جلسه سوم حل الگوریتم

تکلیف

مرتبه اجرایی الگوریتم حداکثر صعودی ترکیبات

$O(1) < O(\log_2^n) < O(n) < O(n\log_2^n) < \underbrace{O(n^2) < O(n^3) < \dots}$

ثابت — لگاریتمی — خطی — چندجمله‌ای

$< \underbrace{O(2^n) < O(3^n)} < \dots < O(n!) < O(n^n)$

نمایی

توابع بازگشتی Recursive — ر.ک. جلد ۲، ص ۲۲ تا ص ۲۳

توابع بازگشتی، تابع یا رابطه‌ای است که برای محاسبه مقدار آن در ازای یک ورودی، مقدار تابع در n، $n-2$ و ... مورد نیاز است. یکی از توابع معروف بازگشتی فاکتوریل

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

شرط توقف یا خاتمه

fact(5)
↓
5 * fact(4)
↓
4 * fact(3)
↓
3 * fact(2)
↓
2 * fact(1)

منظور از مرتبه اجرای توابع بازگشتی تعداد صدازدن‌های تابع است. یعنی چند بار خودش را صدا می‌زند. در مثال فوق وقتی $n=5$ است تابع 5 بار خودش را فراخوانی می‌کند پس $O(n)$ fact(5)

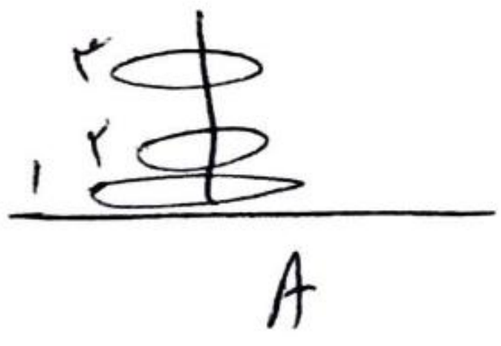
مثال برجهای هانوی Hanoi

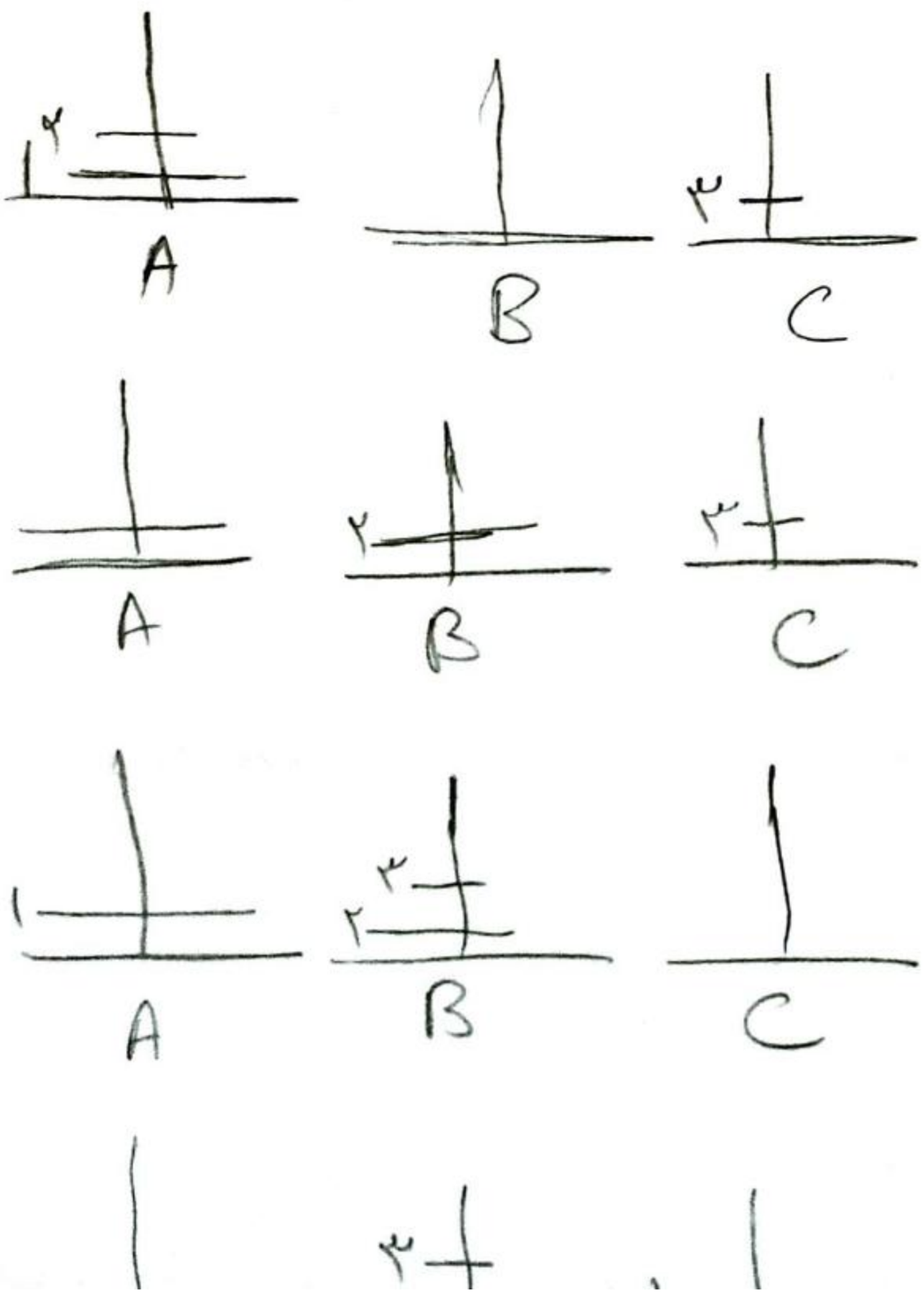
۳ برج با نامهای A و B و C وجود دارد که در ابتدا بر روی برج A ، n حلقه با اندازه‌های متفاوت قرار دارد که بزرگترین حلقه در پایین و به ترتیب تا کوچکترین حلقه قرار دارد. باید این n حلقه از برج A با کمک برج B به برج C انتقال پیدا کنند.

