## سوال ۱

به کمک تحلیل مجانبی روابط هر یک از توابع ردیف A نسبت به B را مشخص و اثبات کنید.

| | A | B | $\mathcal{O}$ | $\theta$ | $\Omega$ |
|---|---|---|---|---|---|
| ۱ | $(\log n)^7$ | $n^{0.7}$ | | | |
| ۲ | $2n^{1.5}$ | $2^n$ | | | |
| ۳ | $3^{3n}$ | $3^{4n}$ | | | |
| ۴ | $\log n!$ | $\log n^n$ | | | |
| ۵ | $(\frac{13}{12})^n$ | $(\frac{12}{13})^n$ | | | |
| ۶ | $n^2$ | $4^{\log n}$ | | | |
| ۷ | $n^{0.1}$ | $0.1^n$ | | | |
| ۸ | $\log \log n$ | $\sqrt{\log n}$ | | | |
| ۹ | $n^{\frac{1}{\log n}}$ | $1$ | | | |
| ۱۰ | $\log \sqrt{n}$ | $\sqrt{\log n}$ | | | |

پاسخ:

بخش یک:

$$n^{0.7} \quad vs \quad (\log n)^7$$

$$\log n^{0.7} \quad vs \quad \log(\log n)^7$$

$$0.7 \log n > 7 \log \log n$$

$$(\log n)^7 \in o(n^{0.7})$$

بخش دو:

$$2^n \quad vs \quad 2n^{1.5}$$

$$\log 2^n \quad vs \quad \log 2n^{1.5}$$

$$n \log 2 \quad vs \quad \log 2 + \log 1.5 \log n$$

$$n > 1.5 \log n + 1$$

$$2n^{1.5} \in o(2^n)$$

بخش سه:

$$3^{4n} \quad vs \quad 3^{3n}$$

$$a < b \implies C^a < C^b$$

$$3^{3n} \in o(3^{4n})$$

بخش چهار:

$$\log n! = \log(1 * 2 * 3 * ... * n) =$$

$$\log 1 + \log 2 + \log 3 + ... + \log n \leq \log n + \log n + \log n + ... + \log n$$

$$\log n! \in \mathcal{O}(n \log n)$$

$$n! = n * (n-1) * (n-2) * ... * 2 * 1$$

$$n! \geq n * (n-1) * (n-2) * ... * (\frac{n}{2} + 1) * (\frac{n}{2})$$

$$n! \geq \frac{n}{2} * \frac{n}{2} * ... * \frac{n}{2} \implies n! \geq (\frac{n}{2})^{\frac{n}{2}}$$

$$\log n! \geq \log(\frac{n}{2})^{\frac{n}{2}}$$

$$\log n! \geq \frac{n}{2}(\log n - \log 2) \implies \log n! \geq Cn \log n - Cn$$

$$\log n! \in \Omega(n \log n)$$

$$\implies \log n! \in \Theta(n \log n)$$

بخش پنج:

$$(\frac{12}{13})^n \quad vs \quad (\frac{13}{12})^n$$

$$\frac{13}{12} > 1$$

$$(\frac{13}{12})^n \in \omega(\frac{12}{13})^n$$

بخش شش:

$$4^{\log n} \quad vs \quad n^2$$

$$n^{\log 4} \quad vs \quad n^2$$

$$n^2 \quad vs \quad n^2$$

$$n^2 \in \Theta(4^{\log n})$$

بخش هفت:

$$0.1^n \quad vs \quad n^{0.1}$$

$$n \log 0.1 > 0.1 \log n$$

$$n^{0.1} \in \omega(0.1^n)$$

بخش هشت:

$$\sqrt{\log n} \quad vs \quad \log \log n$$

$$\sqrt{m} \quad vs \quad \log m$$

$$m^{\frac{1}{2}} > \log m$$

$$\log \log n \in o(\sqrt{\log n})$$

بخش نه:

$$1 \quad vs \quad n^{\frac{1}{\log n}}$$

$$1 \quad vs \quad n^{\log_n^2}$$

$$1 \quad vs \quad 2^{\log_n^n}$$

$$1 \quad vs \quad 2$$

$$n^{\frac{1}{\log n}} \in \Theta(1)$$

بخش ده:

$$\sqrt{\log n} \quad vs \quad \log \sqrt{n}$$

$$\sqrt{\log n} \quad vs \quad \frac{1}{2} \log n$$

$$\log n < \frac{1}{4} (\log n)^2$$

## سوال ۲

آرایه ای به طول $n+1$ داریم که عناصر آن شامل اعداد صحیح 1 تا n هستند. هر خانه از این آرایه به یک خانه دیگر از طریق اندیس های آن اشاره می کند. اما بدلیل وجود یک عنصر تکراری در این آرایه، روند کار با این آرایه دچار اختلال شده است. الگوریتمی ارایه دهید که عدد تکراری موجود در این آرایه را پیدا کند. علاوه بر این، شبه کد مربوط به الگوریتم را بنویسید و پیچیدگی زمانی آن را تحلیل کنید.

