فرهاد اران ۹۹۳۱۰۶

۱ -

stable : Counting Sort

1a 2 4 1b 3 1c

از چپ به راست روی مقادیر حرکت می‌کنیم و مقادیر را در ظرف های مربوطه تمامش می‌کنیم پس از نهایت هنگامی که قصد داریم مقادیر داخل ظرف ها را محاسبه کنیم 1a پیش از 1b و 1b پیش از 1c محاسبه شده است.

not stable : Quick Sort

7 3 4 5a 6 5b 1 3

فرض کنید 5a یا 5b به عنوان محور انتخاب شود ممکن است با توجه به الگوریتم Partition هرکدام از 5 های دیگر قبل یا بعد از 5 انتخاب شده به عنوان محور قرار گیرد.

not stable : Heap Sort هنگامی که heap نهایی ساخته شده و قصد خارج کردن اعداد از آن را داریم رأس heap پاک شده و با آخرین عنصر هرم جابجا می‌شود که ترتیب را به هم می‌زند.

Stable : Merge Sort می‌توان از استقرا استفاده کرد به این صورت که فرض می‌کنیم دو زیر آرایه به صورت stable مرتب شده‌اند دقت بگیریم هنگام merge کردن پایدار بودن حفظ می‌شود. این موضوع به راحتی قابل انجام است هنگام که دو پوینتر ما به صورت موازی در دو زیر آرایه حرکت می‌کنند هنگامی که به عنصر برابر می‌رسند اولویت را با زیرآرایه سمت چپ قرار می‌دهیم.

Stable : Insertion Sort فرض کنید در عدد یکسان در آرایه موجودات هنگامی که الگوریتم به عدد دوم می‌رسد و قصد دارد مکان مناسب برای قرار دادن عدد مورد نظر پیدا کند عدد را بعد از تمام اعداد کوچکتر یا مساوی آن قرار می‌دهد بنابراین عدد دوم بعد از عدد اول قرار گرفته

زهرا امان ۹۹۳۱۰۰۶

۱- SelectionSort : not stalbe   ۲a ۳ ۴ ۵ ۲b ۱

در آرایه بالا الگوریتم به دنبال کوچکترین مقدار که ۱ است می‌گردد و وقتی آن را پیدا کرد با عنصر اول یک ۲a جابجا می‌کند و این باعث می‌شود ۲a بعد از ۲b قرار گیرد.

