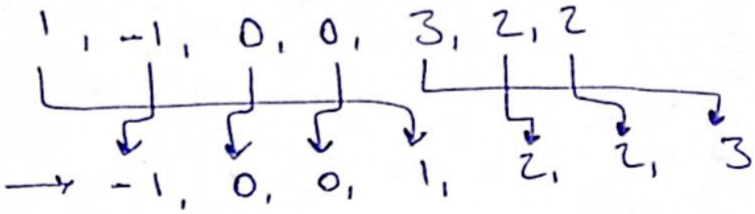
بردیا اردکانیان 9831072

①

Counting Sort: Stable

اعداد ترتیب خود را حفظ می کند چون این الگوریتم به ترتیب پیمایش می کند



Bubble Sort: Stable

داخل حلقه ها نیز ترتیب حفظ که در آرایه اصلی قرار دارند بررسی و Swap می شوند پس ترتیب [i,i+1] حفظ می شود. (چون با بقیه Swap شده)

Selection Sort: not !!

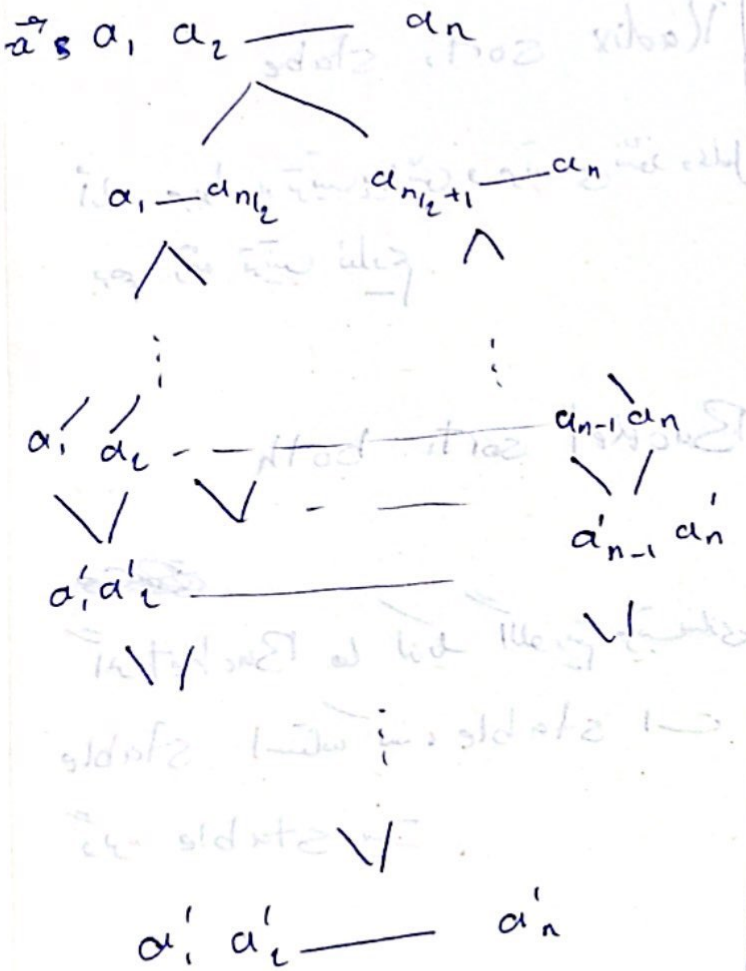
در این روش به دنبال کمترین مقداریم تا آن را با شئ فعلی جابه جا کنیم. این Swap پایداری را از بین می برد. درست مثل مرتب کردن ورق پاسور

Insertion Sort: Stable

در الگوریتم آن به ترتیب روی اعضا اصلی آرایه پیمایش می کنیم در نتیجه ترتیب بهم نمی خورد

Merge Sort: Stable

آرایه به زیر مسئله ها تقسیم می شود ولی در نهایت در بخش ترکیب به ترتیب اولیه در کنار هم قرار می گیرند



Heap Sort: not !!

به عنوان مثال آرایه 5, 3, 2, 1 می تواند heap بسازد و heap ها مخصر به فرد نیست. پس می تواند ترتیب را بهم زند.