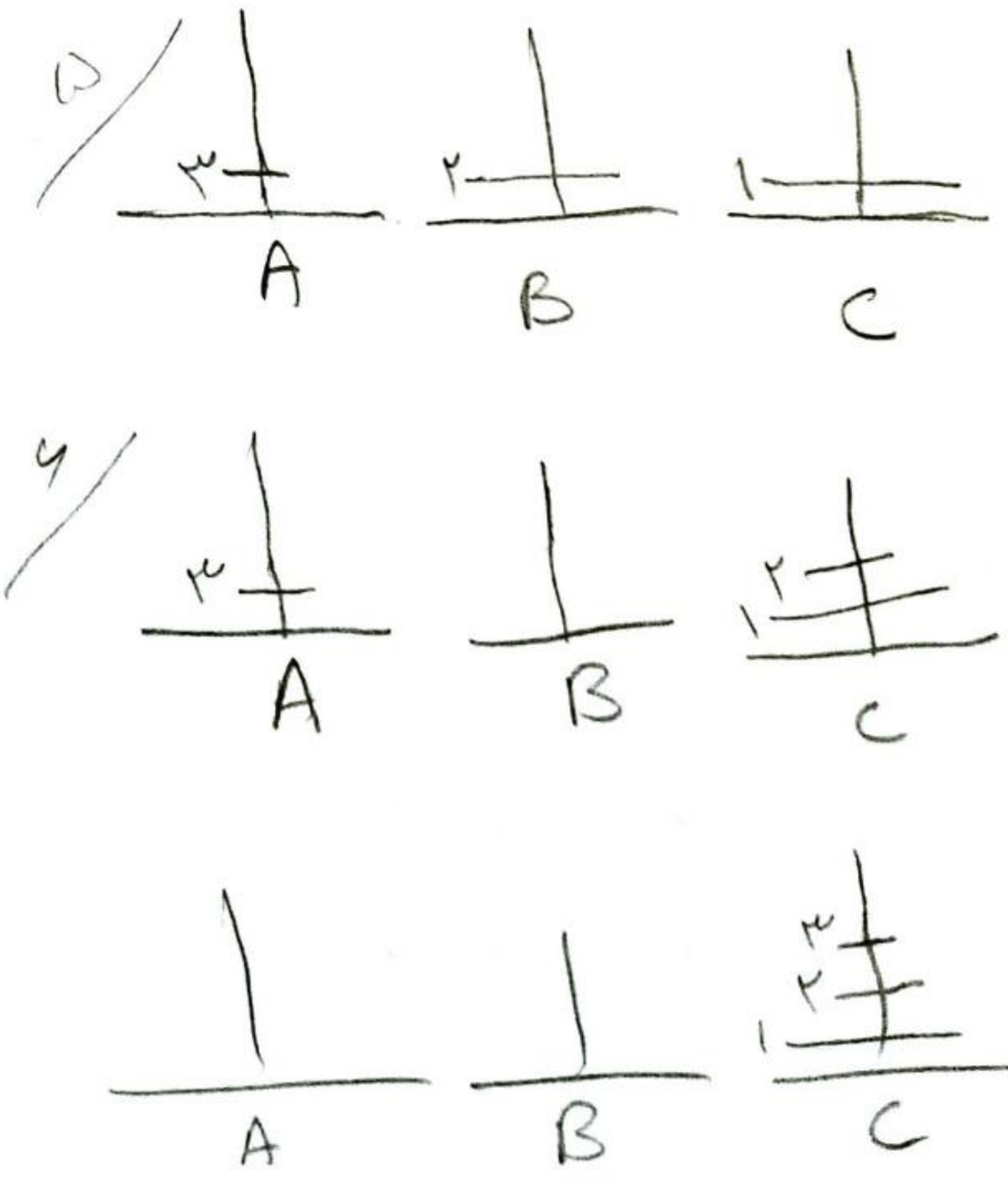
۲ شرط داریم:

۱ـ در هر انتقال فقط یک حلقه منتقل شود

۲ـ حلقه بزرگ هیچگاه روی حلقه کوچک قرار نگیرد





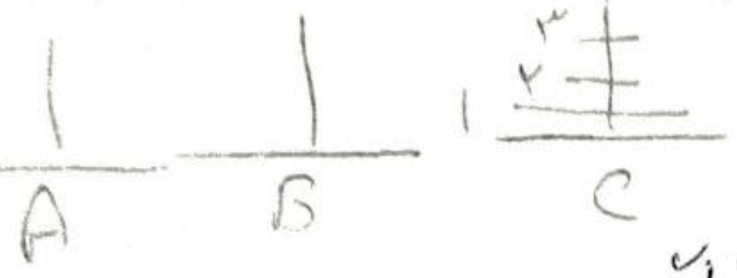


مثال: مسئله برج های هانوی را به ۳ مسئله کوچکتر زیر تقسیم کنید.

۱- انتقال n-1 حلقه از میله A با کمک میله C به میله B.

۲- انتقال یک دیسک از میله A به میله C.

۳- انتقال n-1 دیسک از میله B به کمک میله A به میله C.



تعداد انتقال دیسک برجهای هانوی را برای n حلقه برابر T(n) در نظر میگیریم داریم.

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

برای $n = 3$

$$T(3) = 2T(2) + 1 = 2 \times 3 + 1 = 7$$

$$T(2) = 2T(1) + 1 = 2 \times 1 + 1 = 3$$

می‌توانیم بنویسیم $T(1) = 2^1 - 1 = 1$.