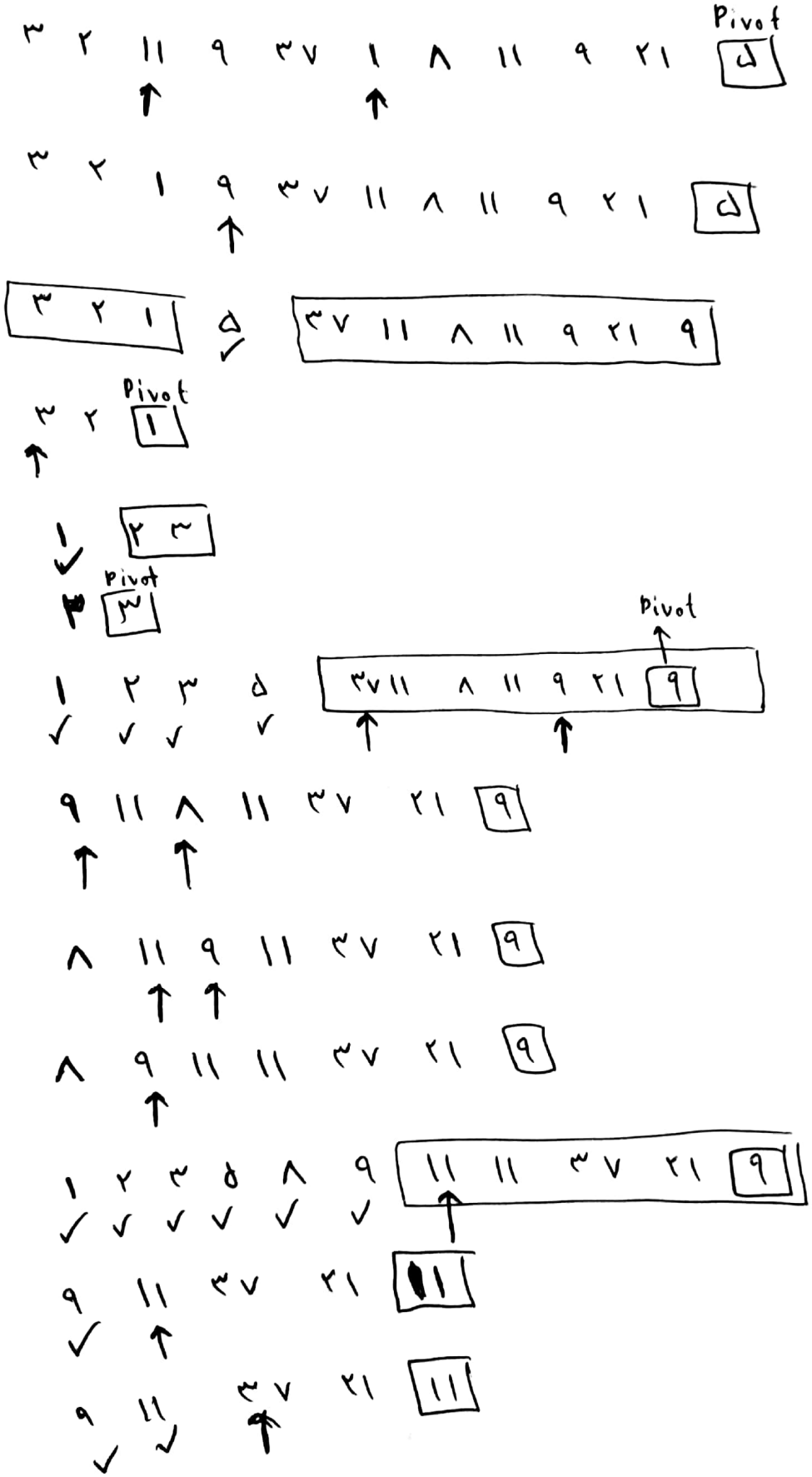
BubbleSort : Stable الگوریتم هر بار آرایه را طی می‌کند و اگر inversion مشاهده کند آن‌ها را swap می‌کند در نتیجه جای ۲ عدد مساوی هیچگاه با هم عوض نمی‌شود.

Bucket Sort : Stable مانند CountingSort چون عناصر مساوی به ترتیب وقوع در ظرف‌های مربوط به خود قرار می‌گیرند و در نهایت طبق همان ترتیب از ظرف خود خارج می‌شوند

Radix Sort : Stable الگوریتم در هر مرحله علاوه حال انجام یک الگوریتم Bucket Sort است و سپس برحسب رقم بعدی این الگوریتم را انجام می‌دهد پس پایدار است.

