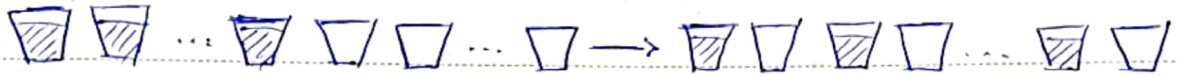
Subject ..... Year ..... Month ..... Day .....

پاسخ تمرین دوم الگوریتم

(1) a) $2n$ لیوان در کنار یکدیگر به صورت ردیف چیده شده‌اند به طوری که n لیوان اول پر از نوشابه و n لیوان دوم خالی‌اند. با کمترین تعداد جابه‌جایی لیوان‌ها، لیوان‌ها را طوری کنار یکدیگر بچینید که یکی در میان پر و خالی باشند.



b) همین مسأله را برای حالتی که لیوان‌ها پر و خالی به شکل تصادفی چیده شده‌اند، حل کنید.

a) فرض کنید لیوان‌ها را از چپ به راست از ۱ تا $2n$ شماره‌گذاری کرده باشیم. حال این جای لیوان شماره‌ی ۲ و شماره‌ی $2n-1$ را عوض کنیم، مسأله به حالت $2(n-2)$ لیوان کاهش می‌یابد. (دو لیوان اول و دو لیوان آخری ترتیب درستی پیدا کرده‌اند و با $2(n-2)$ لیوان باقی‌مانده را مرتب کنیم.) برای لیوان‌های باقی‌مانده هم کافیست همین کار را انجام دهیم. که در این صورت با $\lfloor \frac{n}{2} \rfloor$ حرکت می‌توانیم به ترتیبی که صورت سؤال خواسته برسیم.

هم‌چنین رابطه‌ی بازگشتی این مسأله به شکل زیر است:

$$M(n) = M(n-2) + 1, \quad M(1) = 0, \quad M(2) = 1$$

$M(n)$ ← تعداد جابه‌جایی‌های لازم

⇐ کمترین تعداد حرکت لازم برای رسیدن به ترتیب صورت سؤال $\lfloor \frac{n}{2} \rfloor$ است.

Subject ..... Year ..... Month ..... Day .....

b) برای اثبات درست بودن سوال، با فرض این که لیوان i را از 1 تا 2n شماره گذاری نمایم، در هر خانه با شماره‌ی فرد، حتماً باید لیوان پر داشته باشیم. هم چنین در هر خانه با شماره‌ی زوج، حتماً باید لیوان خالی داشته باشیم. هم چنین هر لیوان یا در جای خودش است یا در جای خودش نیست و اگر لیوانی در جای خودش نباشد، حتماً لیوان دیگری هم داریم که در جای خودش نیست. چون لیوان مذکر جای یک لیوان دیگر را اشغال کرده در نتیجه تعداد لیوان‌هایی که در جای خودشان نیستند، حتماً زوج است. اگر تعداد این لیوان‌ها، 2k باشد، با k حرکت می‌توان آن‌ها را در جای درست‌شان قرار داد (با یکدیگر جابه‌جایشان کرد) $\Leftarrow$ در حالتی که لیوان‌ها به شکل تصادفی چیده شده‌اند، کافیست روی لیوان i بپاشیم، به هر لیوانی رسیدیم که خالی بود و در مکان غلط بود یا برعکس i پر بود و در مکان غلط بود) به دنبال یک لیوان پر که در مکان غلط قرار دارد بگردیم یا برای حالت برعکس به دنبال لیوان خالی‌ای که در مکان غلط قرار دارد بگردیم و جای این دو لیوان را با هم عوض نماییم. $\Leftarrow$ در این حالت با k حرکت می‌توانیم به ترتیب درست برسیم ولی چون جای لیوان i ام غلط را از قبل نمی‌دانیم، باید پیاپی هم این کار را بکنیم.

---

(2) الگوریتم زیر که برای بررسی همبندی گراف است را در نظر بگیرید. (فرض کنید گراف توسط ماتریس مجاورت نمایش داده می‌شود.):

```
Connected ( A[0 ... n-1, 0 ... n-1] ):
    if n = 1 return 1
    else
        if not Connected (A[0 ... n-2, 0 ... n-2]) return 0
        else for j ← 0 to n-2 do
            if A[n-1, j] return 1
            return 0
```

Subject ..... Year ..... Month ..... Day .....

آیا این الگوریتم برای هر گراف غیرجهت دار با $n \geq 0$ رأس به درستی کار می‌کند؟ این پاسخ تان «بله» است، پیچیدگی زمانی این الگوریتم را در بهترین حالت به دست آورید و این پاسخ تان «خیر» است، توضیح دهید چرا.