- Quick sort: not "

باتوجه به انتخاب pivot و عملیات swap

- ممکن است ترتیب را بهم زند.

Radix sort: stable

آرایه مربوط به ترتیب بیمارستان و مرتب می‌شود و علل بهم ریخته ترتیب نداریم.

Bucket sort: both

اگر Bucket ها از یک الگوریتم مرتب‌سازی stable استفاده کنند، stable است وگرنه نیست.

a) Q-sort

3 2 11 9 37 (1) 8 11 9 21 5 | init (2)

1 2 11 9 37 (5) 8 11 9 21 3 | pivot انتخاب P=1

1 (2) 3 5 37 11 8 11 9 21 9 | " " P=5
✓

1 2 3 5 37 11 8 (11) 9 21 9 | " " P=2
✓
pivot يعني

1 2 3 5 8 9 (9) 8 11 37 21 11 | " " P=8
✓

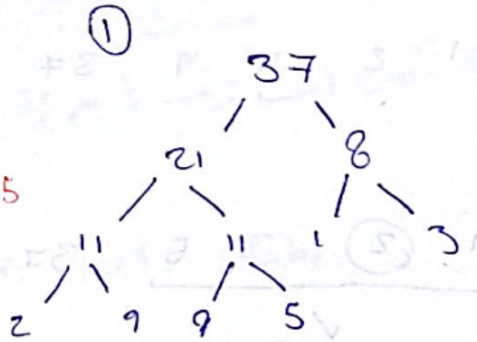
1 2 3 8 9 9 11 37 (21) 11 | " " P=9
✓

1 2 3 8 9 9 11 21 37 | " " P=21
✓

completed

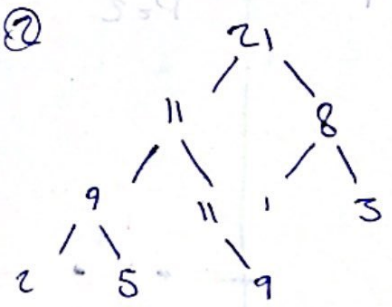
b) Heap sort [3, 2, 11, 9, 37, 1, 8, 4, 9, 21, 5]

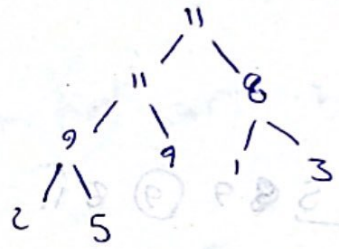
- init max heap from array

① 

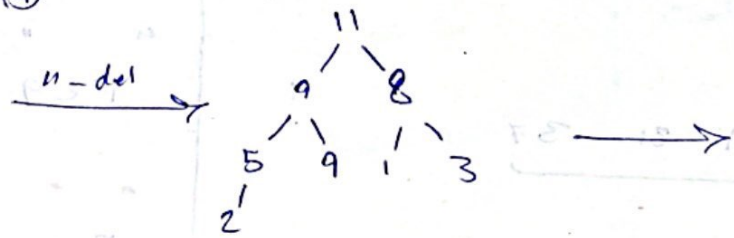
37, 21, 8, 11, 11, 1, 3, 2, 9, 9, 5

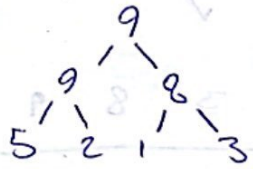
- Swap 37 (first) and 5 (Last) and remove last node. heapify

② 

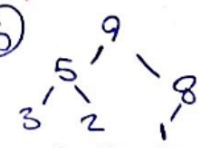
③ 

21, 11, 8, 9, 11, 1, 3, 2, 5, 9, 37 → 11, 11, 8, 9, 9, 1, 3, 2, 5, 21, 37

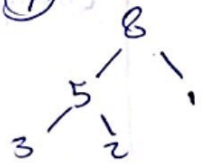
④ 

⑤ 

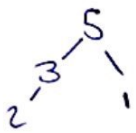
11, 9, 8, 5, 9, 1, 3, 2, 11, 21, 37 → 9, 9, 8, 5, 2, 1, 3, 11, 11, 21, 37

