چند مثال از نمادهای مجانبی

مثال 17: مرتبه زمانی شبه کد زیر کدام است؟

```
for (i =1; i <= n; i = i + 1)
   for (j = 1;  j <= n ; j = j + i)
         x = x + 1 ;
```

مثال 18: کدام گزینه صحیح است؟

۱) $n^{\frac{1}{10}} \in \Omega(\log n)$

۲) $\log(n!) \in O(\log n)$

۳) $8n^2 + 3n - 4 \in O(n \log n)$

۴) $1^2 + 2^n + \ldots + n^n \in O(n^n)$

مثال 19: مرتبه زمانی قطعه کد زیر کدام است؟

```
i := 2
while  i <= n  do
begin
   i := i^2
   x := x+1
end
```

دو مثال از مرتب سازی

مثال 20: فرض کنید آرایه A[1,..n] داده شده است و هدف مرتب سازی آن به صورت صعودی به کمک الگوریتم مرتب سازی حبابی است.

```
Algorithm Bubble _Sort (A, n)
(a) for i = ١ to  n-١ do
       {
(b)        for j = n downto  i+١ do
           {
(c)           if A[j-١] > A[j]  then
              {
(d)              temp = A[j-١]
(e)              A[j-١] = A[j]
(f)              A[j] = temp
              }
           }
```

الگوریتم فوق نخست کوچک ترین عنصر را پس از n-1 مقایسه در جای اول و عنصر کوچک بعدی را پس از n-2 مقایسه در جای دوم و در نهایت عنصر n-1 ام را با یک مقایسه در جای n-1 قرار می دهد.

در این الگوریتم بعد از تکرار iام، آخرین i عضو آرایه بزرگترین و مرتب شده هستند. در هر تکرار این الگوریتم در قسمت مرتب نشده دنبال بزرگترین عضو است.

الف: تحلیل بدترین پیچیدگی زمانی:

این حالت زمانی اتفاق می افتد که آرایه بصورت نزولی مرتب شده باشد. تعداد اجرای دستورات الگوریتم در جدول زیر خلاصه شده است:

دستورالعمل	تعداد دفعات اجرا
(a)	$n-۱$
(b)	$(n-۱)+(n-۲)+...+۲+۱$
(c)	$(n-۱)+(n-۲)+...+۲+۱$
(d)	$(n-۱)+(n-۲)+...+۲+۱$
(e)	$(n-۱)+(n-۲)+...+۲+۱$
(f)	$(n-۱)+(n-۲)+...+۲+۱$

ب: تحلیل بهترین پیچیدگی زمانی:

این حالت زمانی اتفاق می افتد که آرایه بصورت صعودی مرتب شده باشد. تعداد اجرای دستورات الگوریتم در جدول زیر خلاصه شده است:

دستورالعمل	تعداد دفعات اجرا
(a)	$n-۱$
(b)	$(n-۱)+(n-۲)+...+۲+۱$
(c)	$(n-۱)+(n-۲)+...+۲+۱$
(d)	۰
(e)	۰
(f)	۰

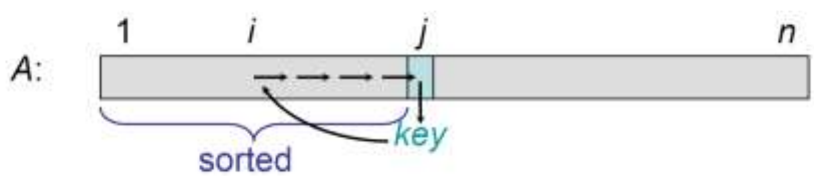
ج: تحلیل پیچیدگی در حالت میانگین:

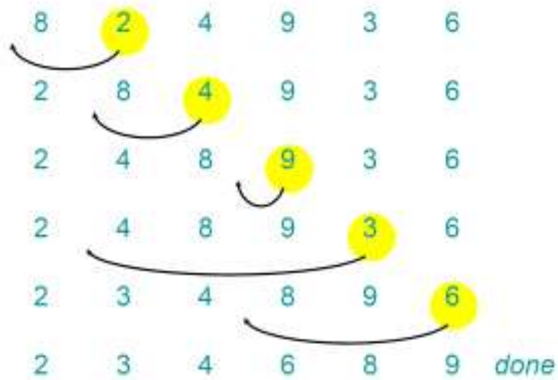
طبق **نتیجه 1** پیچیدگی در این حالت $\theta(n^2)$ است.

مثال 21: تحلیل الگوریتم مرتب سازی درجی

فرض کنید آرایه A[1,..n] داده شده است و هدف مرتب سازی آن به صورت صعودی به کمک الگوریتم مرتب سازی درجی است.

```
INSERTION-SORT (A, n)      ▷ A[1 . . n]
  for j ← 2 to n
       do key ← A[j]
          i ← j – 1
          while i > 0 and A[i] > key
               do A[i+1] ← A[i]
                  i ← i – 1
          A[i+1] = key
```





در این الگوریتم پس از i تکرار، i عنصر اول آرایه مرتب شده هستند. فرض کنید هدف محاسبه تعداد دفعات اجرای دستور مقایسه ای $A[i] > key$ است.

الف: بدترین حالت: بدترین حالت زمانی است که حلقه داخلی همواره اجرا شود. مجموع تعداد اجرای حلقه داخلی $2 + 3 + \cdots n = \frac{n(n+1)}{2} - 1 = O(n^2)$ است.

ب: بهترین حالت: این حالت زمانی اتفاق می افتد که حلقه داخلی اجرا نشود. جمع تعداد اجرای حلقه داخلی $1 + 1 + \cdots + 1 = \Omega(n)$ است.