الگوریتم داده شده غلط است. به عنوان مثال نقض درخت ستاره‌ای را در نظر بگیرید که همبند است (درخت ستاره‌ای: اگر n رأس داریم، درجه‌ی یک رأس، $n-1$ است و باقی رئوس، درجه‌ی 1 دارند.) حال اگر شما راس با درجه‌ی $n-1$، $n-1$ را حذف کنید، ($A[0 \dots n-2, 0 \dots n-2]$ Connected)، 0 می‌شود، پس not آن، 1 شده و الگوریتم 0 برمی‌گرداند و این یعنی درخت ستاره‌ای را ناهمبند در نظر گرفته که غلط است.

---

(3) آیا می‌توان ترتیب سازی درجی (insertion sort) را برای لیست‌های پیوندی (linked lists) هم پیاده سازی کرد؟ اگر بله، آیا ماتند آرایه‌ای ترتیب سازی درجی، باز هم پیچیدگی زمانی، $O(n^2)$ است؟

به قابل پیاده سازی است، فقط برای این که در همان $O(n^2)$ باشد، بخش ترتیب شده لیست را به جای پیمایش از راست به چپ در حالت آرایه‌ای با پیمایش از چپ به راست، پیمایش نمایید.

---

(4) a) بر روی رشته زیر الگوریتم shell_sort را اجرا نمایید.

S, H, E, L, L, S, O, R, T, I, S, U, S, E, F, U, L

(برای انتخاب h، هر دنباله‌ای - به شکل $1 < \dots < h_i < \dots < h_t$ باشد، صحیح است ولی دنباله‌ی $\dots, 121, 40, 13, 4, 1$، بازدهی بهتری دارد.)

b) آیا shellsort، یک الگوریتم ترتیب سازی پایدار (stable) است؟

Subject ..... Year ..... Month ..... Day .....

a) $S_1, H, E_1, L_1, L_2, S_2, O, R, T, I, S_3, U_1, S_4, E_2, F, U_2, L_3$

$h_1 = 13 \longrightarrow \{S_1, E_2\}, \{H, F\}, \{E_1, U_2\}, \{L_1, L_3\}$

$\{L_2\}, \{S_2\}, \{O\}, \{R\}, \{T\}, \{I\}, \{S_3\}, \{U_1\}$

$\{S_4\}$

* روی هر لیست، مرتب سازی درجی را اجرا می کنیم.

⇐ بعد از اتمام مرحله ① :

$E_2, F, E_1, L_1, L_2, S_2, O, R, T, I, S_3, U_1, S_4, S_1, H, U_2, L_3$

$h_2 = 4 \longrightarrow \{E_2, L_2, T, S_4, L_3\}, \{F, S_2, I, S_1\}$

$\{E_1, O, S_3, H\}, \{L_1, R, U_1, U_2\}$

اجرای مرتب سازی درجی روی هر لیست و رشته نهایی بعد از اتمام مرحله ② :

$E_2, F, E_1, L_1, L_2, I, H, R, L_3, S_2, O, U_1, S_4, S_1, S_3, U_2, T$

$h_3 = 1 \longrightarrow$ در این مرحله روی رشته ای که از مرحله قبل به دست آمده ایم

insertion sort را در دستور کار قرار داده و رشته نهایی به شکل زیر می شود:

$E_2, E_1, F, H, I, L_1, L_2, L_3, O, R, S_1, S_2, S_3, S_4, T$

U_2, U_1

Subject ..... Year ..... Month ..... Day .....

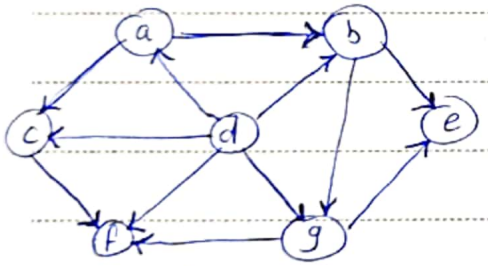
b) خیر، پایدار نیست. چرا که الگوریتم مرتب سازی پایدار، ترتیب نسبی آیتم‌هایی با کلید برابر را به هم نمی‌زند. درحالی که در shell sort این ترتیب ممکن است به هم بخورد. به عنوان مثال دنباله اعداد 5, 2, 3, 4, 2 را در نظر بگیرید. به ازای $h_1 = 4$ داریم:

$$5, 2_I, 3, 4, 2_{II}$$

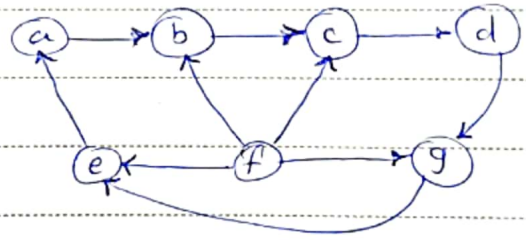
$$h_1 = 4 \rightarrow 2_{II}, 2_I, 3, 4, 5$$

به ازای $h_2 = 1$، آرایه تغییری نخواهد کرد چرا که مرتب شده است. حال اگر به آرایه مرتب شده و آرایه اولی نگاه کنیم می بینیم که در آرایه مرتب شده 2_{II} قبل از 2_I آمده است. درحالی که در آرایه اولیه 2_I قبل از 2_{II} آمده است. پس shell sort پایدار نیست.