⑥ 

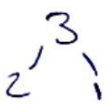
→ 9, 5, 8, 3, 2, 1, 9, 11, 11, 21, 37

⑦ 

→ 8, 5, 1, 3, 2, 9, 9, 11, 21, 37

⑧ 

→ 5, 3, 1, 2, 8, 9, 9, 11, 21, 37

⑨ 

3, 2, 1, 5, 8, 9, 9, 11, 11, 21, 37 → 2, 1, 3, 5, 8, 9, 9, 11, 11, 21, 37 → 1, 2, 3, 5, 8, 9, 9, 11, 11, 21, 37

⑩ ⑪

Heap - Sort

Time : با توجه به الگوریتم Hsort برای ساختن maxheap به $O(\gamma n)$ زمان نیاز داریم. حال باید روی هرکدام از آرایه ها به اندازه n پیمایش کنیم که زمان کلی آن $O(n\gamma n)$

swap_del

Space : تمام الگوریتم بر روی آرایه اصلی اجرا می‌شود پس پیچیدگی مکانی $O(n)$ دارد.

① Quick sort

Time : با انتخاب pivot آرایه به ۲ زیرمسئله تقسیم می‌شود و هرکدام همین روند را تکرار می‌کنند. چون هزینه ترکیب نداریم (عملیات partition خود مرتب می‌کند) پس تنها هزینه است. در بدترین حالت $O(n^2)$ را داریم که pivot = {min, max} و در حالت میانگین pivot = mid از $O(n\gamma n)$

Space : می‌دانیم که Qsort ، inplace است و از آرایه اولیه استفاده می‌کند و تنها همان را تغییر می‌دهد. پس $O(n)$

Counting Sort ②

Time : ابتدا روی آرایه اصلی پیمایش می‌کند که زمان $O(n)$ دارد، در همین حین آرایه‌ای به اندازه K می‌سازد که انیس هرکدام را با پیمایش افزایش می‌دهد. سپس روی آرایه جدید پیمایش می‌کند و هر عنصر را با قبلی جمع می‌زند و در نهایت روی آرایه اصلی پیمایش می‌کند و انیس های مربوط به هرکدام را قرار می‌دهیم که $O(n)$ زمان می‌برد. پس زمان کلی آن $O(n+K)$ است. اگر $K=n$ همان $O(n)$

Space : علاوه بر آرایه اصلی و یک آرایه به طول K داریم. پس پیچیدگی مکانی آن $O(K)$ است

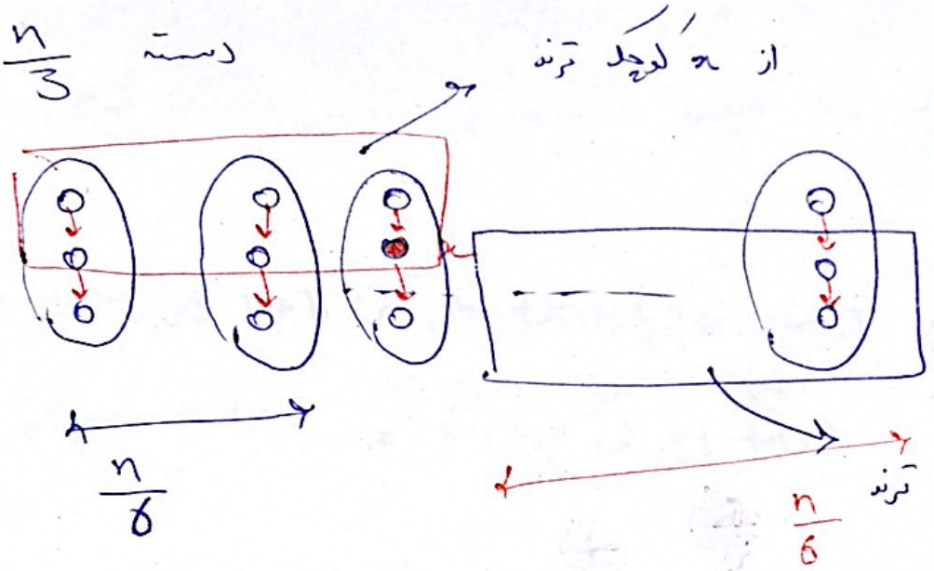
(4) بله!

