محدودیت: زمان* ـ حافظه

← تعداد عملیاتی که الگوریتم انجام می‌دهد.

مسئله ۱) عدد n داده شده است. مقدار عدد nام دنباله فیبوناتچی را خروجی دهید.

$f(n)$: عدد nام فیبوناتچی

$$\begin{cases} f(1)=f(2)=1 \\ f(n)=f(n-1)+f(n-2) & \forall n>2 \end{cases}$$

حل ۱) روش بازگشتی:

```
تابع ← int f(n) {
        if(n<=2)
    +       return 1;
        else
    +       return f(n-1)+f(n-2);
}
```

* فرض کنید T تعداد دفعاتی است که ۲ خط مشخص شده با علامت + اجرا می‌شود. ($T(n)$)

$T(1)=1=f(1)$

$T(2)=1=f(2)$

$T(3)=1+T(2)+T(1)=1+\overbrace{f(2)+f(1)}^{f(3)}=3\geq f(3)$

$T(4)=1+T(3)+T(2)\geq 1+f(3)+f(2)\geq f(4)$

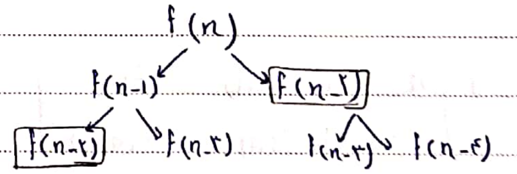
⋮

$T(n)\geq f(n)$ $\quad T_n\simeq\Omega(f(n))$

$f(50)\geq 8\times10^{9}$?

$f(51)\longrightarrow$ تقریباً ۲ برابر

* رشد نمایی



هر $f(n)$ چندین بار محاسبه می‌شود

روش دوم (برنامه نویسی پویا): در این روش مقدار f(i) ذخیره می شود و از آن برای محاسبه مقدارهای بزرگتر استفاده می شود. f آرایه با اندازه n

```
+ F[1]=F[2]=1              ← 2
  for(i=3 → n)             ← n-2+1
+     F[i]=F[i-1]+F[i-2]
  return F[n]
```

سوال: خطوطی که با + مشخص شده‌اند در مجموع چند بار اجرا می شوند؟

$2+n-2+1 = n \simeq O(n)$

سوال: آیا می توان روش دوم را بهینه تر کرد؟

از لحاظ حافظه می توان بهینه کرد.

```
+ a=b=1              2
  for(i=3 → n){      → 3(n-2)
+   temp=b
+   b=a+b
+   a=temp
  }
  return b
```

$3(n-2)+2 = 3n-4 \simeq O(n)$

سوال: آیا باز هم می توان بهتر کرد؟

$\begin{cases} b=a+b \\ a=b-a \end{cases}$ temp حذف می شود ← البته خیلی مهم نیست و عملا بهینه کردن به حساب نمی آید. البته $2n-4$ کمتر از $3n-4$ است اما فرایند خیلی تاثیرگذار نیست. توان اهمیت بیشتری دارد.

راه حل آخر

قضیه

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} f(n+1) & f(n) \\ f(n) & f(n-1) \end{bmatrix}$$

اثبات: با استفاده از استقرا حالت پایه $n=1$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} f(2) & f(1) \\ f(1) & f(0) \end{bmatrix}$$

$f(0) = 0$

برهان: قضیه برای k برقرار است، باید ثابت کنیم برای $k+1$ هم برقرار است.

$$\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}^{k+1} = \begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}^{k} \begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}$$

طبق فرض

$$\begin{bmatrix}F(k+1) & F(k)\\ F(k) & F(k-1)\end{bmatrix}\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix} = \begin{bmatrix}F(k+1)+F(k)=F(k+2) & F(k+1)\\ F(k+1) & F(k)\end{bmatrix}$$

ثابت کنیم $n=2^k$. در این روش برای محاسبه‌ی عدد nام فیبوناتچی کافی است k بار ماتریس را در خودش ضرب کنیم.

$$\underbrace{\left(\left(\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}\right)^2\right)^2\cdots}_{k\text{ بار}}$$

$$n=2^k \rightarrow \begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}^n$$

$$A^n = \underbrace{\left(\left(A^2\right)^2\right)\cdots}_{k\text{ بار}}{}^2$$

```
int F(n) {
        A = [1 1; 1 0]
    for(i:1 → k)
**      A = A x A   ⟵ تعداد ضرب‌ها در این خط انجام می‌شود k بار است
    return A[1,2]
}
```

مثال: برای محاسبه‌ی عدد 3×10^9ام فیبوناتچی:

$$2^{28} = 2^2\times2^{26} > 4\times16^9 > 4\times10^9 > 3\times10^9$$