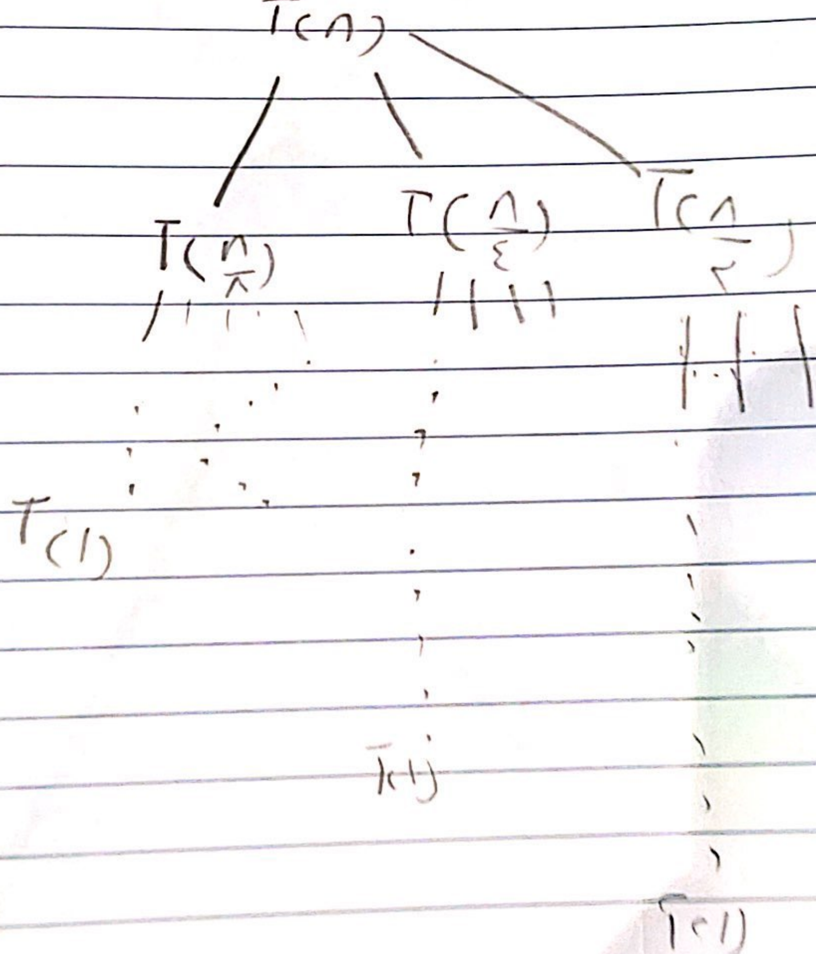
a) $T(n) = T(n-4) + n^2 = T(n-8) + (n-4)^2 + n^2 = \dots$

$T(1) + 4^2 + 8^2 + \dots\dots n^2 = n(n+1)(n-1)\dots$

b)



$\frac{7n}{8}$

$\frac{7^2 n}{8^2}$

فرض این که تمامی شاخه‌ها هم عمق هستند

$$T(n) \leq \sum_{i=0}^{\log n} \left(\frac{7}{8}\right)^i n = \frac{1-\left(\frac{7}{8}\right)^{\log_8 n+1}}{1-\frac{7}{8}} n = 8n\left(1-\frac{7}{8} n^{-0.0}\right)$$