می‌توان از Radix Sort برای مرتب‌سازی استفاده کرد. اینطوری که الگوریتم counting sort را به روی رقم اُم اجرا کنیم.

Time: اگر d رقم با پایه b داشته باشیم زمان الگوریتم ما $O(d(n+b))$ خواهد بود. پایه ۱۰ است پس $b=10$ همچنین n^2-1 بزرگ‌ترین عدد است و تعداد ارقام برابر $O(\log_b n)$ باشد.

پس Time complexity ما برابر $O(O(\log_b n)\times(n+b))$ حال اگر $n=b$ یعنی پایه برابر n قرار دهیم پیچیدگی زمانی $O(O(\log_n n)\times(n+n)) = O(n)$ الگوریتم ما خطی است.

) مانند مسئله میانه ها به دنبال این اعداد خواهیم گشت. اگر اعداد $x_1, x_2, \dots, x_n$ را در گروه های 3 تایی بریزیم و در $O(1)$ میانه هر گروه را حساب کنیم ($O(1) \times \frac{n}{3}$) که $O(n)$ است. حال باید میانه‌ای را انتخاب کنیم.



حداقل
حداقل اعداد کوچک تر از x $= 2 \times \frac{n}{6} = \boxed{\frac{n}{3}} \longrightarrow$ " بزرگ تر از x $\longrightarrow \frac{2n}{3}$

با انتخاب x ، $\frac{1}{3}$ اعداد از آن بزرگ تر هستند

(6) الف)

• ابتدا میانه هر آرایه را محاسبه می‌کنیم. 3 حالت کلی داریم:

① میانه‌ها برابر باشند

② میانه A بزرگ‌تر از میانه B باشد

③ میانه B بزرگ‌تر از میانه A باشد

① همان میانه جواب است

② جواب در یکی از 2 بخش:
- آرایه A، از اول آرایه تا میانه آرایه A
- آرایه B، از میانه آرایه B تا آخر آرایه B

③ در 2 بخش است:
- آرایه A، از میانه آرایه تا آخر آرایه A
- آرایه B، از اول آرایه تا میانه آرایه B

④ حال نمایندگان بالا را تا جایی ادامه می‌دهیم که تنها 2 عضو در زیرآرایه به وجود آید (تقسیم)

⑤ از فرمول: $(\max(A[0], B[0]) + \min(A[1], B[1])) / 2$ استفاده می‌کنیم (ترکیب)

Time complexity: چون هر سری فضای مسئله نصف شده پس $O(\log n)$

ابتدا با استفاده از یکی از الگوریتم های select میانه را پیدا می کنیم . سپس روی کل آرایه پیمایش

می کنیم و با توجه به میانه، عناصر را به ۲ آرایه مختلف تقسیم می کنیم . در یکی عناصر کوچک تر ا

و در یکی عناصر بزرگ تر. $A \rightarrow \overrightarrow{M}[0, Md]$ میانه $= Md$

$B \rightarrow \overrightarrow{M}[Md, n]$

حال باید از آرایه B که عناصر بزرگ تر از میانه می است K امین عنصر بزرگ ترین

را انتخاب کنیم.

سپس آن را به عنوان pivot تصور می کنیم و عناصر کوچک تر را به قبل و

بزرگ تر را به بعد (partition) حال بین میانه Md و pivot

در Qsort

را مرتب می کنیم.

① پیدا کردن میانه — $O(n \lg n)$

② بخش بندی با میانه — $O(n)$

③ K امین عنصر بزرگ تر = pivot — $O(n)$

④ partition بر اساس pivot

⑤ مرتب سازی $M_{[Md, Pivot]}$ — $O(K \lg K)$

Time complexity : $O(n \lg n + K \lg K)$

(8)

روی کل اعداد پیمایش می‌کنیم تا هزینه کل را محاسبه کنیم.