5) با استفاده از DFS، هریک از گراف‌های زیر را با topological sort مرتب کنید.

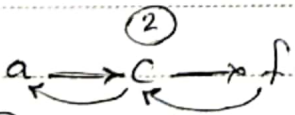


(a)



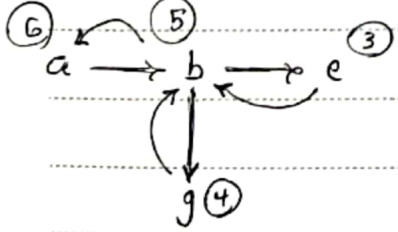
(b)

a) با شروع از رأس a داریم:



حال از ترتیب دیده شدن رئوس را برعکس می‌نویسیم و ترتیب توپولوژیک می‌رسیم:

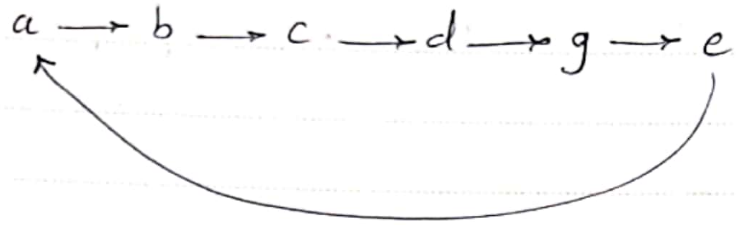
d ④



d, a, b, g, e, c, f

Subject Year .... Month .... Day ....

b) با شروع از رأس a خواهیم داشت:



=> گراف بخش b دور دارد و یک back edge از e به a دارد. در اگر یک دور جهت دار در یک گراف داشته باشیم، توپولوژیکال سورت آن بی پاسخ خواهد بود.

6) تمامی جایگشت های {1, 2, 3, 4} را با استفاده از:

a) الگوریتم bottom-up minimal-change

b) الگوریتم Johnson-Trotter

c) الگوریتم lexicographic-order

تولید کنید.

a) کافی است در هر مرحله، یک عدد را به جایگشت های موجود اضافه کنیم، فقط ترتیب اضافه کردن عدد به جایگشت های موجود یکی در میان از راست به چپ و از چپ به راست خواهد بود:

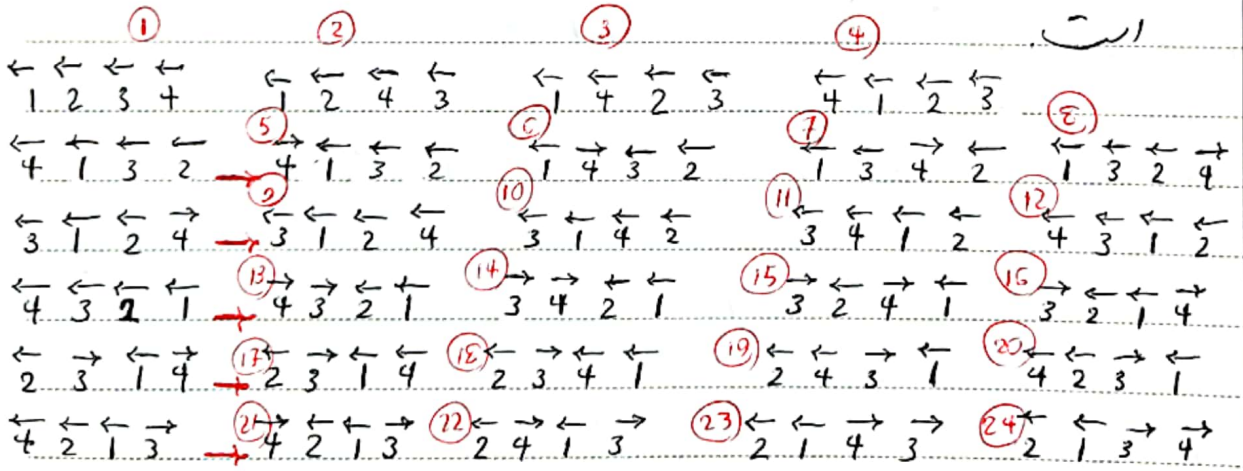
عملیات	جایگشت ها	افزودن
—	1	① افزودن 1
از راست به چپ	12 21	② افزودن 2
از راست به چپ به 12 از چپ به راست به 21	123 132 312 321 231 213	③ افزودن 3

Subject ..... Year ..... Month ..... Day .....

