B۰۶۶A"۰۶۶"۹F۰۶"۸F۰۶"۷F۰۶"۶F۰۶"۵F۰۶"۴F۰۶"۳F۰۶"۲F۰۶"۱F۰۶"۰F۰۶"

الف) از یک $k-dtree$ استفاده می‌کنیم.

ب) الف) الگوریتم زیر را ارائه می‌دهیم

```
FIND K'TH BIGGEST(M: MAXHEAP, K: INT)
let m : MaxHeap
m.add(M.root)
let count = 0
let node : Node
while count < k
    node = m.pop() // extract max
    count = count + 1
    m.add(node.leftchild)
    m.add(node.rightchild)
return node.value
```

توضیح به این شکل عمل کردم که یک ماکس هیپ ساختم که مقادیری که گره‌هایش نگهداری می‌کنند خود اشاره‌گری به گره‌های مکس هیپ اصلی است. و از ریشه شروع کردیم و هر بار آمدیم ماکسیمم این هیپ کمکی را پاپ کردیم (که می‌شود اشاره‌گر به گره با بیشترین مقدار در هیپ اصلی) و بچه‌های آن گره را به هیپ اضافه می‌کنیم برای ادامه پیمایش.

شبه‌اثبات می‌دانیم در هیپ همیشه گره پدر از فرزندانش بزرگ‌تر است. در هر مرحله وقتی بزرگ‌ترین گره را از هیپ در می‌آوریم بچه‌هایش می‌توانند پس از آن گزینه خوبی باشند برای پیمایش. البته اگر گره‌های قبلا اضافه شده دیگر گزینه خوبی باشند طبیعتا پیمایش از آن ادامه می‌یابد. به عبارتی بچه‌های این گره بهترین کاندیدا برای اضافه شدن هستند چون تا وقتی گره‌های دیگر دیده نشده اند فرزندانشان کاندیدای خوبی نیستند (چون پدرشان از خودشان بزرگتر و در نتیجه گزینه بهتری است).

مرتبه زمانی در هر دور از حلقه یک گره از هیپ کمکی کم می‌شود و دو گره فرزند آن (از هیپ اصلی) به آن اضافه می‌شود. به عبارتی تا مرحله k ام اندازه هیپ k می‌شود و در تمام این k مرحله اندازه آن کمتر مساوی k است و در نتیجه تمام عملیات‌های حذف و اضافه‌ای که روی آن انجام می‌شود از مرتبه $logk$ می‌باشد. و با k مرحله اجرای حلقه مرتبه کل $O(klogk)$ می‌شود.

ب) الگوریتم زیر را ارائه می‌دهیم

```
VISIT(N: NODE, K: INT, X: INT, &COUNTER: INT)
// notice that counter is changed globally because of reference
// we only wish to find k bigger nodes than x
if n.val >= x
    return
// one more match
++counter
if counter >= k
    return
visit(n.left, k, x, ptr to counter)
visit(n.right, k, x, ptr to counter)
```

این الگوریتم زمانی پایان میابد یا k عدد بزرگ‌تر از x پیدا شود، یا دیگر در این پیمایش‌ها در تمام شاخه‌ها با عددی کوچک‌تر از x مواجه شویم. از نظر زمانی در هر شاخه یا k پیشروی می‌کند یا درجا متوقف می‌شود. به عبارتی هر گرهی که فرزندانش از x کوچک تر باشند با خودشان (که از x بزرگ‌تر بود) سرشکن می‌شود و حتما به ازای هر دو گره الکی یک گره بزرگ‌تر می‌بینیم واگر گرهی از x کوچک تر باشد تمام فرزندانش نیز کوچک‌تر است پس نیازی به بررسی فرزندانش نیست.

از طرف دیگر اگر k عضو بزرگ‌تر از x پیدا کنیم حتما k امین عضو بزرگ‌تر نیز از x بزرگ‌تر خواهد بود.

در نتیجه طی $O(k)$ عملیات می‌توان به پاسخ رسید. الگوریتم به شکل زیر ارائه می‌دهیم:

۱. همه علامت زده ها را در یک آرایه میریزیم و آن را با حافظه $O(k)$ و مرتبه زمانی $O(klogk)$ مرتب می‌کنیم.

۲. آرایه با گنجایش $n-k$ می‌گیریم و یک دور در آرایه اصلی پیمایش می‌کنیم و هر عضوی که علامت ندارد را در آن آرایه کپی می‌کنیم.

۳. حال دو آرایه مرتب شده داریم. کافی است بر روی هر یک یک نشان‌گر قرار دهیم و عضو کوچک تر را در یک آرایه مقصد به اندازه n بنویسیم. که در بدترین حالت $O(n+k)$ زمان خواهد گرفت.

۴. در نهایت از $O(n)$ حافظ استفاده کردیم و $O(klogk) \in O(nlogk)$ نیز زمان گرفتیم.

ساختمان داده‌ها (۲۲۸۲۲)

مدرس: حسین بومری

[زمستان ۹۹]

سوال ۸: مرتب‌سازی ادغامی چندتایی — نگارنده: آئیریا محمدی

الف) هر یک از بخش‌های لیست اندازه k دارند و در بدترین حالت در $O(k^۲)$ مرتب می‌شوند. جمعا n/k لیست داریم پس روی هم $O(\frac{n}{k} \cot k^۲) = O(nk)$ زمان می‌گیرند.

ب) تقسیم یک لیست به n/k لیست مرتب مانند این است که بخشی از عمق الگوریتم مرج سورت را طی کرده باشیم (تا وقتی که اندازه هر لیست k شده باشد). در هر مرحله اندازه زیرلیست‌ها دوبرابر می‌شود پس انگار $log.k$ مرحله جلوییم. و در نتیجه تعداد عملیات باقی مانده متناسب با عمق باقی مانده است که می‌شود $log \cdot n - log \cdot k = log\frac{n}{k}$. در هر مرحله نیز از $\Theta(n)$ کار انجام می‌دهیم پس مرتبه زمانی این ادغام از $\Theta(nlog\frac{n}{k})$ خواهد بود.

ج) k نمی‌تواند از $\Omega(n)$ باشد چرا که الگوریتم ما از مرتبه $\Omega(n^۲)$ خواهد شد. به ازای $k = O(log(n))$ خواهیم داشت

$$T = O(n \cdot logn + nlog\frac{n}{logn}) = O(nlogn - nlog(log(n))) = O(nlogn)$$

پس $k(n) \in O(logn)$.

د) اگر a, b ثابت باشند داریم $T(n,k) = ank + bnlog\frac{n}{k} = ank + bnlogn - bnlogk$. قرار می‌دهیم $\frac{d}{dk}T = ۰$ که می‌دهد

$$an + ۰ - bn/k = ۰ \rightarrow k = b/a$$

یعنی یک مقدار بهینه ثابت برای k وجود داره. می‌توانیم برای n های نسبتا بزرگ k های مختلف از $O(logn)$ را امتحان کنیم و بهترین مورد را انتخاب کنیم.