اگر هزینه مرحله i ام $C(i)$ باشد:

$$\sum_{i=1}^{n} C(i) = \sum_{i=1}^{\lg n} 2^i + \sum i \leq \sum_{i=1}^{\lceil \lg n \rceil} 2^i + n$$

iهایی که توان 2 ندارند — i همه iها را در نظر گرفتیم

$$\Rightarrow = 2 + 2^2 + 2^3 + \dots + 2^{\lceil \lg n \rceil} + n = 2^{1+\lceil \lg n \rceil} - 1 + n \quad (I)$$

$$2^{1+\lceil \lg n \rceil} \leq 2n \quad (II) \xrightarrow{I, II} 2^{1+\lceil \lg n \rceil} - 1 + n \leq 2n - 1 + n$$

$$\Rightarrow 3n - 1 \leq 3n \in O(n)$$

حال برای میانگین گیری تقسیم بر n می‌کنیم پس $O(1)$

با استفاده از روش potential محاسبه می‌کنیم.

$$\Phi_0 = 0 \quad , \quad \Phi_i = 2i - 2^{1+\lfloor \lg i \rfloor}$$

$$\xrightarrow[\text{اول}]{\text{عملیات}} \hat{C}_i = C_i + \Phi_i - \Phi_{i-1} = 1 + 2i - 2^{1+\lfloor \lg i \rfloor} - (2i - 2 - 2^{1+\lfloor \lg (i-1) \rfloor})$$

$$= 1 + 2i - 2^{1+\lfloor \lg i \rfloor} - 2i + 2 + 2^{1+\lfloor \lg (i-1) \rfloor} = 3$$

اگر توانی از 2 باشد $\longrightarrow 2^m = i \longrightarrow \hat{C}_i = i + 2i - 2^{1+m} - (2(i-1) - 2^{1+m-1}) = 3i - 2^{1+m} - 2i + 2 + 2^m$

$$\longrightarrow i - 2^{1+m} + 2 + 2^m = i - 2 \times 2^m + 2 + 2^m = i + 2^m - 2 \times 2^m + 2 = i + 2^m(1-2) + 2$$

$$= i - 2^m + 2 \Rightarrow i - i + 2 = 2 \quad (2^m = i)$$

هزینه سرشکن مربوط به هر کدام از عملیات‌ها برابر 3 می‌باشد.

روش accounting

credit = هزینه واقعی − هزینه سرشکن

هزینه iام $\longrightarrow$ $c_i = \begin{cases} 0 & \text{اگر تا کی از 2 باشد} \\ 1 & \text{اگر تا کی از 2 نباشد} \end{cases}$ $\quad \hat{c}_i = 3$

operation	$\sum \hat{c}$	$\sum \frac{\text{action Sum}}{\text{cost}}$	credit
1	3	1	2
2	6	3	3
3	9	4	5
4	12	8	4
⋮	⋮		
	$3n$		

محاسبه $\sum \hat{c}_i$ در $O(1)$ است، credit هزینه در n عملیات $O(n)$

(9) با استفاده از روش potential :

شمارنده K بیتی داریم که n تا ۱ دارد. همچنین m تا ۰ بعد از آخرین ۱ داریم (می‌تواند $m=0$ باشد) اگر بیت کم ارزش $=0$ باشد، باید از هزینه بیت پرارزش‌تر قرض بگیرد و روی آن پیاده‌اش کند.

$$\hat{C}_i = C_i + \varphi_i - \varphi_{i-1}$$

$$\Rightarrow \hat{C}_i = 1 + m + n + m - 1 + n = 2m$$

در بدترین حالت‌ی تمامی $K-1$ تا صفر بیت‌های کم ارزش‌تر داشته باشیم و در پرارزش‌ترین بیت ۱ داشته باشیم.

در این صورت $O(K)$ هزینه می‌دهیم

$$\Rightarrow$$

$O(K)$ = تحلیل سرشکن decrement

حال برای Reset کردن در بهترین حالت اگر K بیت داشته باشیم $O(K)$ تا عملیات نیاز داریم تا صفر شوند.

n = تعداد ۱ های کل شمارنده K بیتی

$$\hat{C}_i = C_i + \varphi_i - \varphi_{i-1} = K + 0 - n = K - n$$

اگر تعداد n ها ثابت باشد $n \in O(1)$

آنگاه

$O(K)$ = تحلیل سرشکن Reset