عملیات	حالت ۱				افزودن
از راست به چپ به 123	1234	1243	1423	4123	④ افزودن ۴
از چپ به راست به 132	4132	1432	1342	1324	
از راست به چپ به 312	3124	3142	3412	4312	
از چپ به راست به 321	4321	3421	3241	3214	
از راست به چپ به 231	2314	2341	2431	4231	
از چپ به راست به 213	4213	2413	2143	2134	

b) به تمام اعداد ۴ رقمی به عنوان ورودی نگاه می‌کنیم. دوباره بزرگ‌ترین عدد mobile را پیدا می‌کنیم و آن را با عددهایی که به آن اشاره می‌شود، عوض می‌کنیم. همچنین همه‌ی اعدادی که از بزرگ‌ترین عدد mobile، بزرگ‌تر را با هم عوض کنیم.

تعریف عدد mobile: اگر عددی با عددی دیگر اشاره کند و از آن بزرگ‌تر باشد، mobile است.



Subject ..... Year ..... Month ..... Day .....

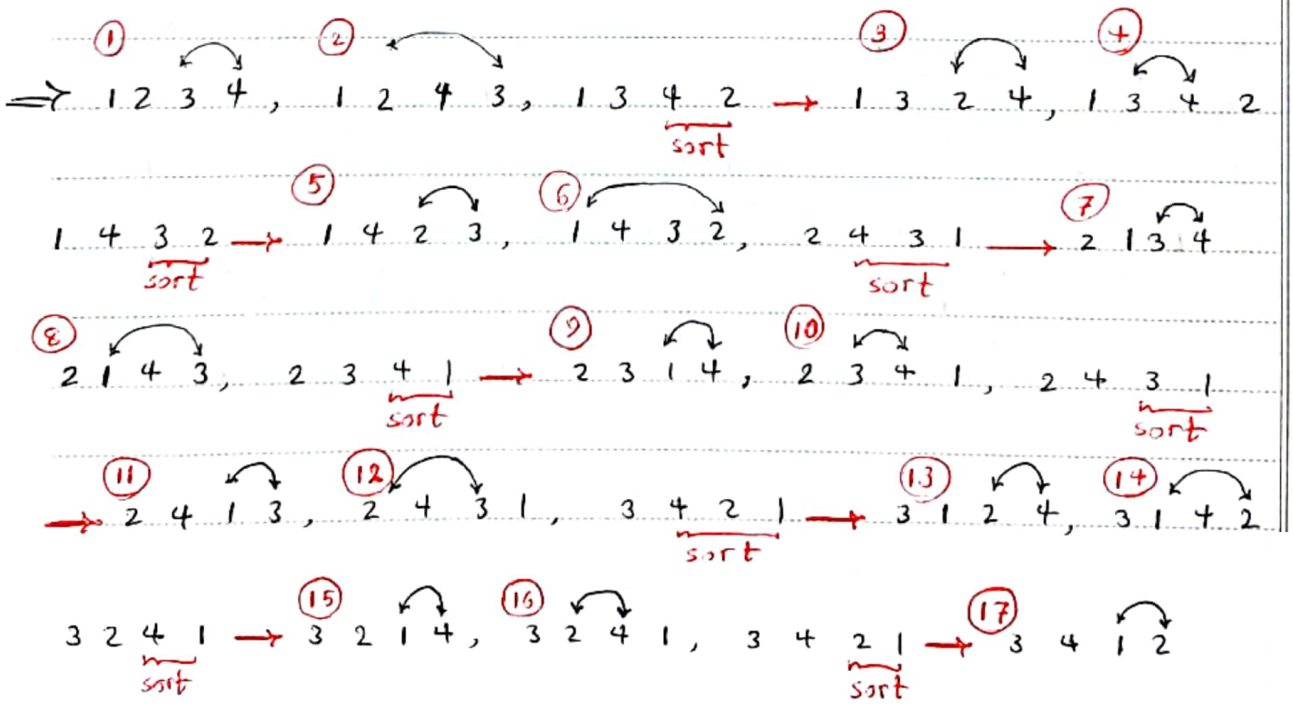
c) ابتدا اعداد داده شده را به ترتیب غیرنزولی مرتب می‌کنیم. سپس باید جایگشت بالاتر متناظر تولید نماییم. این کار را تا زمانی که به جایگشتی برسیم که تمامی ارقام (اعداد) به شکل غیر صعودی مرتب شده باشند، باید ادامه دهیم.

طریقه تولید جایگشت بالاتر: ① سمت راست‌ترین عددی که از عدد بعدی اش کوچک‌تر باشد را پیدا می‌کنیم.

