به نام خدا

ساختمان داده‌ها و الگوریتم‌ها (۴۰۲۵۴)
دانشگاه صنعتی شریف
مدرس: مهدی صفرنژاد

تمرین دوم
انتشار: ۲۵ مهر ۱۴۰۰

مرتب‌سازی، روابط بازگشتی و داده‌ساختارهای پایه

سؤالات را با دقت بخوانید و روی همه آن‌ها وقت بگذارید. تمرین‌های تئوری تحویل گرفته نمی‌شوند اما از آن‌ها سؤالات کوییز مشخص می‌شود، بنابراین روی سؤالات به خوبی فکر کنید و در کلاس‌های حل تمرین مربوطه شرکت کنید.

سؤال ۱. در یک آرایه، یک نابه‌جایی برابر یک جفت جایگاه است که عدد داخل جایگاه اولی بزرگتر از عدد داخل جایگاه دومی باشد. برای مثال در یک آرایه مرتب‌شده تعداد نابه‌جایی‌ها برابر صفر است و در یک آرایه مرتب‌شده برعکس این تعداد برابر $\binom{n}{2}$ است.

آ. ثابت کنید اگر تعداد نابه‌جایی های یک آرایه d باشد و الگوریتم insertion sort را روی آن اجرا کنیم با مرتبه زمانی $O(n+d)$ مرتب‌سازی انجام می‌شود.

ب. فرض کنید یک آرایه نامرتب به ما داده‌اند به طوری‌که هر کس با جایگاه اصلی آن حداکثر ۱۰ جایگاه فاصله دارد. برای مثال عدد کمینه حداکثر در اندیس ۱۰ است. (آرایه از اندیس صفر شروع می‌شود.) با ارائه روش مناسب برای مرتب‌سازی ثابت کنید این کار در مرتبه زمانی $O(n)$ ممکن است.

پاسخ:

در Insertion sort یک حلقه n تایی داریم و به ازای هر اندیس به اندازه تعداد اعداد بزرگتر از خودش که قبل از او ظاهر شده‌اند پیمایش می‌کنیم که این تعداد دقیقا برابر نابه‌جایی هایی است که این اندیس می‌سازد. پس در مجموع به ازای هر اندیس این مقادیر را اگر جمع کنیم به ما مجموع تعداد نابه‌جایی ها را می‌دهد.

حالا برای بخش دوم ثابت می‌کنیم تعداد نابه‌جایی های دنباله‌ای با این شرط $O(n)$ است. برای اینکار صرفا کافیست بگوییم اگر عددی مثل $a > b$ قبل از b ظاهر شده باشد. $a - b \leq 20$ چرا که در غیر این‌صورت یکی از a یا b با جایگاه اصلی‌شان بیش از ۱۰ واحد فاصله دارند. بنابراین تعداد نابه‌جایی‌ها حداکثر $20n$ است و طبق بخش الف با اجرای مرتب‌سازی درجی الگوریتمی از $O(n+20n) = O(n)$ داریم.

سؤال ۲. الگوریتمی را توصیف کنید که روی n عنصر از ۱ تا k که به عنوان ورودی دریافت می‌کند، پیش‌پردازشی از $O(n+k)$ انجام می‌دهد و سپس می‌تواند هر پرسش مربوط به اینکه چه تعداد از این n عنصر در بازه $[a,b]$ قرار می‌گیرند را در $O(1)$ پاسخ دهد.

پاسخ:

برای این سوال سه آرایه A, B, C را نگه می‌داریم که در آن:

۱. آرایه A شامل n عضو است و $A[i]$ عدد i ام ورودی را نگه می‌دارد.

۲. آرایه B شامل k عضو است و $B[i]$ مشخص می‌کند که عدد i چند بار در ورودی تکرار می‌شود (برای پر کردن این آرایه همانند counting sort عمل می‌کنیم و در $O(n)$ انجام شدنی است).