نرهار ایمان ۹۹۳۱۰۰۹

۲ - Quick Sort



فرهاد امان ۹۹۳۱۰۰۶

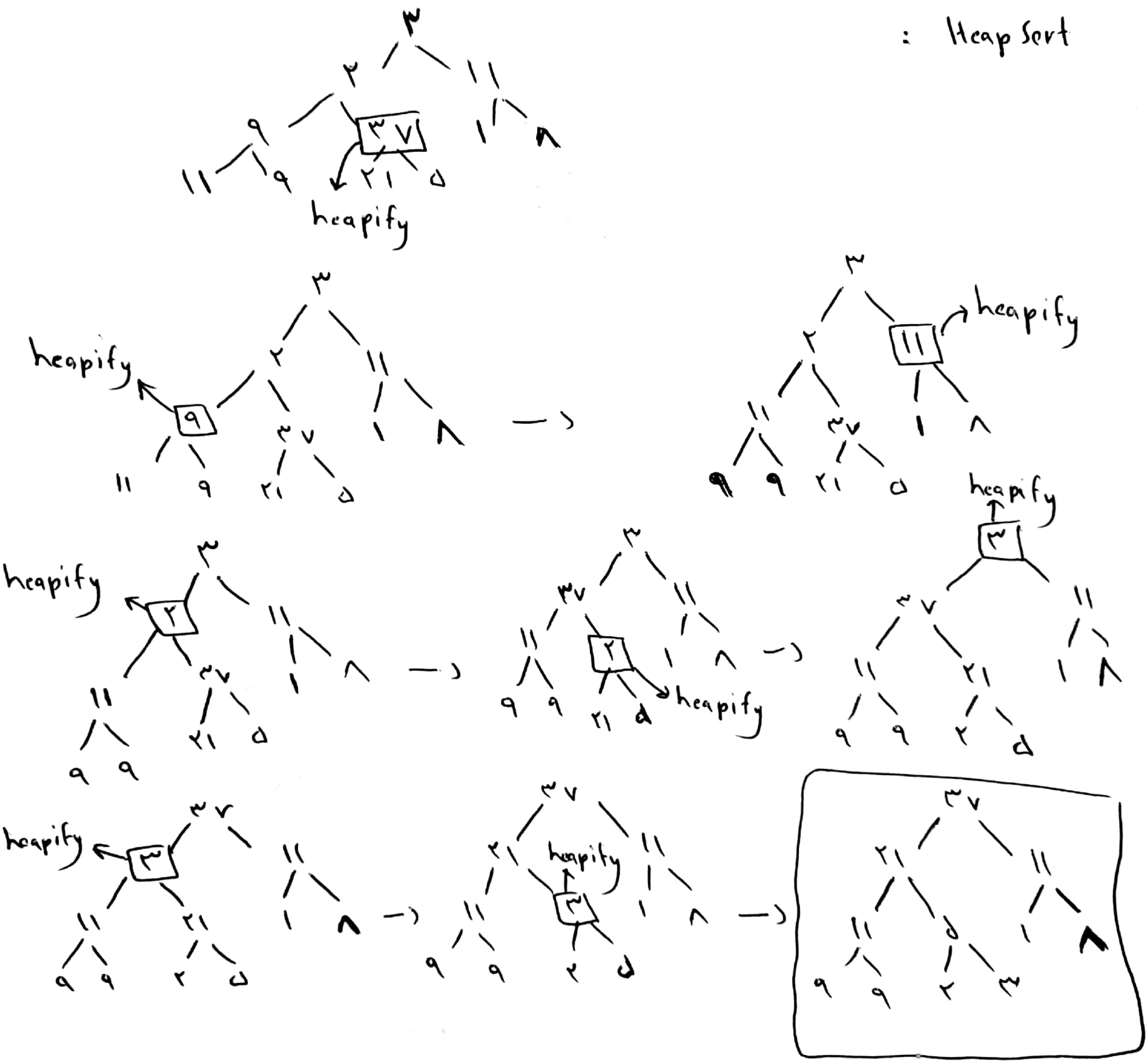
۲-

Pivot

۱ ۲ ۳ ۵ ۸ ۹ ۱۱ ۱۱ ۲۱ ۳۷

۱ ۲ ۳ ۵ ۸ ۹ ۱۱ ۱۱ ۲۱ ۳۷

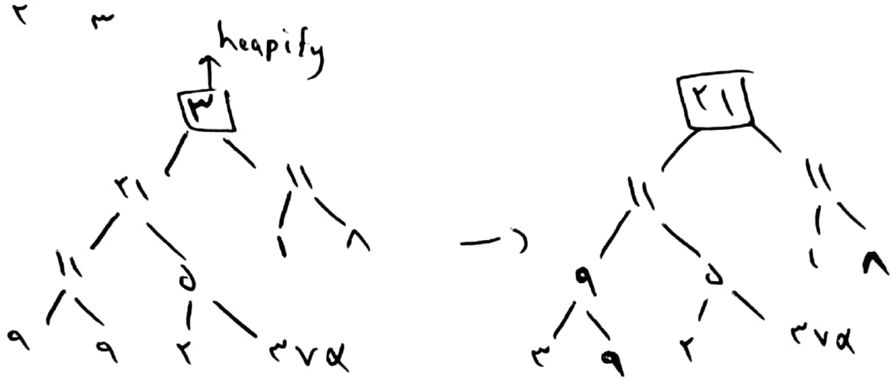
Heap Sort :



