوریم. با استفاده از ایده حل و تقسیم عمل می کنیم و در هر مرحله آرایه ورودی را از وسط نصف می کنیم(به عنوان مثال به دو زیرآرایه left و right تقسیم می کنیم) و عناصر نیمه دوم left را با عناصر نیمه اول right، به ترتیب عوض می کنیم(swap). حال چون هر کدام از زیرآرایه های left و right شبیه آرایه اولیه می باشند می توانیم همین کار را به صورت بازگشتی روی هر کدام از آن ها انجام دهیم تا موقعی که طول آرایه ورودی دو شود.

* توجه شود که چون تعداد زوج ها به صورت توانی از دو می باشد، در هر مرحله که آرایه را نصف می کنیم طول هر دو زیرآرایه چپ و راست یکسان خواهد بود.

زمان اجرای الگوریتم از رابطه بازگشتی $T(n) = 2T(n/2) + O(n)$ پیروی می‌کند که با استفاده از قضیه اصلی در میابیم که زمان اجرای الگوریتم $O(nlogn)$ می باشد.

۶. آ) از ایده تقسیم وغلبه استفاده می کنیم. بدین صورت که در هر مرحله آرایه ورودی را نصف می کنیم تا زمانی که به یک آرایه به طول یک برسیم. فرض کنیم هر بار که به صورت بازگشتی تابع را روی یک آرایه صدا می زنیم سه مقدار i, j, sum را بر می گرداند. حال در حالتی که طول آرایه ورودی تابع یک می باشد، i, j برابر اندیس این عنصر در آرایه اصلی و sum برابر با مقدار این عنصر می باشد. برای merge کردن دو آرایه نیز به صورت زیر عمل می کنیم(نماد l برای زیرارایه سمت چپ و نماد r برای زیرارایه سمت راست استفاده شده است.):

اگر sumL و sumR هر دو منفی بودند، در این صورت هر کدام که بزرگتر بود به همراه i, j مربوط به آن sum به عنوان جواب حاصل از merge شدن باز گردانده می شود.

sumR منفی و sumL نامنفی با شد. در این صورت sumL iL, jL به عنوان جواب حاصل از merge شدن بازگردانده می شود.

sumR, sumL هر دو مثبت باشند. در این صورت ابتدا در زیرآرایه سمت چپ، بیشینه مقدار جمع را با شروع از mid بدست می آوریم.فرض کنیم بیشینه جمع در بازه $[newI, mid]$ رخ دهد و مقدار آن برابر با newSumL باشد. سپس در زیرآرایه سمت راست، بیشینه مقدار جمع با شروع از $mid+1$ را بدست می آوریم و به طریق مشابه مقدار های $[mid+1, newJ], newSumR$ را بدست می آوریم.حال داریم:

اگر sumR بزرگتر از sumL و $newSumR + newSumL$ بود مقدار های sumR, iR, jR را به عنوان جواب حاصل از merge شدن بر میگردانیم.

اگر sumL بزرگتر از sumR و $newSumR + newSumL$ بود مقدار های sumL, iL, jL را به عنوان جواب حاصل از merge شدن بر میگردانیم.

اگر $newSumR + newSumL$ بزرگتر از sumR و sumL بود مقدار های $newSumL + newSumR, newI, newJ$ را به عنوان جواب حاصل از merge شدن بر میگردانیم.

زمان اجرای الگوریتم از رابطه بازگشتی $T(n) = 2T(n/_2)+O(n)$ پیروی می‌کند که با استفاده از قضیه اصلی در میابیم که زمان اجرای الگوریتم $O(nlogn)$ می باشد.

ب از ایده تقسیم و غلبه برای حل مساله استفاده می کنیم بدین صورت که در هر مرحله بزرگترین زیررشته، بزرگترین زیررشته prefixe (زیررشته ای که از عنصر اول شروع می شود) و بزرگترین زیررشته postfix (زیررشته ای که با عنصر آخر تمام می شود) را به دست می آوریم. حال به صورت بازگشتی عمل می کنیم.

دنباله را از وسط به دو قسمت تبدیل می کنیم و به صورت بازگشتی الگوریتم را اجرا می کنیم:

سه زیررشته ماکزیمم کلی و prefix ، postfix را به صورت زیر حساب می کنیم:

$$maxPrefix = maxPrefix(left) if maxPrefix(left) != left, maxPrefix(left) + maxPrefix(right) otherwise$$

$$maxPostfix = maxPostfix(right) if maxPostfix(right) != right, maxPostfix(right) + maxPostfix(left) otherwise$$

$$max = maximum(maxLeft, maxRight, maxPrefix, maxPostfix, maxPostfix(left) + maxPrefix(right)$$

* توجه کنید که مقدار های i، j بسته به اینکه کدام یک از حالت ها اتفاق می افتد مشابه قسمت قبل محاسبه شده و برگردانده می شوند.

زمان اجرای الگوریتم از رابطه بازگشتی $T(n) = 2T(n/_2)+O(1)$ پیروی می‌کند که با استفاده از قضیه اصلی در میابیم که زمان اجرای الگوریتم $O(n)$ می باشد.