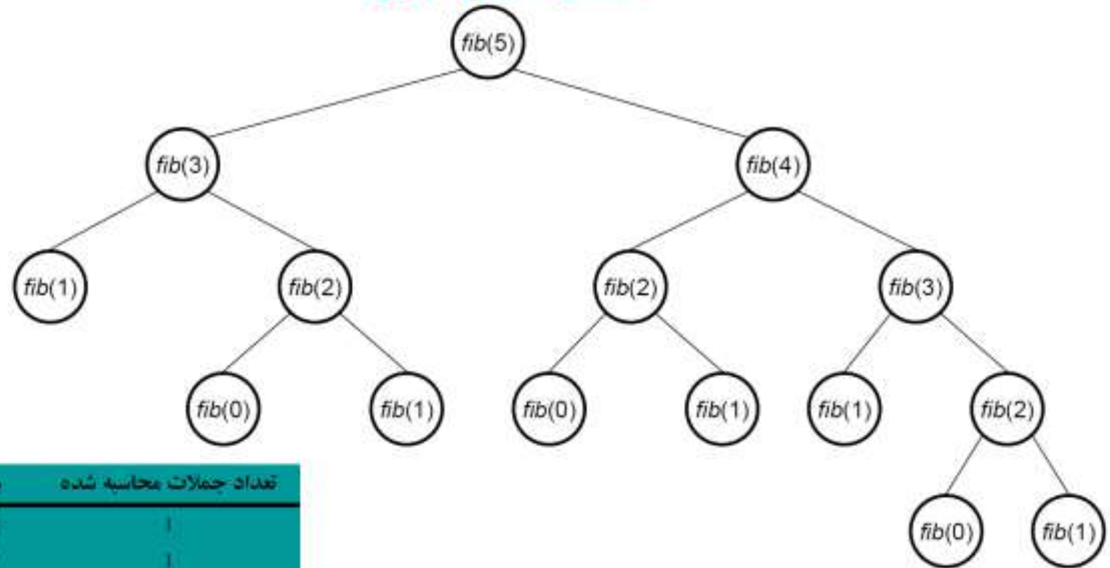
ج: حالت میانگین: در این حالت لازم است در هر مرحله نصف آرایه تا جایی که مرتب شده مقایسه شود. در نتیجه تعداد اجرای حلقه داخلی $O(n^2)$ است.

برخی الگوریتم ها ممکن است بازگشتی باشند. مثال های زیر را ببینید.

مثال 22: تعداد فراخوانی های برنامه زیر را محاسبه کنید:

```
Algorithm Fib (n)
if n ≤ ١ then return (n)
   else
      return (Fib (n-١) + Fib (n-٢))
```

عدم کارآیی



n	تعداد جملات محاسبه شده
0	1
1	1
2	3
3	5
4	9
5	15
6	25

- علت ناکارآیی: محاسبات تکراری
- مثلا در این مثال $fib(2)$ سه بار محاسبه شده است.

فرض کنید $T(n)$ تعداد فراخوانی تابع Fib باشد، پس داریم:

$$T(n) = T(n-1) + T(n-2) + 1$$

$$T(0) = 1$$

$$T(1)=1$$

- هر بار که n به اندازه ۲ واحد افزایش می یابد، تعداد جملات محاسبه شده بیش از ۲ برابر افزایش می یابد، یعنی:
  - $T(n) > 2 * T(n - 2) > 2^{n/2}$ when $n \geq 2$
  - $$\begin{aligned} T(n) &> 2 * T(n - 2) \\ &> 2 * 2 * T(n - 4) \\ &> 2 * 2 * 2 * T(n - 6) \\ &\ldots \\ &> \underbrace{2 * 2 * \ldots * 2}_{n/2 \text{ بار}} * T(0) = 2^{n/2} \end{aligned}$$
- اثبات بوسیله استقراء

- **پایه استقراء:**

$$T(2) = 3 > 2 = 2^{2/2}$$
$$T(3) = 5 > 2.83 \approx 2^{3/2}$$

- **فرض استقراء:**

$$T(m) > 2^{m/2} \qquad , 2 \leq m < n$$

- **گام استقراء:**

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + 1 \\ &> 2^{(n-1)/2} + 2^{(n-2)/2} + 1 \qquad \text{طبق فرض استقراء} \\ &> 2^{(n-2)/2} + 2^{(n-2)/2} = 2 * 2^{(n-2)/2} = 2^{n/2} \end{aligned}$$

مثال 23: تعداد فراخوانی تابع زیر را حساب کنید:

```
int  fact(int n)
{
    if (n == 1)  return 1;
else return  n * fact(n - 1);
}
```

داریم

$$T(n) = T(n-1) + 1.$$

پس

$$T(n) = T(n-1) + 1 = T(n-2) + 2 = \cdots = 1 + (n-1) = O(n).$$

مثال 24: مساله برج هانوی با سه میله A,B,C و n حلقه را در نظر بگیرید و فرض کنید هدف انتقال حلقه ها از A به B فقط از طریق C است، یعنی نمی توان مستقیم حلقه ها را A به B منتقل کرد. اگر n حلقه داشته باشیم تعداد جابجایی لازم را بدست آورید.

گامهای الگوریتم:

1: ابتدا n-1 حلقه را طبق شرایط بالا از میله A به میله C انتقال می دهیم.

2: حلقه آخر را از میله A به حلقه B انتقال می دهیم.

3: n-1 حلقه میله C را با رعایت شرایط بالا به میله B منتقل می کنیم.

بنابراین تعداد کل فراخوانی های تابع برابر است با

$$T(1) = 1$$

$$T(n) = 2T(n-1) + 1 = \cdots = 2 + 2(2 + 2 + \cdots 2^{n-1}) = 2^n - 1.$$

اثبات با استقراء: