پاسخ سوالات تمرین اول

تحلیل الگوریتم

-۱

الف) پیچیدگی زمانی قطعه کد زیر را حساب کنید. (مراحل محاسبه را بنویسید)

هزینه اجرای هر خط را می نویسم و در پایان با هم جمع میکنیم.

```
int j = 0; // Const = C1

for (int i = 0; i < n; i++) { // Repeat n+1
        if (A[i] != B[j]) {
                int k = j; // Const = C2
                while (A[i] != B[k]) { // Max repeat n + 1
                        K++; // C3
                }
                swap(B[j], B[k]);  // Constant time complexity = C4
        }
        j++; // C5
}
```

Time complexity = $C1 + (n+1)[C2 + (n+1)C3 + C4 + C5]$

$= C1 + (n+1)[(n+1)C3 + C6]$

$= C1 + (n+1)^2 C3 + (n+1) C6$

$= (n+1)^2 C3 + (n+1)C6 + C1$

$= O((n+1)^2) = O(n^2)$

و اما چرا در قطعه کد بالا if-else تاثیری روی Time Complexity نداشت؟

به دلیل اینکه زمانی که داریم پیچیدگی زمانی محاسبه میکنیم, دستورات شرطی ممکن است رخ بدهند یا ندهند و ما می خواهیم بدترین حالت را در نظر بگیریم. زیرا در مواقعی که n به سمت بی نهایت میل میکند, تعداد دفعاتی که دستور if باعث میشود قطعه کد درون آن اجرا نشود نسبت به اندازه مسئله کوچک می باشد بنابراین قابل صرفه نظر کردن می باشد.

حلقه ی While درونی نیز در بدترین حالت تا آخر آرایه رو پیمایش میکند. بنابراین از مرتبه n+1 اجرا می شود.

ب)(امتیازی) با توجه به قطعه کد بالا بگویید که الگوریتم ما چه کاری را دارد انجام میدهد؟

این الگوریتم دو آرایه را ورودی می گیرد و یکی را به عنوان Pattern یا الگو ثابت نگه میدارد و اعضای آرایه دیگر را طوری جا به جا میکند تا مانند آرایه Pattern بشود.

مثال:

Before:

A = [1,2,3,4,5,6]

B = [4,2,3,5,1,6]

After:

A = [1,2,3,4,5,6]

B = [1,2,3,4,5,6]

۲ - (امتیازی)

ثابت کنید در قطعه کد زیر در حالت میانگین، تعداد دفعاتی که عبارت Hello world چاپ می‌شود از مرتبه زمانی $O(log\ n)$ است.

```
for (int i = 0; i < n; i++) {
        randomNumber = A random number from 1 to i ();
        if (randomNumber == 1)
                print("Hello world");
}
```

اگر کمی در روند اجرای قطعه کد فوق دقت کنیم درمیابیم که در پیمایش نخست چیزی چاپ نمی شود، در پیمایش دوم حتما عبارت مدنظر چاپ می‌شود، در پیمایش سوم که عدد صحیح رندومی بین ۱و ۲ در randomNubmer است، احتمال اینکه randomNumber برابر با ۱ باشد برابر ۱/۲ است پس احتمال چاپ عبارت مذکور برابر ۱/۲ است، در پیمایش چهارم احتمال چاپ شدن برابر ۱/۳ و به همین ترتیب.

پس برای به دست آوردن میانگین تعداد دفعاتی که عبارت Hello world جاپ میشود باید جمع دنباله زیر که معروف به سری هارمونیک است را به دست آوریم.

$$0\ ,\ 1\ ,\ \frac{1}{2}\ ,\ \frac{1}{3}\ ,\ \frac{1}{4}\ ,\ \dots\ ,\ \frac{1}{n}$$

برای به دست آوردن جمع دنباله فوق، از یک حد بالا استفاده نموده و جمع آن را به دست می‌آوریم و از دو روش آن را حل می کنیم.

روش اول :

در روش نخست برای اینکه حاصل $\sum_{i=1}^{n} \frac{1}{i}$ را به دست آوریم آن را با با جمع یک دنباله بزرگتر مقایسه می کنیم.

$$1 + \frac{1}{2} + \frac{1}{3}$$

$$1 + \frac{1}{2} + \frac{1}{3}$$

$$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \dots + \frac{1}{n} \leq 1 + 1 + \dots + 1 = \log n$$

پس میتوان نتیجه گرفت که جمع سری هارمونیک از $O(\log n)$است.

روش دوم :

در این روش از انتگرال استفاده می کنیم.

$$\sum_{i=1}^{n} \frac{1}{i} \simeq \int_{1}^{n} \frac{1}{x} dx = \ln x$$

پس میتوان نتیجه گرفت که جمع سری هارمونیک از او بزرگ لگاریتم n است.

مرتب سازی

۳- . . . فرض کنید یک آرایه مرتب شده توسط I-index Bubble Sort دارید. می خواهیم یک مقدار مخصوص T را در آن جست و جو کنیم.

الف) الگوریتمی طراحی کنید که با پیچیدگی زمانی $O(n)$ این جست و جو را انجام دهد. (الگوریتم را نوشته و طریقه محاسبه پیچیدگی زمانی آن را شرح دهید)

ساده ترین روش پیاده سازی یک Linear Search می باشد. به این صورت که از عضو اول تا nام را بررسی کنیم, و در صورت یافتن پاسخ آن را اعلام کنیم.

```
for (int i = 0; i < n; i++) // n+1
        if (A[i] = T)
                return True;
return False;
```

$$T = (n + 1)\,C = O(n)$$

همانند سوال اول, مشاهده می کنید که در مقادیر بالا, بودن یا نبودن دستور شرطی تفاوتی ایجاد نمی کند.

ب) حال سعی کنید الگوریتمی ارائه دهید که همین جست و جو را با پیچیدگی زمانی $O(log\ n)$ انجام دهد.

تا به اینجایی کار از ویژگی مرتب بودن آرایه هیچ استفاده ایی نکردیم. برای حل این قسمت نیاز بود تا از این ویژگی کمک بگیریم. میدانیم که آرایه ما بر اساس I-index-Bubble-Sort مرتب سازی شده است. پس می دانیم که در آرایه ما یک عضو وجود دارد که دارای ویژگی زیر است:

$$A_{i-1} > A_i < A_{i+1}$$

به این عضو Pivot آرایه می گوییم و اگر آن را بیابیم می توانیم آرایه را به دو قسمت مرتب شده تقسیم کنیم و روی آن Binary Search بزنیم. با Binary Search در درس مبانی برنامه نویسی آشنا هستید و میدانیم که از اوردر $O(log\ n)$ می باشد.

پس ابتدا سعی میکنیم تا عضو Pivot را بیابیم و سپس آرایه را تقسیم کنیم و روی هر قسمت BS بزنیم.

برای یافتن Pivot می خواهیم از یک روش بهینه استفاده کنیم.

قبول داریم که اگر عضو اول آرایه کوچک تر از عضو آخر آرایه باشد، آرایه ما مرتب شده بر اساس ایندکس 0 می باشد (یعنی آرایه کاملا درست مرتب شده است). پس ابتدا چک میکنیم که اگر $A[0] < A[n]$ آنگاه آرایه ما مرتب است و Pivot برابر 0 است و کافی است Binary Search بزنیم.

اما اگر این شرط برقرار نبود آنگاه در ابتدا عضو میانی آرایه را انتخاب میکنیم و چک میکنیم که آیا در شرط زیر صدق می کند یا نه؟

$$A_{i-1} > A_i < A_{i+1}$$

اگر صدق می کرد, مکان همان عضو را باز می گردانیم. در غیر این صورت آرایه را به دو قسمت تقسیم میکنیم و روی هر دو قسمت مجددا این کار را انجام می دهیم.