۳. آرایه C شامل k عضو است و $C[i]$ برابر است با (مقدار تجمعی آرایه B تا خانه i) $B[i] + C[i-1]$. (و مقدار $(C[0] = B[0]$

در نهایت جواب هر کوئری در $O(1)$ را رابطه $C[b] - C[a] + B[a]$ بدست می‌آید.

سؤال ۳. داده ساختاری طراحی کنید که اعمال زیر را به صورت بهینه انجام دهد:

آ. یک عدد را به انتهای لیست اضافه کند.

ب. یک عدد را از انتهای لیست کم کند.

پ. k عنصر انتهایی لیست را قرینه کند که k برای داده ساختار همیشه یک عدد ثابت است (در عمل نتیجه مثل این است که k عنصر را به ترتیب بخواند و روی هم بریزد، سپس آن‌ها را وارونه کند و سپس به لیست برگرداند.)

ت. عناصر را به ترتیبی که در لیست قرار دارند چاپ کند.

پاسخ:

یک لیست‌پیوندی دوطرفه(به صورتی که هر node در لیست، قبلی و بعدی خود را نگه می‌دارد.) و یک پشته در نظر می‌گیریم. برای درج یک عدد به لینکدلیست، عدد را به آخرین node اضافه می‌کنیم و اولین node را حذف کرده و مقدارش را در پشته push می‌کنیم.

برای حذف عدد، node نهایی در لیست را حذف می‌کنیم، سپس از پشته pop می‌کنیم و عدد خروجی را در ابتدای لینکدلیست قرار می‌دهیم.

برای وارونه کردن جایگاه k عنصر نهایی لیست، کافی است تا مقادیر next و previous مربوط به هر node را با هم‌دیگر جابجا کنیم تا لیست‌پیوندی‌مان وارونه شود.

همچنین برای چاپ عناصر به ترتیب، می‌توان با تعریف یک پشته کمکی، هر عنصر از پشته اول را ابتدا pop کرده و سپس در پشته کمکی push کنیم تا پشته‌ای با ترتیب عکس داشته باشیم، سپس همین روند را دوباره تکرار کرده و اعداد را به پشته اول برگردانیم، با این تفاوت که این بار هر عددی که از پشته کمکی pop می‌شود، قبل از برگشتن به پشته اول، چاپ نیز خواهد شد. سپس با شروع از ابتدای لیست‌پیوندی و پیمایش آن می‌توان عناصر را به ترتیبی که در لیست هستند در خروجی نوشت.

سؤال ۴. داده ساختاری طراحی کنید که بتواند اعمال $Push$، Pop و $FindMin$ (یافتن و برگرداندن کوچک‌ترین عنصر) را در $O(1)$ انجام دهد. سپس با فرض اینکه می‌دانیم نمی‌توان یک آرایه را در حالت کلی در بهتر از $O(n \log n)$ مرتب کرد، ثابت کنید اگر این داده ساختار بخواهد عمل $DeleteMin$ را هم پشتیبانی کند، نمی‌تواند آن را هم در مرتبه $O(1)$ انجام دهد.

پاسخ: دو پشته در نظر بگیرید. در پشته اول به صورت معمولی از push و pop استفاده می‌کنیم. در پشته دوم، عمل push را تنها در صورتی انجام می‌دهیم که عنصر در حال درج، از عنصر بالای پشته کوچک‌تر باشد. همچنین هنگام صدا زده‌شدن دستور pop اگر عنصر بالای پشته اول، برابر با عنصر بالای پشته دوم(که همواره مینیموم فعلی اعداد موجود می‌باشد) بود، در پشته دوم هم pop انجام می‌دهیم و در غیر این‌صورت فقط عدد بالای پشته اول pop می‌شود. در رابطه با حذف عنصر در مرتبه زمانی ثابت، به این موضوع توجه کنید که در صورت امکان این امر، می‌توان با درج همه عناصر یک آرایه در این داده‌ساختار و اجرای DeleteMin تا زمانی که داده‌ساختار خالی شود، آرایه را در زمان خطی مرتب کرد که این کار غیرممکن می‌باشد!