### مثال

ورودی : $[3,1,3,4,2]$
خروجی : $3$

پاسخ:

اگر هر خانه را یک گره در نظر بگیریم که به خانه دیگری اشاره می کند. یک لیست پیوندی را تشکیل می دهد. همچنین به دلیل وجود عدد تکراری در قسمتی از این لیست یک حلقه وجود دارد و عدد تکراری در ابتدای این حلقه قرار گرفته است.
الگوریتم: برای پیدا کردن حلقه می توان به از الگوریتم زیر استفاده کرد.
روش حل بدین صورت است که از دو اشاره گر استفاده میکنیم.
slow: در هر مرحله یک گام جلو می رود.
fast: در هر مرحله دو گام جلو می رود.
هر دو اشاره گر همزمان از ابتدای لیست شروع به حرکت می کنند تا زمانی که به هم برسند. برای پیدا کردن نقطه ی شروع حلقه،یکی از اشاره گره ها را در ابتدای لیست قرار می دهیم و همزمان حرکت می کنند تا زمانی که به هم برسند و اینگونه ابتدای حلقه پیدا می شود. هزینه زمانی از $\mathcal{O}(n)$ خواهد بود.

```
func(nums)
        slow=fast=0
        while True:
                slow=nums[slow]
                fast=nums[nums[fast]]
                if slow==fast:
                        break
        fast=0
        while True:
                slow=nums[slow]
                fast=nums[fast]
                if slow == fast:
                        return slow
```

## سوال ۳

آرایه ای به طول n داریم که تا اندیس k ام، عناصر به صورت صعودی مرتب شده اند. میخواهیم کل آرایه را مرتب کنیم. از چه الگوریتم مرتب سازی استفاده می کنید؟ آیا مقدار k در انتخاب شما موثر است؟ دلیل خود را بنویسید و زمان اجرای دقیق الگوریتم انتخابی خود را با توجه به n و k بدست آورید و با دیگر الگوریتم های مرتب سازی مقایسه کنید.(حداقل با سه الگوریتم مرتب سازی مقایسه کنید.)

پاسخ :

مرتب سازی حبابی :
در این الگوریتم ترکیب داده ها تاثیری در روند اجرا ندارد و پیچیدگی زمانی همچنان $\mathcal{O}(n^2)$ خواهد بود.
( توجه داشته باشید الگوریتم مرتب سازی حبابی مورد بحث بر اساس الگوریتم ارایه شده در مرجع درس می باشد و الگوریتم مرتب سازی حبابی وجود دارد که ترتیب داده ها در روند اجرا تاثیر دارد.)

مرتب سازی انتخابی :
ترکیب داده های ورودی تاثیری در روند اجرا ندارند و پیچیدگی زمانی همچنان از $\mathcal{O}(n^2)$ می باشد.

مرتب سازی ادغامی :
مرتب سازی ادغامی برای هر نوع ترکیب داده ای به طور یکسان عمل می کند و پیچیدگی زمانی آن از $\mathcal{O}(n \log n)$ می باشد.

مرتب سازی درجی :
در این الگوریتم ترکیب داده ها تاثیر مستقیم در هزینه زمانی دارد و اگر داده ها مرتب یا نیمه مرتب باشند هزینه زمانی کاهش می یابد.

مقدار k می تواند تاثیر گذار باشد.
اگر k عددی بسیار کوچک باشد بدین معنا است که بخش زیادی از آرایه نامرتب است و در این صورت بهتر است از الگوریتم مرتب سازی ادغامی استفاده کنیم که در مقایسه با دیگر الگوریتم های مرتب سازی عملکرد بهتری دارد.
اگر مقدار k عددی بسیار بزرگ باشد این بدان معنا است که بخش زیادی از آرایه مرتب است و در این صورت می توان از هر یک از الگوریتم های مرتب سازی حبابی، انتخابی و یا درجی استفاده کنیم. اما ترجیحا از الگوریتم مرتب سازی در محل [1] استفاده می کنیم که هزینه مکانی کمتری دارد و یا پیاده سازی آن ساده تر است. همچنین با توجه به مساله می توان اولویت های دیگری مانند پایدار بودن الگوریتم را نیز در نظر گرفت.
اگر k عددی بزرگتر از $\frac{n}{2}$ باشد آنگاه مرتب سازی درجی می تواند انتخاب مناسبی باشد زیرا برای k عنصر اول فقط یک مقایسه انجام می شود.و هزینه دقیق زمانی ...