اگر این کار را به صورت بازگشتی انجام دهیم آنگاه در پایان کار عضو Pivot را پیدا میکنیم.

```
Find pivot:
If (A == null || A.length == 0) return -1;
If (A.length == 1 || A[0] < A[A.length - 1]) return 0;
Int start = 0, end = A.length - 1;
while(start <= end) {
        Int mid = (start + end) / 2;
        If (mid < A.length && A[mid] > A[mid+1]) return (mid + 1);
        Else if (A[start] <= A[mid]) start = mid + 1;
        Else end = mid - 1;
}
```

```
Binary Search:
int binarySearch(int arr[], int l, int r, int x) {
   if (r >= l) {
      int mid = l + (r - l) / 2;
      if (arr[mid] == x) return mid;
      if (arr[mid] > x) return binarySearch(arr, l, mid - 1, x);
      return binarySearch(arr, mid + 1, r, x);
   }
   return -1;
}
```

ج) ثابت کنید که الگوریتم شما در قسمت ب از اوردر $O(log\ n)$ می باشد.

ج- می دانیم که این تقسیم کردن آرایه و چک کردن شرط، اوردر زمانی الگوریتم را تغییر نمی دهد. از طرفی زمانی که آرایه به دو قسمت تقسیم می شود یک شرط چک می شود که آیا عضو اول آن از عضو آخر آن کوچک تر هست یا نه. در صورت برقرار بودن شرط، میدانیم آرایه مرتب شده است پس Pivot نمی تواند در آن قرار داشته باشد.

در هر قسمت برای یافتن Pivot، آرایه به دو قسمت تقسیم می شود و فقط یکی از آن دو قسمت بررسی می شود. پس اگر سایز آرایه n باشد، در $Log\ n$ آن را پیدا می کند.

همچنین خود Binary Search مشابه فرآیند Pivot عمل میکند، پس آن هم در $Log\ n$ عضو مورد نظر را پیدا می کند.

داریم :

$$T = Log\,n + Log\,n + Log\,n = 3Log\,n = O(log\,n)$$

یک $Log\ n$ برای Pivot داریم و دوتا برای دو Binary Search که بر روی آرایه تقسیم شده زده می شود.

۴ - قطعه کد زیر شبه کد مرتب سازی حبابی است.

```
Bubble_sort(A):
    for i ← n down to 1 do
        for j ← 1 to i do
            if A[j] > A[j+1]:
                swap(A[j], A[j+1])
            end if
```

الف) پیچیدگی زمانی قطعه کد فوق را در بهترین حالت، بدترین حالت و حالت میانگین محاسبه کنید.

پیچیدگی زمانی در هر سه حالت برابر با $O(n^2)$است. تعداد دفعات اجرای حلقه ی دوم در i امین پیمایش حلقه اول را برابر با k در نظر می گیریم و می دانیم که در پیمایش i ام حلقه اول k برابر با i است. پس تعداد دفعات اجرای if از رابطه زیر به دست می آید.

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2} \in O(n^2)$$

همچنین اجرای swap از $O(1)$است و تنها به برقراری شرط if بستگی دارد و تاثیری در پیچیدگی زمانی الگوریتم نمی گذارد.

ب) آیا امکان دارد با ایجاد تغییری در شبه کد فوق، پیچیدگی زمانی بهترین حالت بهبود یابد؟ در صورت پاسخ مثبت، آن تغییر را اعمال کنید.

بله، می توان با ایجاد تغییری در الگوریتم مرتب سازی حبابی، پیچیدگی زمانی بهترین حالت (زمانی که آرایه مرتب شده است) را از $O(n^2)$ به $O(n)$ تغییر داد.

```
Bubble_sort(A):
    for i ← n down to 1 do
    swapped = False
        for j ← 1 to i do
            if A[j] > A[j+1]:
                swap(A[j], A[j+1])
                swapped = True
            end if
        if swapped == False
            break
        end if
```

رشد توابع

۵- موارد زیر را ثابت کنید.

a) $n^{\frac{1}{\lg n}} = \Theta(1)$

تغییر متغیر $\Rightarrow n = 2^x \Rightarrow n^{\frac{1}{\lg n}} = (2^x)^{\frac{1}{x}} = 2$

$2 = \Theta(1)$!