$$= 8n - 7n^{0.93}$$

پس $T(n) \in \Theta(n)$ است

Year: Month: Date: ( ) Subject

۵) از قضیه اصلی استفاده می کنیم ، $b=3$ ، $a=4$

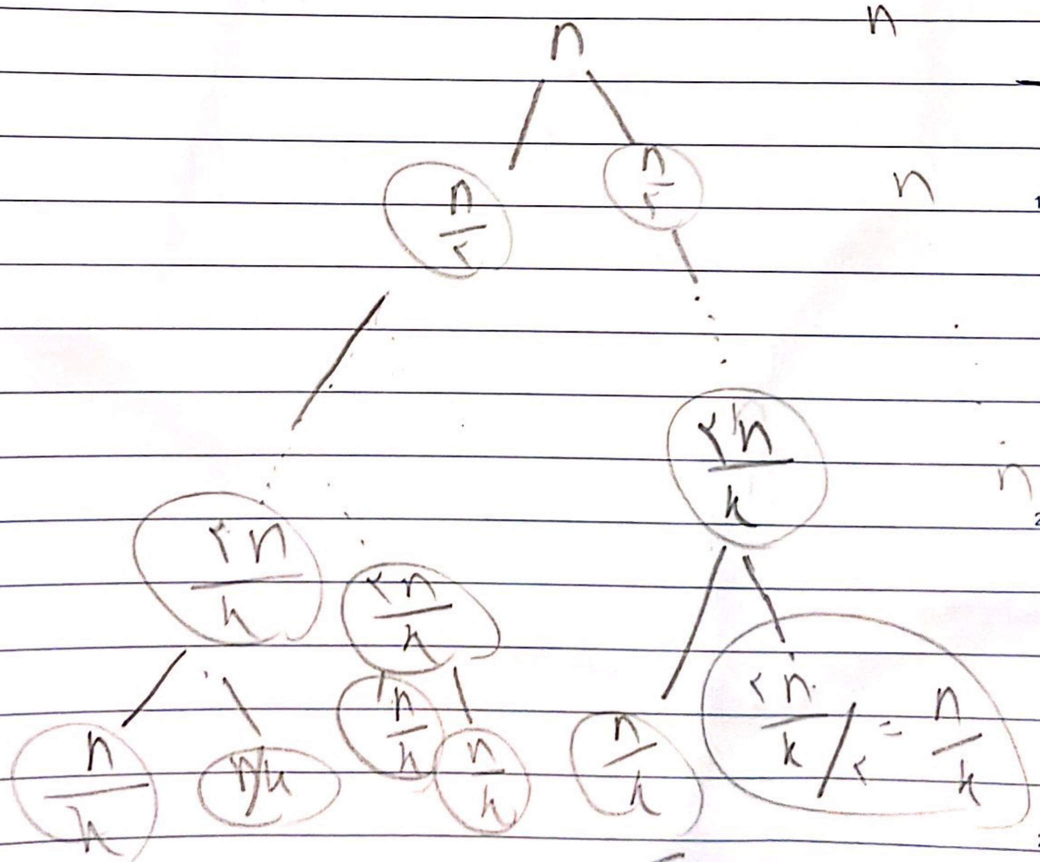
$n^{\log_3 4}$, $f(n)=n^{\frac{1}{2}}$

حالت اول را چک می کنیم

$f(n) \in O(n^{\log_3 4 - \epsilon}) \Rightarrow \log_3 4 - \epsilon \geq \frac{1}{2}$

$\Rightarrow \epsilon > 0$ ✓ $T(n) \in \Theta(n^{\log_3 4})$

۲-



رابطه کل به شکل $T(n) = kT(\frac{n}{k}) + n\log k$

در هر مرحله

هزینه n و $\log k$ گره

Date: ( )

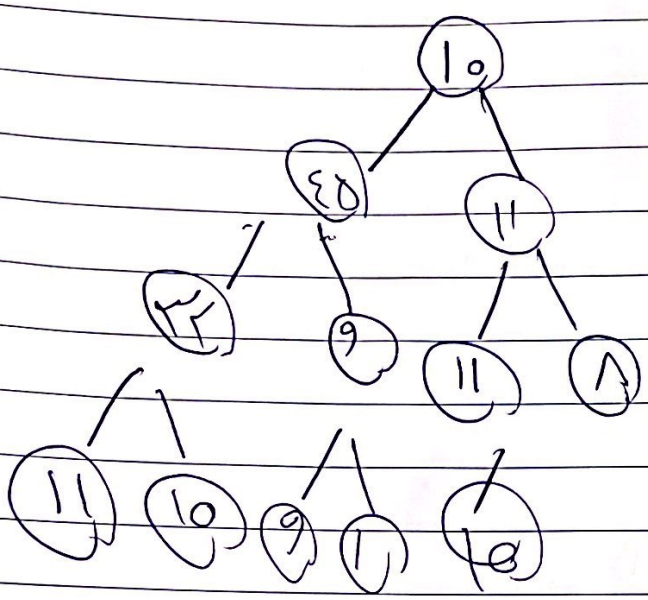
۲ از روش رختی می توان به این جواب رسید:

$$T(n) = \sum_{i=0}^{\log_4 n} \frac{n}{4^i} \log 4^i = n \log 4 \sum_{i=0}^{\log_4 n} 1 = \Theta(n \log_4 n) = \Theta(n \log n)$$

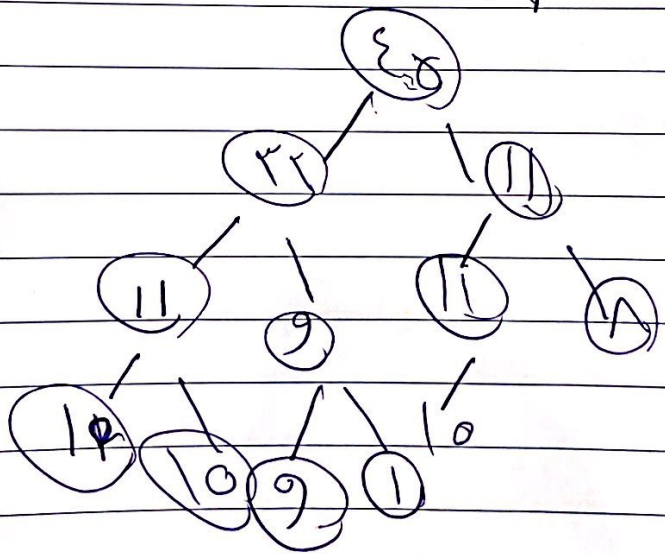
۳- می توان الگوریتمی به این صورت ارائه داد که هربار عنصر $k/2$ دو آرایه را مقایسه کرده و هرکدام کوچکتر بود نیمه سمت چپ $[k-k/2]$ را دور می ریزیم. و هربار این کار را تکرار می کنیم تا زمانی که یک عنصر باقی بماند که جواب را با $O(1)$ برمی گرداند. از آن جا که هربار k نصف می شود در بدترین حالت $O(\log k)$ است. پس حداکثر آن $O(\log(m+n))$ است.

Year: Month: Date: ( ) Subject

۴- مرحله اول:

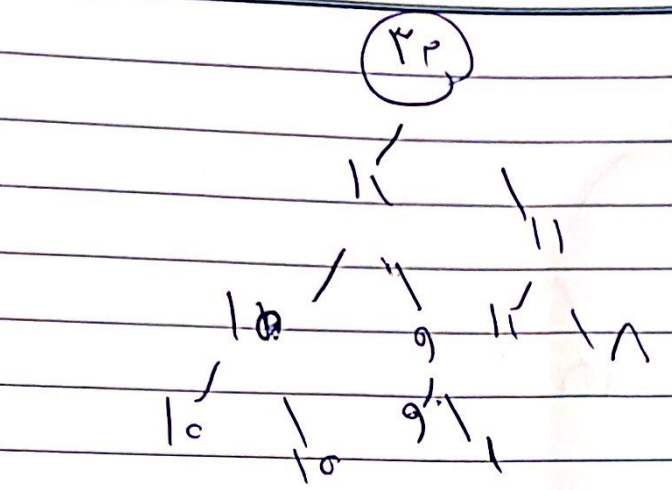


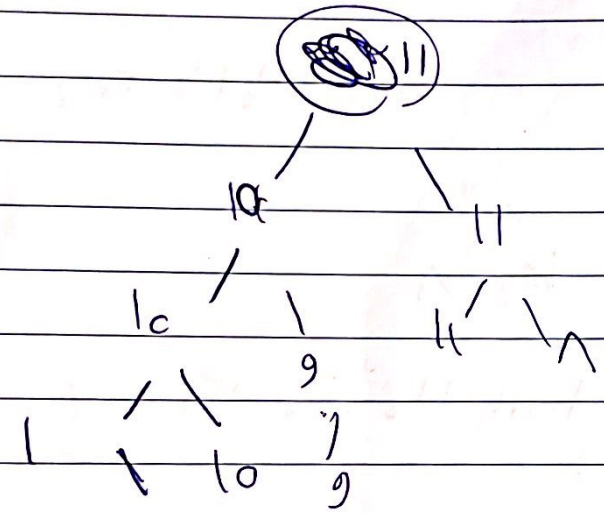
مرحله (۲): صدا زدن Build max heap

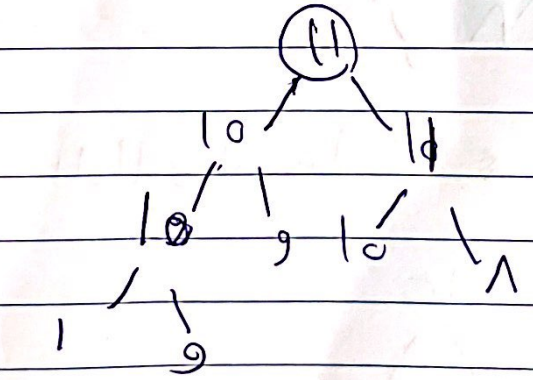


مراحل (۳) تا (۴) با heap جای آخرین عنصر را با ریشه عوض کرده و سپس heapify کرده

Year: Month: Date: ( ) Subject

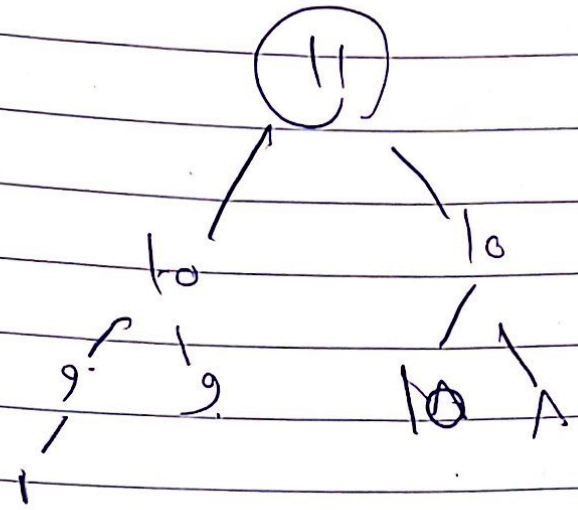






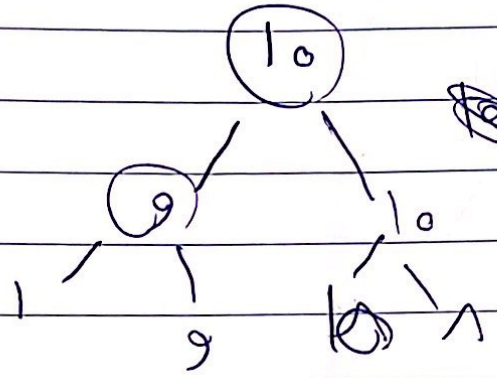
AHURA

Year: Month: Date: ( ) Subject



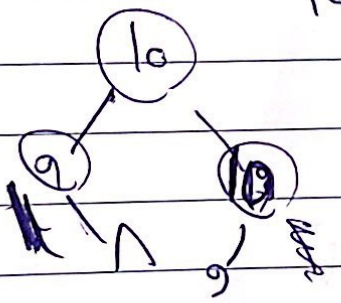
٩

١١، ١١، ٣، ٤، ٥



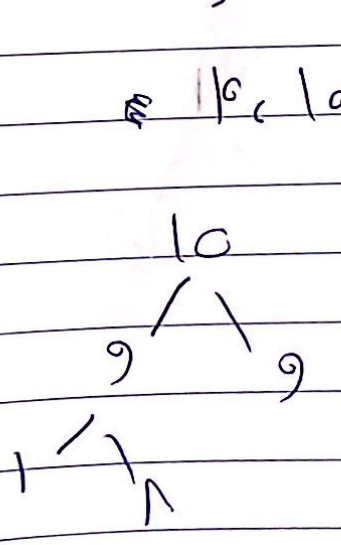
٧

١١، ١١، ١١، ٣، ٤، ٥



٨

١٥، ١١، ١١، ١١، ٣، ٤، ٥

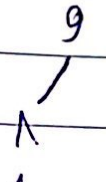


٩

١٥، ١٥، ١١، ١١، ١١، ٣، ٤، ٥

Year: Month: Date: ( ) Subject

١٠

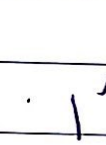


٤٥ ، ٣٢ ، ١١ ، ١١ ، ١١ ، ٥ ، ١٥ ، ١٥ ، ١٥

١١

٤٥ ، ٣٢ ، ١١ ، ١١ ، ١١ ، ٥ ، ١٥ ، ١٥ ، ١٥ ، ٩

١٢



٤٥ ، ٣٢ ، ١١ ، ١١ ، ١١ ، ٥ ، ١٥ ، ١٥ ، ٩ ، ٩

١٣

١

٤٥ ، ٣٢ ، ١١ ، ١١ ، ١١ ، ٥ ، ١٥ ، ١٥ ، ٩ ، ٩ ، ٨

١٤

٤٥ ، ٣٢ ، ١١ ، ١١ ، ١١ ، ٥ ، ١٥ ، ١٥ ، ٩ ، ٩ ، ٨ ، ١

Year: Month: Date: ( ) Subject

-0

Heap Sort:

این الگوریتم از هیپ بیل ساخته شده. برای تابع heapify تمام درخت را از n/2 تا 1 پایین پیمایش می کنیم پس با دوبار صدا زدن هر خانه داریم O(logn). پس برای تابع Build heap از گره آخر شروع کرده تا به ریشه برسیم و هر بار تابع heapify را صدا می زنیم. در آخر heapify n-1 بار صدا زده شده است که در کل می شود O(nlogn). برای پیچیدگی فضایی O(1) را داریم چرا که فقط به فضای آرایه نیاز داشتند