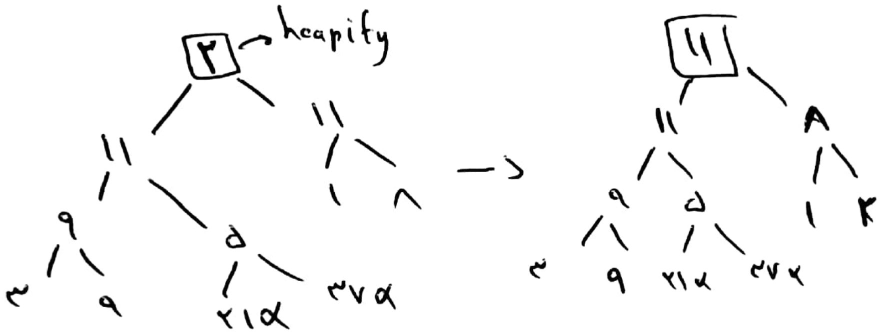
زهرا دامان ۹۹۳۱۰۰۶

۲-

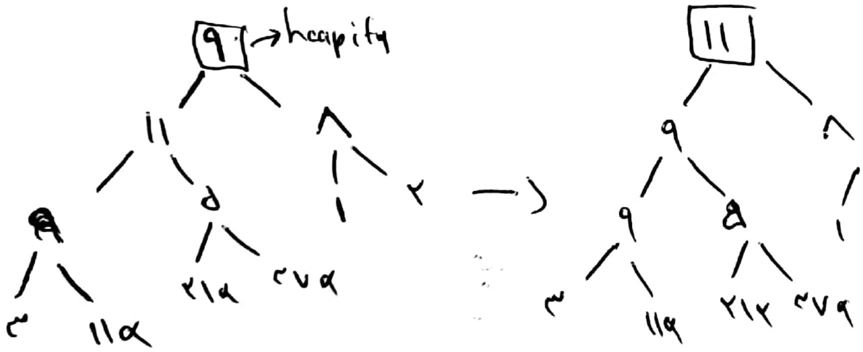
۳۷



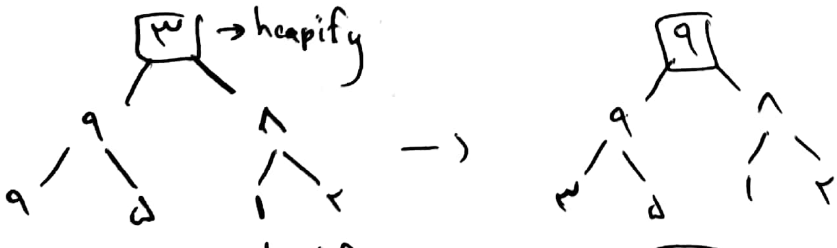
۳۷ - ۲۱



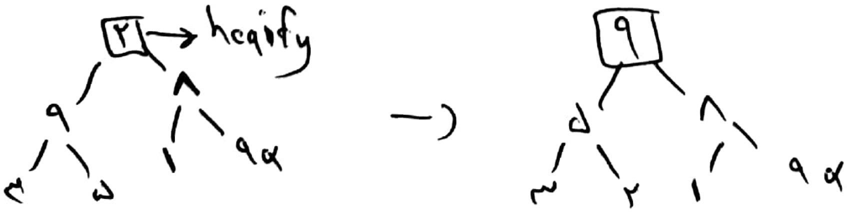
۳۷ - ۲۱ - ۱۱



۳۷ - ۲۱ - ۱۱ - ۱۱



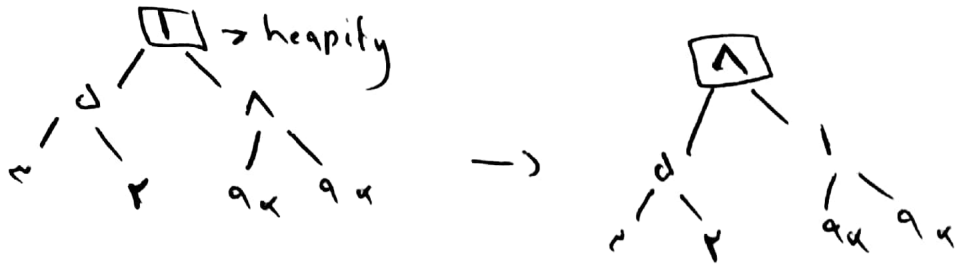
۳۷ - ۲۱ - ۱۱ - ۱۱ - ۹

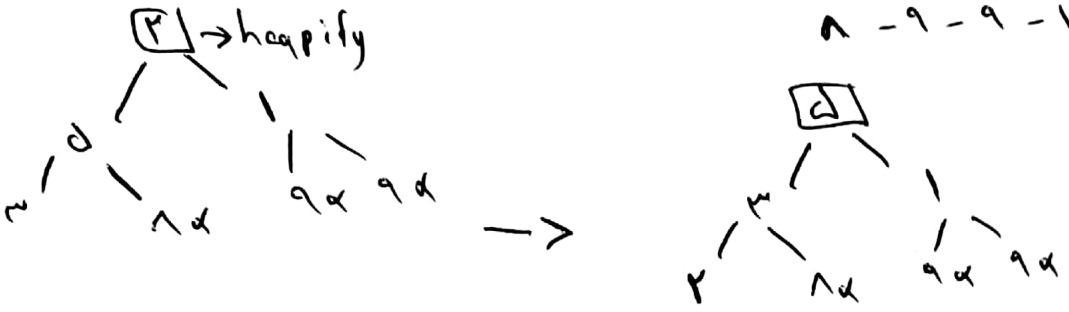


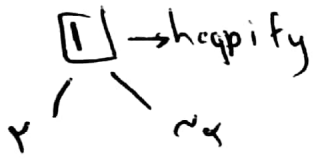
۳۷ - ۲۱ - ۱۱ - ۱۱ - ۹ - ۹

فرهاد ایان ۹۹۴۱۰۶

۲ -







۱ ۲ ۳ ۵ ۸ ۹ ۹ ۱۱ ۱۱ ۲۱ ۳۷

زهرا امان ۹۹۳۱۰۰۶

۳ - Counting Sort : فرض کنید اعداد در بازه ی k باشند در نتیجه حداکثر به k ظرف نیاز داریم و البته به مقدار n هم فضا برای خود اعداد نیاز داریم پس پیچیدگی فضایی آن $O(n+k)$ است. برای زمان ابتدا باید روی اعداد پیمایش شود تا ظرف هرکدام مشخص شود که $O(n)$ زمان می‌خواهد. پس باید هر ظرف پیمایش شود که $O(n+k)$ است. پس پیچیدگی زمانی آن $O(n+k)$ است.

چون الگوریتم‌های heap sort و Quick Sort الگوریتم‌های inplace هستند و تنها به اندازه مقادیر خود فضا می‌گیرند پیچیدگی فضایی هر دو $O(n)$ است.

