نمونه سوالات مربوط به بخش ۵.۸ - ۵.۹

۵.۸.۱ الگوریتم مرتب‌سازی سریع را به گونه‌ای تغییر دهید که اعداد را به صورت غیر-صعودی مرتب کند.

راه‌حل:

در تابع partition، عناصر بزرگ‌تر از pivot را به سمت چپ عنصر آن منتقل می‌کنیم و عناصر کوچک‌تر از آن را به سمت راست آن.

۵.۸.۲ در الگوریتم مرتب‌سازی سریع اگر تمامی عناصر مقدار متفاوت داشته‌باشند، بزرگترین عنصر حداکثر چندبار جابه‌جا می‌شود؟

راه‌حل:

حداکثر $n-1$ بار جابه‌جا می‌شود. چرا که هر عنصر حداکثر $n-1$ بار می‌تواند جابه‌جا شود و پس از آن‌که به جای درست خود برسد دیگر جابه‌جا نمی‌شود. اگر اعداد ورودی جایگشت اعداد ۱ تا n به صورت نزولی باشد، در آن صورت بزرگ‌ترین عنصر $n-1$ بار جابه‌جا خواهدشد.

۵.۸.۳ زمان اجرای مرتب‌سازی سریع را در هنگامی که همه عناصر مقدارشان یکسان باشد، تحلیل کنید.

راه‌حل:

در این حالت در تابع partition تمامی عناصر در یک سمت عنصر pivot قرار می‌گیرند و در این حالت الگوریتم مرتب‌سازی سریع هزینه‌ای برابر $\theta\left(n^2\right)$ خواهدداشت.

۵.۸.۴ نشان دهید بهترین زمان اجرای مرتب‌سازی سریع $\Omega\left(nlogn\right)$ است.

راه‌حل:

داریم:

$$T(n)=min\left(T(q)+T(n-q-1)\right)+\theta(n)$$
$$(1\leq q\leq n-1)$$

هنگامی T کمینه خواهدبود که $q=\frac{n}{2}$. و در این حالت خواهیم داشت: $T=\Omega\left(nlogn\right)$

۵.۸.۵ نشان دهید مرتب‌سازی سریع از $O(n2)$ است.

راه‌حل:

داریم:

$$T(n)=min\left(T(q)+T(n-q-1)\right)+\theta(n)$$
$$(1\leq q\leq n-1)$$

هنگامی T بیشینه خواهد بود که $q=0$ یا $q=n-1$ باشد. پس $T=O\left(n^2\right)$

۵.۸.۶ نشان دهید مرتب‌سازی سریع از $\Omega\left(n^2\right)$ است.

راه‌حل:

به ازای جایگشتی از اعداد ۱ تا n که به صورت نزولی مرتب شده‌اند، مرتب‌سازی سریع از $\theta\left(n^2\right)$ هزینه خواهدداشت. پس مرتب‌سازی سریع از $\Omega\left(n^2\right)$ است.

۵.۸.۷ پایداری الگوریتم مرتب‌سازی سریع را بررسی کنید.

راه‌حل:

این موضوع به پیاده‌سازی تابع partition برمی‌گردد. در پیاده‌سازی مرسوم تابع partition، عنصر آخر به عنوان pivot در نظر گرفته‌می‌شود و تمامی عناصر برابر با آن در سمت چپ آن قرار خواهند گرفت. بقیه عناصری که با هم مساوی هستند نیز ترتیبشان عوض نمی‌شود. پس الگوریتم مرتب‌سازی سریع پایدار است.

۵.۸.۸ درجا بودن یا نبودن الگوریتم مرتب‌سازی سریع را بررسی کنید.

راه‌حل:

چون در طی مرتب‌سازی سریع از حافظه اضافی استفاده نمی‌شود این الگوریتم درجا است.