② فرض کنید عدد در مرحله ① در اندیس k یافت شده باشد (n تعداد اعداد است.) حال در بین اعداد $k+1$ تا n، دنبال کوچک‌ترین عددی که از عدد به دست آمده در مرحله ① بزرگ‌تر باشد، می‌گردیم.

③ اعدادی که در مراحل ① و ② به دست آوردیم را جابجا می‌کنیم.

④ اعداد $k+1$ تا n را به شکل غیرنزولی مرتب می‌کنیم.



Subject ..... Year ..... Month ..... Day .....

(18) 3 4 2 1 , 4 3 2 1 → sort

(19) 4 1 2 3 , (20) 4 1 3 2 , 4 2 3 1 → sort

(21) 4 2 1 3 , (22) 4 2 3 1 , 4 3 2 1 → sort

(23) 4 3 1 2 , (24) 4 3 2 1

---

(7) پیاده سازی الگوریتم B. Heap برای تولید تمام جایگشت ها را در نظر بگیرید:

Heap Permute (n): (A یک آرایه از اعداد 1 تا n است.)

```
if n=1
    write A
else
    for i ← 1 to n do
        HeapPermute (n-1)
        if n is odd
            swap A[1] and A[n]
        else swap A[i] and A[n]
```

a) الگوریتم را به شکل دستی برای n = 2, 3, 4 اجرا کنید

b) درستی الگوریتم را اثبات نمایید

c) پیچیدگی زمانی آن را بدست آورید.

a) n = 2 : 1 2 , 2 1

n = 3 : 1 2 3 , 2 1 3 , 3 1 2 , 1 3 2 , 2 3 1 , 3 2 1

n = 4 : 1 2 3 4 , 2 1 3 4 , 3 1 2 4 , 1 3 2 4 , 2 3 1 4 , 3 2 1 4 , 4 2 3 1

2 4 3 1 , 3 4 2 1 , 4 3 2 1 , 2 3 4 1 , 3 2 4 1 , 4 1 3 2 , 1 4 3 2

3 4 1 2 , 4 3 1 2 , 1 3 4 2 , 3 1 4 2 , 4 1 2 3 , 1 4 2 3 , 2 4 1 3

4 2 1 3 , 1 2 4 3 , 2 1 4 3

Subject ..... Year ..... Month ..... Day .....

b) لم: اگر طول آرایهٔ A، n باشد، در صورتی که n زوج باشد، با اجرای الگوریتم، تمام عناصر موجود در آرایه، یک واحد شیفت به راست حلقوی می خورند. (یعنی تمام عناصر به واحد به راست رفته و جای عنصر اول به آخرین عنصر خواهد نشست.) و اگر n فرد باشد، با اجرای الگوریتم تغییری در ترتیب عناصر رخ نمی دهد!

اثبات لم با استقرا: به ازای $n=2$ (پایهٔ استقرا) لم واضحاً درست است. چرا که الگوریتم خود 1 را جواب خواهد داد.

فرض استقرا ← فرض می کنیم که لم به ازای یک $1 \leq i$ درست باشد. حال دو حالت خواهیم داشت: ① $i+1$ زوج باشد. ② $i+1$ فرد باشد.

حالت (I): $i+1$ زوج است. ⇐ i فرد است ⇐ طبق فرض استقرا، با اجرای الگوریتم روی i عنصر اول، i عنصر اول باید بدون تغییر بماند. حال پس از اجرای الگوریتم روی i عنصر اول در حلقهٔ for، فرض کنید در پیمایش k ام باشیم که $1 < k \leq i+1$ در این صورت چون $i+1$ زوج است، جای عنصر آخر و عنصر k ام آرایهٔ A عوض خواهد شد. و در این صورت در حین اجرای حلقهٔ for تا انتهای آن، جای عنصر اول و آخر، عنصر دوم و آخر، عنصر سوم و آخر و... عوض می شود که در نهایت این جابجایی ها منجر به شیفت به راست حلقوی خواهد شد به عنوان مثال برای حالت $i+1=4$ داریم:

1 2 3 4

$i=1 \longrightarrow$ HeapPermute(3) $\xrightarrow{\text{فرض استقرا}}$ 1 2 3 4 $\xrightarrow{\text{swap}}$ 4 2 3 1

$i=2 \longrightarrow$ HeapPermute(3) $\xrightarrow{\text{فرض استقرا}}$ 4 2 3 1 $\xrightarrow{\text{swap}}$ 4 1 3 2

$i=3 \longrightarrow$ HeapPermute(3) $\xrightarrow{\text{فرض استقرا}}$ 4 1 3 2 $\xrightarrow{\text{swap}}$ 4 1 2 3

$i=4 \longrightarrow$ HeapPermute(3) $\xrightarrow{\text{فرض استقرا}}$ 4 1 2 3 $\xrightarrow{\text{swap}}$ 4 1 2 3

⇐ همان طور که مشاهده می شود، یک واحد شیفت حلقوی به راست داریم.