روش دیگر ترکیب چندین مرتب سازی مختلف می باشد بدین صورت که n-k عنصر مرتب نشده را با

[1] inplace

الگوریتم ادغامی مرتب می کنیم و سپس از عملیات ادغام برای ترکیب دو بخش n-k و n استفاده میکنیم. در این صورت هزینه زمانی آن $\mathcal{O}((n-k)\log(n-k)+n)$ خواهد بود.

## سوال ۴

در شهری، یک تعمیرکار ماهر زندگی می کند که به دلیل مهارت و توانایی بالای او در تعمیر خودرو، مشتریان زیادی به او مراجعه می کنند. این تعمیرکار مجموعه بزرگی از پیچ و مهره ها در اختیار دارد که بر اساس اندازه مرتب شده اند. او برای بهره وری بیشتر و سرعت بخشیدن به روند کار به دنبال روشی است که در سریع ترین زمان ممکن بازه پیچ ها با شماره مدنظرش را پیدا کند. الگوریتمی کارا برای کمک به این تعمیرکار ارایه دهید تا بتواند پیچ هایی را که اندازه آن ها بزرگتر از m و کوچکتر از k هستند را به سرعت بیابد. علاوه بر آن، شبه کد مربوط به الگوریتم خود را بنویسید و پیچیدگی زمانی آن را تحلیل کنید.

پاسخ:

پیچ ها که بر اساس اندازه مرتب شده اند آرایه ای از اعداد مرتب است و تعمیرکار قصد جستجو در این آرایه مرتب را دارد پس از الگوریتمی مشابه با الگوریتم جستجوی دودویی برای حل این سوال استفاده می کنیم. هزینه زمانی از $\mathcal{O}(\log n)$ خواهد بود.

```
func(arr,x)
        low=0
        high=Length(arr)
        while low <= high:
                mid= (low+high)//2
                if arr[mid]==x:
                        return mid
                if high==low:
                        return low
                if arr[mid] > x:
                        high = mid-1
                else:
                        low = mid+1
```

## سوال ۵

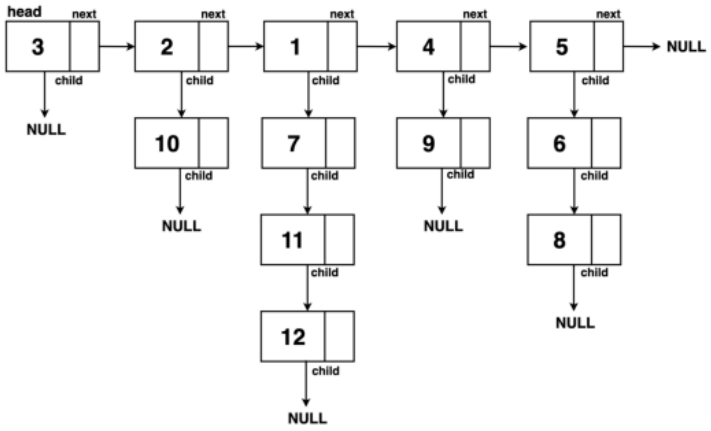
یک لیست پیوندی شامل n گره داریم که هر گره آن دارای دو اشاره گر می باشد:

- next: به گره بعدی اشاره می کند.
- child: به لیست پیوندی دیگری اشاره می کند که گره فعلی راس شروع[۲] آن است.

هر لیست فرزند به صورت صعودی مرتب شده اند. از شما خواسته شده الگوریتمی طراحی کنید که تمام گره ها به صورت مرتب شده در یک سطح قرار بگیرند. علاوه بر آن شبه کد مربوط به الگوریتم خود را بنویسید و پیچیدگی زمانی آن را تحلیل کنید.

## مثال

ورودی :




خروجی : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

[۲] head

پاسخ :

با توجه به این که هر لیست فرعی مرتب شده است. می توان با استفاده از merge آنها را تبدیل به یک لیست مرتب کرد. برای merge نیاز به دو لیست مرتب داریم پس می توان از انتهای لیست شروع کرد و هر دو لیست فرعی را ادغام کرد و تبدیل به یک لیست کرد و این روند را تا زمانی که تنها یک باقی بماند ادامه می دهیم. هزینه زمانی از $\mathcal{O}(n)$ خواهد بود.

```
merge(list1,list2):
        dummy=Node()
        current=dummy()
        while list1 and list2:
                if list1.data <= list2.data:
                        current.child=list1
                        current=list1
                        list1=list1.child
                else:
                        current.child=list2
                        current=list2
                        list2=list2.child
                if list1:
                        current.child=list1
                else:
                        current.child=list2

func(head)
        if not head or not head.next:
                return head
        head2=func(head.next)
        head=merge(head,head2)
        return head
```