سؤال ۵. n نفر قصد دارند تا به یک اتاق وارد شده و پس از مدتی از آن خارج شوند. هر نفر i در زمان a_i وارد اتاق شده و در زمان b_i از آن خارج می‌شود.(به ازای هر i می‌دانیم که $b_i > a_i$ می‌باشد) و همه a_i و b_i ها متمایز هستند. در ابتدای روز چراغ های اتاق خاموش اند و اولین نفری که وارد اتاق می‌شود آن ها را روشن می‌کند. به منظور صرفه‌جویی در مصرف برق اگر نفر i ام،اتاق را ترک کند ولی کسی در اتاق نباشد، باید زمان ترک اتاق چراغ‌ها را خاموش کند. الگوریتمی از مرتبه زمانی $O(n \log n)$ ارائه دهید که تعداد دفعات روشن شدن چراغ‌ها را به دست بیاورد.

پاسخ: فرض کنید که لیست L شامل زمان های ورود و خروج را داریم (یک لیست شامل ۲n عنصر). ابتدا این لیست را به کمک یکی از الگوریتم های مرتب‌سازی (به طور مثال مرتب‌سازی ادغامی) به صورت ادغامی مرتب می‌کنیم. حال دو متغیر cntPeople و cntLight را به منظور نگه‌داری تعداد افراد داخل اتاق و تعداد بار های خاموش/روشن شدن چراغ های اتاق تعریف می‌کنیم. حال با پیمایش لیست L، اگر به زمان a_i برسیم ابتدا متغیر cntPeople را بررسی کرده، اگر برابر صفر بود متغیر cntLight را یک واحد افزایش می‌دهیم. پس از آن مقدار cntPeople را یک واحد افزایش می‌دهیم. اگر به b_i رسیدیم، یک واحد از cntPeople کم می‌کنیم. پس از تمام شدن پیمایش لیست، مقدار cntLight را به عنوان خواسته مساله ارائه می‌کنیم. چون پیمایش لیست از $O(n)$ و سورت لیست از $O(n \log n)$ است، زمان کلی الگوریتم $O(n \log n)$ خواهد بود.

سؤال ۶. برای هر یک از موارد زیر، الگوریتم مرتب سازی مناسب (از بین مرتب سازی انتخابی ، مرتب سازی درجی یا مرتب سازی ادغامی) را انتخاب کنید و انتخاب خود را توجیه کنید.

آ. فرض کنید یک داده ساختار D به شما داده می شود و از دو عملیات توالی استاندارد پشتیبانی می کند: $D.get\_at(i)$ در بدترین حالت از مرتبه $\Theta(1)$ و $D.set\_at(i, x)$ در بدترین حالت از مرتبه $\Theta(n \log n)$ است. الگوریتمی برای مرتب سازی داده های D به صورت درجا (in-place) انتخاب کنید.

ب. فرض کنید آرایه مرتب شده A شامل n عدد صحیح است که هر یک از آنها در یک کلمه ماشین قرار می گیرد. حال فرض کنید شخصی تعدادی جابه جایی از مرتبه $loglogn$ بین جفت آیتم های مجاور در A انجام می دهد تا A دیگر مرتب شده نباشد. الگوریتمی را برای مرتب سازی مجدد اعداد صحیح در A انتخاب کنید.

پاسخ: قسمت الف به الگوریتم مرتب سازی درجا نیاز دارد ، بنابراین نمی توانیم مرتب سازی ادغامی را انتخاب کنیم، زیرا مرتب سازی ادغامی درجا نیست. مرتب سازی درجی $O(n^2)$ تا عمل get_at و set_at انجام می دهد، بنابراین با این ساختار داده $O(n^3.logn)$ زمان می برد.

از طرفی مرتب سازی انتخابی $O(n^2)$ عمل get_at انجام می دهد اما فقط $O(n)$ عمل set_at انجام می دهد. پس مرتب سازی با این ساختار داده، حداکثر $O(n^2.logn)$ زمان می برد و ما مرتب سازی انتخابی را قبول می‌کنیم.