Subject ..... Year ..... Month ..... Day .....

حالت (II): $i+1$ فرد است $\Leftarrow$ n زوج است $\Leftarrow$ طبق فرض استقرا با اجرای الگوریتم روی n عنصر اول، n عنصر اول، یک واحد به راست شیفت حلقوی می‌خوانند.
حال پس از اجرای الگوریتم روی n عنصر اول در حلقهٔ for، چون $i+1$ فرد است $i+1$ بار جای عنصر اول و آخر عوض خواهد شد. که این تغییرات نهایتاً به خنثی شدن شیفت به راست کل می‌شود و باعث می‌شود پس از اتمام حلقهٔ for، آرایه بدون تغییر باقی بماند. به عنوان مثال برای حالت $i+1=3$ داریم:

1 2 3

$i=1 \rightarrow$ HeapPermute(2) $\xrightarrow{\text{فرض استقرا}}$ 2 1 3 $\xrightarrow{\text{swap}}$ 3 1 2

$i=2 \rightarrow$ HeapPermute(2) $\xrightarrow{\text{فرض استقرا}}$ 1 3 2 $\xrightarrow{\text{swap}}$ 2 3 1

$i=3 \rightarrow$ HeapPermute(2) $\xrightarrow{\text{فرض استقرا}}$ 3 2 1 $\xrightarrow{\text{swap}}$ 1 2 3

همان طور که مشاهده می‌شود ترتیب عناصر آرایه تغییری نکرد.

حال با استفاده از نتایج برهان آمده از اثبات لم و استقرا خود الگوریتم را اثبات می‌کنیم.

برهان: برای $n=1$، الگوریتم Heap واضح درست است و پایهٔ استقرا برقرار است.
فرض استقرا: الگوریتم Heap تمام جایگشت‌های یک آرایه با طول n را تولید می‌کند.
حال با توجه به فرض استقرا و نتایج اثبات لم می‌دانیم زمانی که n عنصر اول جایگشت داده می‌شوند، هرکدام از عناصر آرایهٔ A، یک بار در خانهٔ آخر آرایه، قرار خواهند گرفت. از آن جایی که جایگشت‌های یک آرایه را می‌توانیم با برداشتن یک عنصر دلخواه از آن و تولید جایگشت‌های عناصر باقی‌مانده و چسباندن آن عنصر به جایگشت‌ها تولید کنیم، پس الگوریتم Heap برای یک آرایه با طول $i+1$ هم کار می‌کند. (عنصر اول را بری می‌داریم، تولید جایگشت n عنصر با استقرا، اضافه کردن عنصر اول به جایگشت‌ها، عنصر دوم را بری می‌داریم و تولید جایگشت n عنصر باقی مانده با استقرا، اضافه کردن عنصر دوم به جایگشت‌ها و ....) در واقع در این الگوریتم چون تمام عناصر می‌توانند در آخرین مکان آرایه قرار گیرند، می‌توان گفت این الگوریتم برای آرایه به طول $i+1$، در مکان $i+1$ ام اش، عنصر حذف شده را نگه می‌دارد، با فرض استقرا جایگشت‌های n عنصر اول را تولید می‌کند و در نهایت عنصر حذف شده را به جایگشت‌ها اضافه می‌کند و چون تمام عناصر می‌توانند

Subject .... Year .... Month .... Day ....

در آخرین مکان قرار گیرند (اثبات از استقرا هم) و در واقع به نوعی حذف شده است. تمام جایگشت‌های آرایه با $i+1$ عنصر هم تولید خواهد شد.

c) برای تعداد swapها می‌توانیم رابطه بازگشتی زیر را بنویسیم:

$$S(n) = \sum_{i=1}^{n} \left(S(n-1)+1\right) \Rightarrow S(n) = n\,S(n-1) + n \quad , S(1) = 0$$

$$\xrightarrow[n!]{\text{تقسیم طرفین}} \frac{S(n)}{n!} = \frac{n\,S(n-1)}{n!} + \frac{n}{n!} = \frac{S(n-1)}{(n-1)!} + \frac{1}{(n-1)!}$$

$$\xrightarrow{T(n) = \frac{S(n)}{n!}} T(n) = T(n-1) + \frac{1}{(n-1)!} \quad , \quad T(1) = 0$$

$$\Rightarrow T(n) = T(1) + \sum_{i=1}^{n-1} \frac{1}{i!} = \sum_{i=1}^{n-1} \frac{1}{i!}$$

$$T(n) = \frac{S(n)}{n!} \Rightarrow S(n) = n! \sum_{i=1}^{n-1} \frac{1}{i!}$$

$$\Rightarrow T(n) \approx n!\left(e-1-\frac{1}{n!}\right) \in \Theta(n!)$$

---

(8) a) نشان دهید حرکات دیسک‌ها در مسئله برج هانوی می‌تواند برای تولید Gray code نیز استفاده شود.