در Quick Sort اگر مراحل تقسیم آرایه و انتخاب محور به نحوی باشد که همواره بزرگترین یا کوچکترین عدد به عنوان محور انتخاب شود.

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

اما اگر همیشه میانه به عنوان محور انتخاب شود.

$$T(n) = 2T(n/2) + \Theta(n) = O(n \lg n)$$

به طور متوسط اگر بتوانیم بگوییم که محور انتخاب شده همواره از ضریب ثابتی از اعداد بزرگتر و از ضریب ثابتی از اعداد بزرگتر است به طور مثال $\frac{1}{10}$ اعداد

$$T(n) = T(n/10) + T(9n/10) + \Theta(n) = O(n \lg n)$$

درباره ی ساخت heap در الگوریتم heapsort می‌توان گفت که زمان $O(n)$ را لازم دارد. در ادامه باید اعضای آرایه به ترتیب از heap استخراج شوند که هرکدام نیاز به یک عملیات heapify با $O(\lg n)$ زمان پیدا می‌شود که زمان کل الگوریتم $O(n \lg n)$ می‌شود.

نرهاد امان ۹۹۳۱۰۰۶

۴ - کافیست از روش Radix Sort و همچنین تغییر مبنای اعداد استفاده کنیم به این صورت که مبنای اعداد را n در نظر می‌گیریم.

$$O(n \times d)$$

$$d = \log_{\text{مبنا}} M = \log_{n} n^2 - 1 = O(1)$$

$$O(n \times d) \rightarrow \boxed{O(n)}$$

همچنین در ابتدا باید n ظرف آماده کنیم که آن هم از $O(n)$ طول می‌کشد.

فرهاد امان ۹۹۴۱۰۰۶

۵ - کافیت از یک ساختمان داده map استفاده کنیم برای هرکدام از اعدادی که امکان وجود در لیست ما را دارند یک متغیر در نظر می‌گیریم (برای این کار می‌توان max امکان اعداد داخل لیست را در $O(n)$ محاسبه کرد و آرایه‌ای به اندازه max گرفت) پس بر روی آرایه حرکت می‌کنیم و با وقوع هر مقدار متناظر آن در آرایه map را یک واحد افزایش می‌دهیم این کار در $O(n)$ صورت می‌گیرد. در آخر مقادیر داخل map را بررسی می‌کنیم و هرکدام از $\frac{n}{3}$ بیشتر بود پاسخ مسأله ما است.

۶ - الف) ابتدا میانه هرکدام از دو آرایه را در $O(1)$ با توجه به مرتب بودن آرایه‌ها محاسبه می‌کنیم. پس این دو میانه به دست آمده با هم مقایسه می‌شوند که دو حالت برای مقایسه وجود دارند. به عنوان مثال فرض کنیم میانه آرایه اول m_1 و میانه آرایه دوم m_2 باشد اگر $m_1 < m_2$ می‌دانیم که تمام اعداد نیمه چپ آرایه اول کوچکتر از اعداد نیمه راست آن هستند از طرفی تمام اعداد نیمه چپ آرایه دوم کوچکتر خواهند بود پس بنابراین نیمه سمت چپ آرایه اول کنار گذاشته خواهد شد و همین اتفاق هم برای نیمه راست آرایه دوم می‌افتد اگر $m_1 > m_2$ باشد هم عکس همین اتفاق می‌افتد پس متوجه می‌شویم هربار نیز هر آرایه نصف می‌شود و به صورت بازگشتی الگوریتم را روی اعداد باقی مانده اجرا می‌کنیم. و با توجه به نصف شدن نیز آرایه‌ها و هزینه $O(1)$ در هر مرحله هزینه‌ی کل $O(\log n)$ است.

ب) الگوریتم این بخش مانند بخش اول است فقط توجه کنید که در مراحل نصف شدن آرایه نیز آرایه زودتر به عدد ۱ می‌رسد و جواب مسأله مشخص می‌شود پس هزینه‌ی کلی $O(\log \min(m,n))$ است.