قسمت ب عملکرد مرتب سازی انتخابی و ادغامی به ورودی بستگی ندارد. آنها بدون در نظر گرفتن ورودی در زمان $\theta(n^2)$ و $\theta(nlogn)$ اجرا می شوند. از طرف دیگر ، مرتب سازی درجی می تواند از حلقه داخلی زود خارج شود ، بنابراین می تواند در زمان $O(n)$ بر روی برخی ورودی ها اجرا شود. برای اثبات اینکه مرتب سازی درجی در زمان $O(n)$ برای ورودی های ارائه شده اجرا می شود ، توجه داشته باشید که انجام یک جابه جایی بین عناصر مجاور می تواند تعداد نابه جایی ها را در آرایه حداکثر یک واحد تغییر دهد. از طرفی، هر بار که مرتب سازی درجی دو مورد را در حلقه داخلی جابه جا می کند ، یک نابه جایی را برطرف می کند. بنابراین ، اگر یک آرایه k جابه جایی از آرایه مرتب شده باشد فاصله داشته باشد، مرتب سازی درجی در زمان $O(n+k)$ اجرا می شود. برای این سوال، از آنجا که $k = loglogn = O(n)$ ، مرتب سازی درجی در زمان $O(n)$ اجرا می شود ، بنابراین ما مرتب سازی درجی را انتخاب می کنیم.

سؤال ۷. مواردی وجود دارد که از ما خواسته می شود داده هایی را که تقریبا مرتب شده اند مرتب کنیم. در یک آرایه kــ مرتب شده هیچ عنصری بیشتر از k از موقعیت آن در آرایه مرتب شده فاصله ندارد. در سوالات زیر، A یک آرایه kــ مرتب شده با $k \ll n$ است:

آ. زمان اجرای Insertion-Sort و Merge-Sort در A چگونه است؟

ب. زمان اجرای Bubble-Sort در A چقدر است (به شبه کد زیر مراجعه کنید)؟

```
1  Bubble-Sort(A)
2      sorted ← false
3      while sorted = false
4          sorted ← true
5          for i ← 1 to length[A] -1
6              if A[i] > A[i + 1]
7                  swap A[i] and A[i + 1]
8                  sorted ← false
```

پاسخ: مرتب سازی درجی: $O(kn)$. یک "وارونگی" را در A به عنوان یک جفت (i, j) تعریف کنید که $i < j$ و $A[i] > A[j]$ است. اگر هر عنصر در k فاصله از موقعیت درست خود قرار داشته باشد، حداکثر $O(kn)$ وارونگی وجود دارد. بنابراین زمان اجرا برای مرتب سازی درجی $O(n + I) = O(kn)$ است، که I تعداد وارونگی ها در A است. دلیل این است که I دقیقاً تعداد عملیات جابجایی است که در مرتب سازی درجی به آن نیاز داریم. برای حلقه داخلی الگوریتم، هر عنصر $A[i]$ چند برابر li که li تعداد عناصر j است به طوری که $j < i, A[j] > A[i]$.

$$number\,of\,shifting\,operations\,needed = \sum_{i=1}^{n} l_i \tag{۱}$$

$$= \sum_{i=1}^{n} |\{(j,i)|(j,i)\,is\,an\,inversion\,in\,A, \forall j\}| \tag{۲}$$

$$= total\,number\,of\,inversions\,in\,A \tag{۳}$$

اکنون حلقه خارجی الگوریتم مرتب سازی درجی را در نظر بگیرید. n بار تکرار می شود زیرا اندیس در طول آرایه حرکت می کند و مستقل از تعداد وارونگی است. بنابراین کل زمان اجرای مرتب سازی درجی $O(I + n) = O(kn)$ است.

مرتب سازی ادغامی: $\Theta(nlogn)$. همان تابع بازگشتی. $T(n) = 2T(n/2) + \Theta(n)$.