Counting Sort:

برای شمارش هر خانه آرایه زمان k نیاز داریم و برای پیدا کردن خانه درست آن n پس O(n+k).

بدترین حالت: زمانی اتفاق می افتد است که بزرگ ترین داده بسیار بزرگ تر از بقیه داده ها باشد. که O(n) می شود

بهترین حالت: زمانی که تمامی داده دارای مقدار ساوی باشند پس اینجا O(1+n)

Year: Month: Date: ( ) Subject

حالت میانه: $O(n+k)$.

پیچیدگی فضایی یک آرایه با سایز k استفاده کردیم که $O(k)$ می شود.

Quick Sort:

در حالت میانه Partition بندی عناصر زمان $O(n)$ دارد. در بدترین حالت $O(n^2)$ که در زمان مرتب $O(n\log n)$. در بدترین حالت. پیچیدگی فضایی $O(n)$ بودن آرایه اتفاق می افتد.

پیچیدگی زمانی:

$$quicksort = heapsort$$

فضایی: $heapsort < counting\ sort < quicksort$

Month: Date: ( )

۶- بدترین حالت: بدترین حالت برای زمانی است که Partition مسئله را به دو زیر مسئله با اندازه های 0 و n-1 تقسیم کند.

$$T(n) = T(n-1) + T(0) + \Theta(n) = T(n-1) + \Theta(n)$$

$$= T(n-2) + \Theta(n-1) + \Theta(n) = \dots = n\Theta(n) = \Theta(n^2)$$

بهترین حالت: اگر با اجرای Partition مسئله را به دو زیر مسئله با اندازه حداکثر $n/2$ تقسیم کند بهترین حالت اتفاق افتاده است $T(n) = 2T(\frac{n}{2}) + \Theta(n)$. طبق قضیه اصلی:

$$\Theta(n \log n)$$