فرهاد ایمان ۹۹۴۱۰۶

۷ - الگوریتم Quick Select را در کلاس بررسی کردیم. این الگوریتم امکان پیدا کردن k امین عضو بزرگ آرایه را در $O(n)$ به ما می‌دهد. یک بار با این الگوریتم میانه را در $O(n)$ پیدا می‌کنیم بار بعدی میانه + k این عضو را در $O(n)$ می‌یابیم. پس روی آرایه پیمایش کرده و هر عنصری که از نظر اندازه بین میانه و میانه+k این عنصر قرار دارند را در $O(n)$ جدا می‌کنیم. در نهایت $O(k)$ عدد داریم که با استفاده از یک الگوریتم مرتب سازی مانند mergesort آن‌ها را در $O(k \log k)$ مرتب می‌کنیم. در نهایت هزینه ما برابر $O(n + k \log k)$ خواهد بود.

۸ - Aggregate : به ازای هر عدد توان ۲ به اندازه خود عدد هزینه می‌کنیم پس تمام هزینه اعداد توان دو برابر $\sum_{i}^{k} 2^i$ است که 2^{k+1} خواهد بود. اگر آخرین عدد توان ۲ عدد 2^k باشد پس هزینه‌ی کل این بخش از $O(n)$ است هر کدام از اعداد غیر توان ۲ هزینه $O(1)$ دارند که در مجموع هزینه $O(n)$ خواهند داشت پس می‌توان نتیجه گرفت کل تابع هزینه‌ای $O(n)$ دارد.

پس می‌توان نتیجه گرفت هزینه سرشکن هر عمل $O(1)$ است. $\frac{O(n)}{n} = O(1)$

Accounting : کافیت در هر مرحله برای یک عملیات به جای یک هزینه ۱ واحدی ۳ واحد هزینه کنیم. با توجه این که ۱ واحد برای عملیات های عادی هزینه نیازات در هر عملیات عادی ۲ واحد ذخیره می‌شود. این ۲ واحد اضافه برای نقشگاهی ذخیره می‌شود که بخواهیم یک عملیات با 2^k واحد هزینه انجام دهیم اینگونه از قبل هزینه لازم را پرداخت کرده‌ایم. چون ۳ واحد هزینه از $O(1)$ است پس هر عملیات $O(1)$ واحد هزینه سرشکن دارد.

فرهاد ایمان ۹۹۴۱۰۰۶

۸- Potential [redacted]

$\Phi(h)$ ، $2n$ - Total Cost

h را تعداد عملیات های صورت گرفته در نظر می گیریم ■ $\Phi(h)$ نباید هیچگاه منفی شود.

$$c_i + \Phi(h_i) - \Phi(h_{i-1}) = \boxed{3}$$

پس نتیجه می گیریم ■ هر عملیات از $O(1)$ می باشد.

۹- هر بار که یک عمل increment اتفاق می افتد یک ۰ تبدیل به ۱ شده و چند ۱ به ۰ تبدیل می شوند. کافیت علاوه بر هزینه ی ۱ کردن ۰ هزینه ای اضافی از پیش ذخیره کنیم.
به عنوان مثال برای ۱ کردن یک بیت ۰ به جای ۱ واحد هزینه ۲ واحد هزینه کنیم.
در این حالت هنگام عملیات reset ی داریم تنها بیت هایی که از پیش ۱ شده اند قرار است به ۰ تبدیل شوند که هزینه ۰ شدن آن ها از پیش پرداخت شده است.
از طرفی می دانیم که عملیات decrement درست مانند عکس عملیات increment است و تعداد decrement ها حداکثر به اندازه increment ها است پس می توان گفت که با وجود decrement هزینه حداکثر ۲ برابر خواهد شد که باز هم تأثیری روی $O(1)$ باقی می ماند.
هزینه کلی ندارد و هزینه هر عملیات همچنان از