برای بخش دوم سوال ایده الگوریتم مرتب سازی حبابی این است که از چپ شروع کنید، موارد مجاور را مقایسه کنید و مورد بزرگتر را به سمت راست "حباب بزنید" تا زمانی که همه موارد به موقعیت مناسب خود برسند. از بخش قبل، ما می دانیم که حداکثر $O(kn)$ وارونگی وجود دارد. پس از هر عملیات جابه جایی در خط ۶ ، تعداد وارونگی یکی کاهش می یابد. بنابراین خط ۶ حداکثر در زمان $O(kn)$ اجرا می شود ، که گلوگاه زمان اجرا است.

سؤال ۸. الگوریتم مرتب سازی پایدار الگوریتمی است که در آن عناصر با ارزش مساوی در آرایه خروجی (مرتب شده) به همان ترتیب در آرایه ورودی ظاهر می شوند. کدام یک از Insertion-Sort ، Merge-Sort و Bubble-Sort پایدار هستند؟

پاسخ: مرتب سازی درجی: پایدار. ما می توانیم آن را با استفاده از متغیر حلقه زیر ثابت کنیم: در ابتدای هر عمل در حلقه for اگر $A[a] = A[b], a < b \leq j-1$ و متمایز باشد ، پس $A[a]$ قبل از $A[b]$ در آرایه اولیه ظاهر می‌شود.

مرتب سازی ادغامی: پایدار. می توانیم آن را با استقرا ثابت کنیم.

حالت پایه: وقتی Merge-Sort را با اندیس p و r فراخوانی می کنیم به طوری که $p \geq r$ بنابراین $(p == r)$ ، همان آرایه را برمی گردانیم. بنابراین فراخوانی Merge-Sort بر روی یک آرایه با اندازه یک ، همان آرایه را باز می گرداند که پایدار است.

استقراء: فرض می کنیم که فراخوانی Merge-Sort در یک آرایه با اندازه کمتر از n یک آرایه مرتب شده پایدار را برمی گرداند. سپس نشان می دهیم که اگر Merge-Sort را بر روی یک آرایه با اندازه n فراخوانی کنیم ، یک آرایه مرتب شده پایدار برمی گردانیم. هر فراخوانی Merge-Sort شامل دو فراخوانی به MergeSort در نیم آرایه است و علاوه بر این ، این دو زیر مجموعه را ادغام کرده و برمی گرداند. از آنجا که فرض می کنیم که فراخوانی های Merge-Sort در آرایه های کوچکتر آرایه های مرتب شده پایدار را برمی گرداند ، ما فقط باید نشان دهیم که مرحله ادغام در دو آرایه مرتب شده با ثبات یک آرایه پایدار را برمی گرداند. $A[i] = A[j], i < j$ در آرایه اولیه ، به $f(i) < f(j)$ در آرایه جدید نیاز داریم ، جایی که f تابعی است که موقعیت های جدید را در آرایه مرتب شده می دهد. اگر i و j در نیمی از فراخوانی Merge-Sort قرار داشته باشند، در صورت فرض، آن را حفظ می کند. اگر i و j در زیر آرایه های مختلف قرار بگیرند ، پس می دانیم که i در زیر آرایه چپ و j در زیر زیر آرایه راست از $i < j$ قرار دارد. در مرحله ادغام ، عناصر را از زیر آرایه چپ می گیریم در حالی که کمتر یا مساوی عنصر زیر آرایه راست است (خط ۱۳). بنابراین ، ما عنصر i را قبل از گرفتن عناصر j و $f(i) < f(j)$ ، ادعایی که سعی در اثبات آن داریم، می گیریم.

مرتب سازی حبابی: پایدار. اثبات اصلی از طریق استقراء است. نکته کلیدی این است که عناصر با ارزش مساوی عوض نمی شوند (خط ۵ در شبه کد در مسئله قبل (ب)).

سؤال ۹. روابط بازگشتی زیر را تحلیل کنید. فرض کنید $T(1) = \Theta(1)$.