b) $n! = \omega(2^n)$

$$\begin{aligned}
\lim_{n\to\infty} \frac{2^n}{n!} &= \lim_{n\to\infty} \frac{2^n}{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1+\Theta\left(\frac{1}{n}\right)\right)} \\
&= \lim_{n\to\infty} \frac{1}{\sqrt{2\pi n}\left(1+\Theta\left(\frac{1}{n}\right)\right)} \left(\frac{2e}{n}\right)^n \\
&\le \lim_{n\to\infty} \left(\frac{2e}{n}\right)^n \\
&\le \lim_{n\to\infty} \frac{1}{2^n} = 0,
\end{aligned}$$

c) $n! = o(n^n)$

$$\begin{aligned}
\lim_{n\to\infty} \frac{n^n}{n!} &= \lim_{n\to\infty} \frac{n^n}{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1+\Theta\left(\frac{1}{n}\right)\right)} \\
&= \lim_{n\to\infty} \frac{e^n}{\sqrt{2\pi n}\left(1+\Theta\left(\frac{1}{n}\right)\right)} \\
&= \lim_{n\to\infty} O(\frac{1}{\sqrt{n}})e^n \\
&\ge \lim_{n\to\infty} \frac{e^n}{c\sqrt{n}} && \text{(for some constant } c > 0) \\
&\ge \lim_{n\to\infty} \frac{e^n}{cn} \\
&= \lim_{n\to\infty} \frac{e^n}{c} = \infty.
\end{aligned}$$

۶- صحیح یا غلط بودن گزاره‌های زیر را اثبات کنید(در صورتی که گزاره ها غلط هستند مثال نقض کافیست)

a) $f(n) = O(g(n)) \Rightarrow g(n) = O(f(n))$

b) $f(n) = O(g(n)) \Rightarrow 2^{f(n)} = O(2^{g(n)})$

c) $f(n) = O(f(n^2))$

d) $f(n) = O(g(n)) \Rightarrow g(n) = \Omega(f(n))$

e) $f(n) = \Theta(f(\frac{n}{2}))$

f) $f(n) + o(f(n)) = \Theta(f(n))$

a) False, Counterexample: $n = O(n^2)$ but $n^2 \neq O(n)$

b) False. Counterexample: $2n = O(n)$ but $2^{2n} \neq 2^n$

c) False, Counterexample: $f(n) = \frac{1}{n}$ but $\frac{1}{n} \neq O(\frac{1}{n^2})$

d) True. $\exists c, n_0 \; s.t \; \forall n \geq n_0 \Rightarrow f(n) \leq c.g(n)$

$$\Rightarrow \frac{1}{c} f(n) \leq g(n) \Rightarrow g(n) = \Omega(f(n))$$

e) False. Counterexample: $f(n) = 2^{2n}$ but $2^{2n} \neq O(2^n)$

f) True,

Let $g(n) = o(f(n))$, so we have:

$$\exists c, n_0 \; s.t \; \forall n \geq n_0 \Rightarrow 0 \leq g(n) \leq c.f(n)$$

But we want to prove that $f(n) + o(f(n)) = \Theta(f(n))$ so we should prove:

$$\exists c_1, c_2, n_0 \; s.t \; \forall n \geq n_0 \Rightarrow c_1.\, f(n) \leq f(n) + g(n) \leq c_2.\, f(n)$$

۷- مشخص کنید که به ازای هر جفت (A , B) آیا A از Θ , ω ,Ω,o,O تابع B هست یا خیر(مانند ردیف اول)

A	B	O	o	Ω	ω	Θ
n^{2}	n^{3}	yes	yes	no	no	no
$lg^{k} n$	n^{ϵ}	yes	yes	no	no	no
n^{k}	c^{n}	yes	yes	no	no	no
2^{n}	$2^{n/2}$	no	no	yes	yes	no
$2^{2lg\, n}$	n^{2}	yes	no	yes	no	yes
$n!$	$n.2^{n}$	no	no	yes	yes	no
$n^{lg(lg\,(n))}$	$(lg\,(n))^{lg(n)}$	yes	no	yes	no	yes
$n^{lg\,(n)}$	$(log\, n)^{2}$	no	no	yes	yes	no
$\frac{n^{2}}{log\, n}$	$n(log\, n)^{2}$	no	no	yes	yes	no
$n^{\frac{1}{n}}$	$\sqrt{2}^{log\, n}$	yes	yes	no	no	no