b) نشان دهید چگونه از Gray Code می‌توان برای حل مسئله برج هانوی استفاده کرد.

a) فرض می‌کنیم دیسک‌ها از کوچک به بزرگ، از 1 تا n شماره گذاری شده باشند. حرکات دیسک k را با k امین بیت نمایش می‌دهیم به طوری که سمت چپ‌ترین بیت

نمایندهٔ کوچک‌ترین دیسک است و سمت راست‌ترین بیت نمایندهٔ بزرگ‌ترین دیسک.
حال چون در ابتدا حرکتی نداریم، تمام بیت‌ها را صفر در نظر می‌گیریم. حال تا رسیدن به پاسخ برج هانوی، اگر دیسک iام حرکت کرد، بیت iام را معکوس می‌کنیم (اگر ۱ است، ۰ می‌کنیم و اگر ۰ است، ۱ می‌کنیم.)

b) هرگاه برای قرار دادن یک دیسک، انتخاب داشتیم، همواره دیسک با شمارهٔ فرد را روی دیسک با شمارهٔ زوج باید قرار دهیم و اگر دیسک با شمارهٔ زوج نداشتیم باید دیسک را در یک میلهٔ خالی قرار دهیم. به طور مشابه یک دیسک با شمارهٔ زوج را باید روی دیسک با شمارهٔ فرد قرار دهیم و اگر دیسک فردی نبود، روی میلهٔ خالی قرار دهیم. در واقع معادل کلاکش با تطابقی است که در بخش a بیان کردیم.

---

9) a) با استفاده از الگوریتم ضرب دهقان روسی، دو عدد 26 و 47 را در هم ضرب نمایید.

b) آیا به لحاظ پیچیدگی زمانی فرقی دارد که n را در m ضرب کنیم یا m را در n (با روش ضرب دهقان روسی)؟

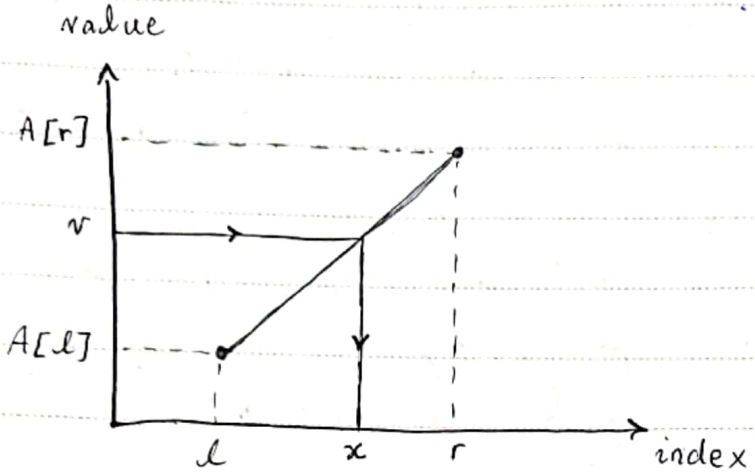
a)

26	47	26
52	23	52
104	11	104
208	5	208
416	2	0
832	1	832
		1222

b) اگر عددی را تقسیم می‌کنیم، n باشد ضرب به روش دهقان روسی $\log_2 n$ مرحله خواهد داشت و اگر عددی که تقسیم بر 2 می‌شود، m، $\log_2 m$ مرحله داریم پس در کل به لحاظ زمانی فرق دارد و اگر عدد کوچک‌تر را تقسیم کنیم، بهتر است.

Subject ..... Year ..... Month ..... Day .....

(10) برای interpolation search مثالی از بهترین حالت ورودی بزنید و نشان دهید که این الگوریتم در بهترین حالت، خطی است.



v ← مقداری که دنبالش می‌گردیم $A[r]$ ← انتهای بازه $A[l]$ ← ابتدای بازه

x ← اندیس v r ← اندیس انتهای بازه l ← آدرس ابتدای بازه

معادله خطی که از $(l, A[l])$ و $(r, A[r])$ می‌گذرد به شکل زیر است:

$$value = \frac{A[r] - A[l]}{r - l}(index - l) + A[l]$$

حال اگر به جای value، v و به جای index، x قرار دهیم خواهیم داشت:

$$v = \frac{A[r] - A[l]}{r - l}(x - l) + A[l]$$

$$\Rightarrow x = l + \frac{(r-l)(v - A[l])}{A[r] - A[l]}$$

حال اگر $A[l] < v < A[r]$ باشد، دو حالت دیگر که $v = A[l]$ باشد، $x = l$ خواهد شد و حالتی که $v = A[r]$ باشد، $x = r$ خواهد شد و حتی در این حالت هم بعد از مقایسه v با $A[x]$ با موفقیت به پایان خواهد رسید (v پیدا شده).