آ. $T(n) = 3\,.\,T(n/\sqrt{2}) + O(n^4)$

پاسخ:

الف. ما می توانیم با انتخاب $f(n) = \Theta(n^4)$ حد بالای $T(n)$ را انتخاب کنیم. با توجه به حالت (۳) قضیه اساسی می دانیم که $T(n) = O(n^4)$، زیرا $\log_b a = \log_{\sqrt{2}} 3 = 2\log 3$ و $\Theta(n^4) \subset \Omega(n^{2lg3+\epsilon})$ برای $\epsilon \leq (4 - 2\log 3)$ و $3(n/\sqrt{2})^4 = 3/4n^4 < cn^4$ برای هر $3/4 < c < 1$ است.

ب. $T(n) = 4\,.\,T(n/2) + n^2\sqrt{n}$

پاسخ:

ب. از مورد ۳ قضیه اصلی استفاده می کنیم، زیرا $f(n) = n^2\sqrt{n} = n^{2.5}$ و $n^{\log_b a} = n^{log_2 4} = n^2$. می توانیم $\epsilon = 0.1$ را برای برآوردن شرایط قضیه انتخاب کنیم. علاوه بر این اگر $c = 0.9$ باشد، می توانیم تأیید کنیم که $4(n/2)^{2.5} \leq cn^{2.5}$ بنابراین شرایط مورد ۳ در نظر گرفته شده است و نتیجه می گیریم که $T(n) = \Theta(n^2\sqrt{n})$.

پ. $T(n) = 14\,.\,T(n/3) + n^2\,ln\,n$

پاسخ:

پ. با استفاده از قضیه اصلی، $a = 14, b = 3$ و $n^{\log_b a} = n^{\log_3 14}$ بدست می آید و $f(n) = n^2 ln\,n = O(n^{\log_3 14 - \epsilon})$ که ضریب ϵ برابر است با:

$$\frac{n^2 ln\,n}{n^{\log_3 14}} = \frac{ln\,n}{n^{\log_3 \frac{14}{9}}} = O(n^{-\epsilon})$$

بنابراین ، با مورد ۱ قضیه اصلی داریم: $T(n) = \Theta(n^{\log_3 14})$.

ت. $T(n) = 161^2\,.\,T(\sqrt[161]{n}) + 161.(log\,n)^2$

پاسخ:

ت. $k = n^2$ را تعریف می کنیم و $G(k) = T(n)$. پس $G(k) = 161^2.G(k/161) + 161k^2$. از آنجا که $\log_{161}(161^2) = 2$ است ، توسط قضیه اساسی (مورد ۲) نتیجه $G(k) = \Theta(k^2\,logk)$ است.

بنابراین $T(n) = G(k) = \Theta(k^2\,logk) = \Theta(log^2 n\,loglogn)$.

ث. $T(n) = T(\lfloor\sqrt[4]{n}\rfloor) + T(\lceil\sqrt{n}\rceil) + log\,n$

پاسخ:

ث. عملگرهای کف/سقف را نادیده می‌گیریم (با آنها به عنوان اعداد صحیح برخورد کنید). سپس ، با جایگزینی $m = logn$ ، رابطه بازگشتی $T(2^m) = T(2^{m/4}) + T(2^{m/2}) + m$ را بدست می آوریم. با استفاده از $S(m) = T(2^m)$ ، این مقدار $S(m) = S(m/4) + S(m/2) + m$ می شود که پاسخ آن با استفاده از درخت بازگشتی $S(m) = \Theta(m)$ می‌شود، بنابراین ، $T(n) = T(2^m) = S(m) = (m) = \Theta(logn)$

ج. $T(n) = 3 \,.\, T(n/3 + 5) + n/2$

پاسخ:

ج. سوال را با حدس و بررسی حل می کنیم. با استقرا ، $T(n)$ یک تابع یکنواخت در حال افزایش است. بنابراین ، برای حد پایین، از آنجا که $T(n/3) \leq T(n/3+5)$ ، با استفاده از مورد ۲ قضیه اصلی ، ما $T(n) = \Omega(nlogn)$ داریم. برای حد بالا ، می توانیم $T(n) = O(nlogn)$ را نشان دهیم.