۵.۸.۹ چرا زمان اجرای الگوریتم‌ها را در حالت میانگین بررسی می‌کنیم؟

راه‌حل:

زیرا احتمال برخورد با بدترین حالت بسیار کم است و پخش‌کردن داده‌ها به صورت تصادفی می‌تواند این احتمال را بسیار کمتر هم بکند. از این رو ما در اکثر موارد هزینه‌ای برابر حالت میانگین برای الگوریتم‌هایمان خواهیم داشت.

۵.۸.۱۰ الگوریتم مرتب‌سازی سریع در حالت میانگین چه هزینه‌ای دارد؟

راه حل:

$O(nlogn)$

۵.۸.۱۱ فرض کنید در هر مرحله از مرتب‌سازی سریع، آرایه به دو بخش با نسبت اندازه k و $1-k$ تقسیم شود که k یک عدد ثابت با شرط $0\leq k\leq\frac{1}{2}$ است. نشان دهید حداقل عمق یک برگ در درخت بازگشت(Recursion Tree) $\frac{-\log n}{\log k}$ است.

راه حل:

حدقل ارتفاع این‌گونه به دست می‌آید که در هر مرحله قسمت به کوچک‌تر برویم. یعنی در هر مرحله اندازه هر قسمت از n به kn کاهش پیدا می‌کند. پس بعد از i مرحله تعداد عناصر برابر $k^i n$ خواهدبود. وقتی که به یک برگ می‌رسیم تعداد عناصر برابر ۱ خواهدبود. m را کوچک‌ترین مقداری در نظر بگیرید که $k^m n=1$. (یعنی اینکه به یک برگ رسیده‌ایم.) پس $k^m=1/n$ که

اگر از دو طرف لگاریتم بگیریم خواهیم داشت: $m=\frac{-\log n}{logk}$

۵.۸.۱۲ سوال قبل را در نظر بگیرید. ثابت کنید حداکثر عمق یک برگ در درخت بازگشت(Recursion Tree) $\frac{-\log n}{\log(1-k)}$ است.

راه حل:

حداکثر ارتفاع این‌گونه به دست می‌آید که در هر مرحله قسمت به بزرگ‌تر برویم. یعنی در هر مرحله اندازه هر قسمت از n به $(1-k)n$ کاهش پیدا می‌کند. پس بعد از i مرحله تعداد عناصر برابر $(1-k)^i n$ خواهدبود. وقتی که به یک برگ می‌رسیم تعداد عناصر برابر ۱ خواهدبود. m را بزرگ‌ترین مقداری در نظر بگیرید که $(1-k)^m n=1$. (یعنی اینکه به یک برگ رسیده‌ایم.) پس $(1-k)^m=1/n$ که اگر از دو طرف لگاریتم بگیریم خواهیم داشت: $m=\frac{-\log n}{\log(1-k)}$

۵.۸.۱۳ مرتب‌سازی سریع در حالتی تحلیل کنید که در هر مرحله عناصر را به ۳ قسمت تقسیم کنیم.

راه حل:

در این حالت با حالت عادی تفاوتی نخواهد کرد و فقط در بهترین حالت و حالت میانگین لگاریتم در مبنای ۳ خواهدبود که باز هم پیچیدگی الگوریتم تغییر نمی‌کند.

۵.۸.۱۴ n عدد را در نظر بگیرید که در آن‌ها k عدد متفاوت وجود دارد و هر عدد دقیقا $\frac{n}{k}$ بار تکرار شده است. الگوریتمی از $O(n\,logk)$ ارایه دهید برای مرتب کردن این اعداد.

راه حل:

مانند مرتب‌سازی سریع عمل می‌کنیم اما در هر مرحله ابتدا با هزینه $O(n)$ میانه اعداد را پیدا کرده و آن را به عنوان pivot در نظر می‌گیریم. هم‌چنین وقتی به $\frac{n}{k}$ عدد رسیدیم که همه آن‌ها برابر بودند الگوریتم را خاتمه می‌دهیم. پس بنابراین $\log k$ مرحله داریم که در هر مرحله $O(n)$ هزینه می‌کنیم. پس در کل الگوریتم از $O(n\log k)$ خواهدبود.