از طرفی داریم:

$$0 < \frac{(r-l)(v - A[l])}{A[r] - A[l]} < r - l$$

Subject ..... Year ..... Month ..... Day .....

بنابراین:

$$0 \le \left\lfloor \frac{(r-l)(v-A[l])}{A[r]-A[l]} \right\rfloor \le r-l-1$$

$$\Rightarrow l \le l + \left\lfloor \frac{(r-l)(v-A[l])}{A[r]-A[l]} \right\rfloor \le r-1$$

$\Leftarrow$ اگر interpolation search در پیش‌بینی خانه کلید را پیدا نکند، اندازه آرایه‌ای که باید در بازگشت بررسی قرار گیرد (برای یافتن کلید) را حداقل 1 واحد کاهش می‌دهد. پس اگر حالتی داشته باشیم که هر بار اندازه باقی‌مانده آرایه، 1 واحد کم شود، بدترین حالت رسیده‌ایم که در این صورت پیچیدگی زمانی $O(n)$ است.

به عنوان مثال آرایه $A = \{0, 1, 2, \dots, n-1\}$ را در نظر بگیرید و فرض کنید که اندیس‌ها هم از 0 شروع می‌شوند. حال اگر بخواهیم با استفاده از interpolation search، $v = n - 1.5$ را پیدا کنیم، در پیمایش kام خواهیم داشت: $l = 0, r = n-k$ و $1 \le k \le n$. ادامه این ادعا را با استفاده از استقرا اثبات می‌کنیم:

اثبات ادعا با استقرا روی k: برای $k=1$، $l=0, r=n-1$ برقرار است و پایه استقرا برقرار است. حال برای فرض استقرا، ادعا را برای $1 < k < n$ درست در نظر می‌گیریم و برای $k+1$ که $1 < k < n$، $l=0, r=n-k$ برقرار است. حال طبق رابطه‌ای که برای x به دست آوردیم خواهیم داشت:

$$x = l + \left\lfloor \frac{(r-l)(v-A[l])}{A[r]-A[l]} \right\rfloor \Rightarrow x = 0 + \left\lfloor \frac{((n-1.5)-0)(n-k)}{(n-1)-0} \right\rfloor$$

$$\Rightarrow x = \frac{(n-1.5)(n-k)}{n-1} = \frac{(n-1)(n-k) - 0.5(n-k)}{n-1}$$

$$= n-k-0.5\frac{n-k}{n-1} < (n-k)$$

Subject ..... Year ..... Month ..... Day .....

حال داریم:

$$x = \frac{(n-1.5)(n-K)}{n-1} = (n-K) - 0.5\,\frac{n-K}{n-1} > (n-K) - \frac{n-K}{n-1}$$

$$\geq (n-K) - 1$$

$$\Rightarrow (n-K)-1 < \frac{(n-1.5)(n-K)}{n-1} < n-K$$

$$\Rightarrow \left\lfloor \frac{(n-1.5)(n-K)}{n-1} \right\rfloor = (n-K)-1 = n-(K+1)$$

$A[x] = A[n-(K+1)] = n-1$ داریم $K < n$ برای $\Rightarrow$

$\Rightarrow l=0,\ r = n-(K+1) \Rightarrow$ برای $K=n$ هم حکم استقرا اثبات شد

---

(11) یک ماتریس $n \times n$ داریم که هر سطر و ستون آن به شکل صعودی مرتب شده است. یک الگوریتم با پیچیدگی زمانی $O(n)$ برای پیدا کردن یک عدد در این ماتریس طراحی نمایید.

در هر مرحله کلید را با آخرین عنصر سطر اول ماتریس مقایسه می‌کنیم (یا $M_{1,n}$ مقایسه می‌کنیم) اگر کلید از آخرین عنصر سطر اول بزرگ‌تر باشد، پس نمی‌تواند در سطر اول ماتریس باشد. (چون باقی عناصر سطر اول حتماً کوچک‌تر از کلیدند) $\Rightarrow$ در این حالت کل سطر اول حذف می‌شود. اگر کلید از آخرین عنصر سطر اول کوچک‌تر باشد، پس نمی‌تواند در آخرین ستون ماتریس باشد (چون باقی عناصر ستون آخر ماتریس، حتماً بزرگ‌تر از کلیدند) این کار را تا زمانی که کلید با عنصر سطر اول و ستون آخر ماتریس باقی مانده برابر شود، یا کل ماتریس حذف شود ادامه می‌دهیم. چون در هر مرحله یک سطر کامل یا یک ستون کامل حذف می‌شود و حتماً n سطر و n ستون داریم $\Rightarrow$ در بدترین حالت این الگوریتم از $O(n)$ می‌شود.