برای پایه استقرا $\forall n \leq 30$، می توانیم d۱ را به اندازه کافی بزرگ انتخاب کنیم به طوری که $T(n) < d1nlogn$.

برای مرحله استقرا ، برای همه $k < n$ فرض کنید که $T(n) < d1nlogn$. سپس برای $k = n > 30$ ، داریم $n/3 + 5 < n - 15$ ، پس:

$$
\begin{aligned}
T(n) &= 3T(n/3+5) + n/2 && (۴)\\
&< 3T(n-15) + n/2 && (۵)\\
&< 3(3T((n-15)/3+5) + (n-15)/2) + n/2 && (۶)\\
&< 9T(n/3) + 3(n-15)/2 + n/2 && (۷)\\
&< 9d_1(n/3)log(n/3) + 2n - 45/2 && (۸)\\
&< 3d_1n\,log(n/3) + 2n - 45/2 && (۹)\\
&< 3d_1n\,logn && (۱۰)
\end{aligned}
$$

آخرین خط برای $n \geq 1$ و $d_1 > 2/3\,log3$ صادق است. بنابراین ما می توانیم با استقرا اثبات کنیم که برای همه n ها $T(n) < cnlogn$، که $c = max\{d_1, 3d_1, 2/log3\}$ است. بنابراین ، $T(n) = O(nlogn)$ و $T(n) = \Theta(nlogn)$.

سؤال ۱۰. با داشتن پشته‌ای که یک کاراکتر را در هر گره ذخیره می‌کند، الگوریتمی از مرتبه زمانی خطی (با فرض زمان ثابت برای pop ، push و size) طراحی کنید که بررسی می‌کند آیا کاراکترها یک palindrome را تشکیل می‌دهند یا خیر. ممکن است از یک پشته دوم استفاده کنید ، اما از هیچ داده‌ساختار دیگری استفاده نکنید. نیازی به حفظ ورودی ندارید.

پاسخ: ابتدا عناصر $n//2$ اول را پوش می‌کنیم و آنها را به پشته دوم پوش می‌کنیم. اگر n فرد باشد ، عنصر بعدی (عنصر وسط) را بدون استفاده بیشتر از آن اضافه می کنیم. سپس به طور همزمان از هر دو دسته پاپ می‌کنیم و بررسی می کنیم که آیا این عناصر یکسان هستند:

```
def buildStack(word):
    stack = []
    for i in range(len(word)): stack.append(word[i])
    return stack

def isPalindrome(stack):
    stack2 = []
    n = len(stack)
    for i in range(n//2):
        stack2.append(stack.pop())
    if n%2==1: stack.pop()
    for i in range(n//2):
        front = stack.pop()
        back = stack2.pop()
        if not front == back: return False
    return True
```

زمان اجرا: هر دو حلقه for دارای $O(n)$ تکرار و تعداد ثابتی عملیات زمان ثابت در هر تکرار دارد. تمام عملیات دیگر $O(1)$ زمان می برد. به طور کلی الگوریتم در زمان $O(n)$ اجرا می شود.

صحت: فرض کنید که stack در ابتدا شامل: $word[0], ..., word[n-1]$ باشد. الگوریتم ابتدا آخرین $n//2$ عنصر را ظاهر می کند و آنها را به stack۲ پوش می کند (و سپس اگر n فرد باشد عنصر بعدی را پوش می کند).

اکنون stack شامل کلمات $word[0], ..., word[n//2-1]$ و stack۲ شامل کلمات $word[n-1], ..., word[n-n//2]$ است. بنابراین، در حلقه دوم $word[i]$ با $word[n-i-1]$ به ازای $0 \leq i < n//2$ مقایسه می شود.

موفق باشید