$$T(2) = 2^2 - 1 = 3$$

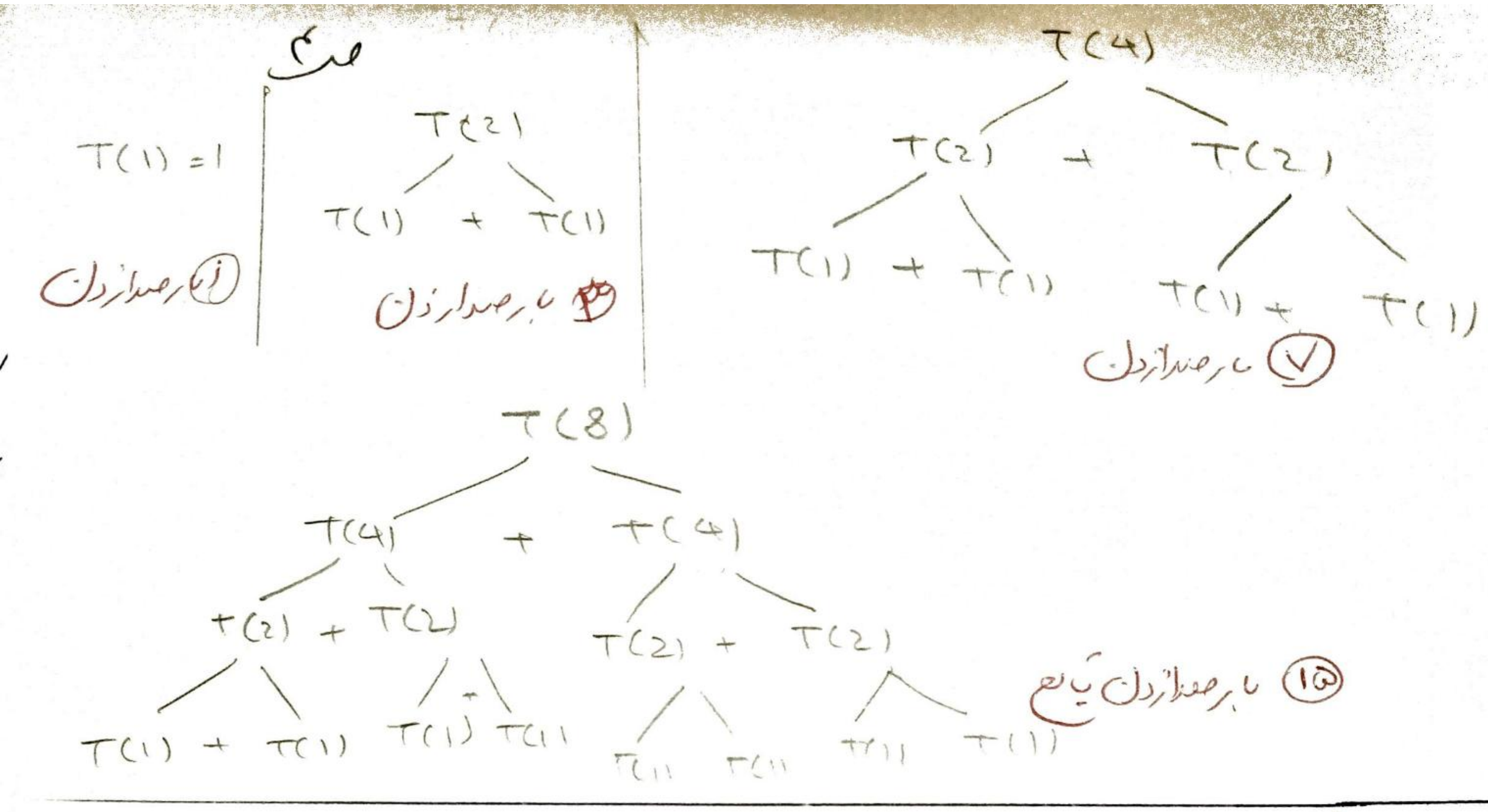
$$T(3) = 2^3 - 1 = 7$$

$$T(n) = 2^n - 1 \Rightarrow O(2^n)$$

مثال ته صفحه ۳۲

مرتبه اجرایی تابع زیر

```
int T(int n)
{
  if(n<=1)
     return 1;
else
   return  T(n/2) + T(n/2)
}
```



رابطه بازگشتی جمع اعداد n

$$F(n) = 1 + 2 + 3 + 4 + \dots + n-1 + n$$

$F(1) = 1$

$F(2) = 1+2 = 3 \Rightarrow F(1) + 2$

$F(3) = 1+2+3 = 6 \Rightarrow F(2) + 3$

$F(4) = 1+2+3+4 = 1 \Rightarrow F(3) + 4$

$$F(n) = F(n-1) + 1$$

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

$$T(n) \frac{n(n+1)}{2} \Rightarrow O(n^2)$$

HW رابطه بازگشتی تابعی جهت محاسبه فاکتوریل عدد صحیح n برای عدد صحیح n

H.W رابطه‌ی بازگشتی یافتن باقیمانده عدد صحیح n بر عدد صحیح n

H.W رابطه‌ی بازگشتی یافتن ضریب ترکیب $\binom{n}{m}$

دنباله فیبوناچی ص۲۷ ت

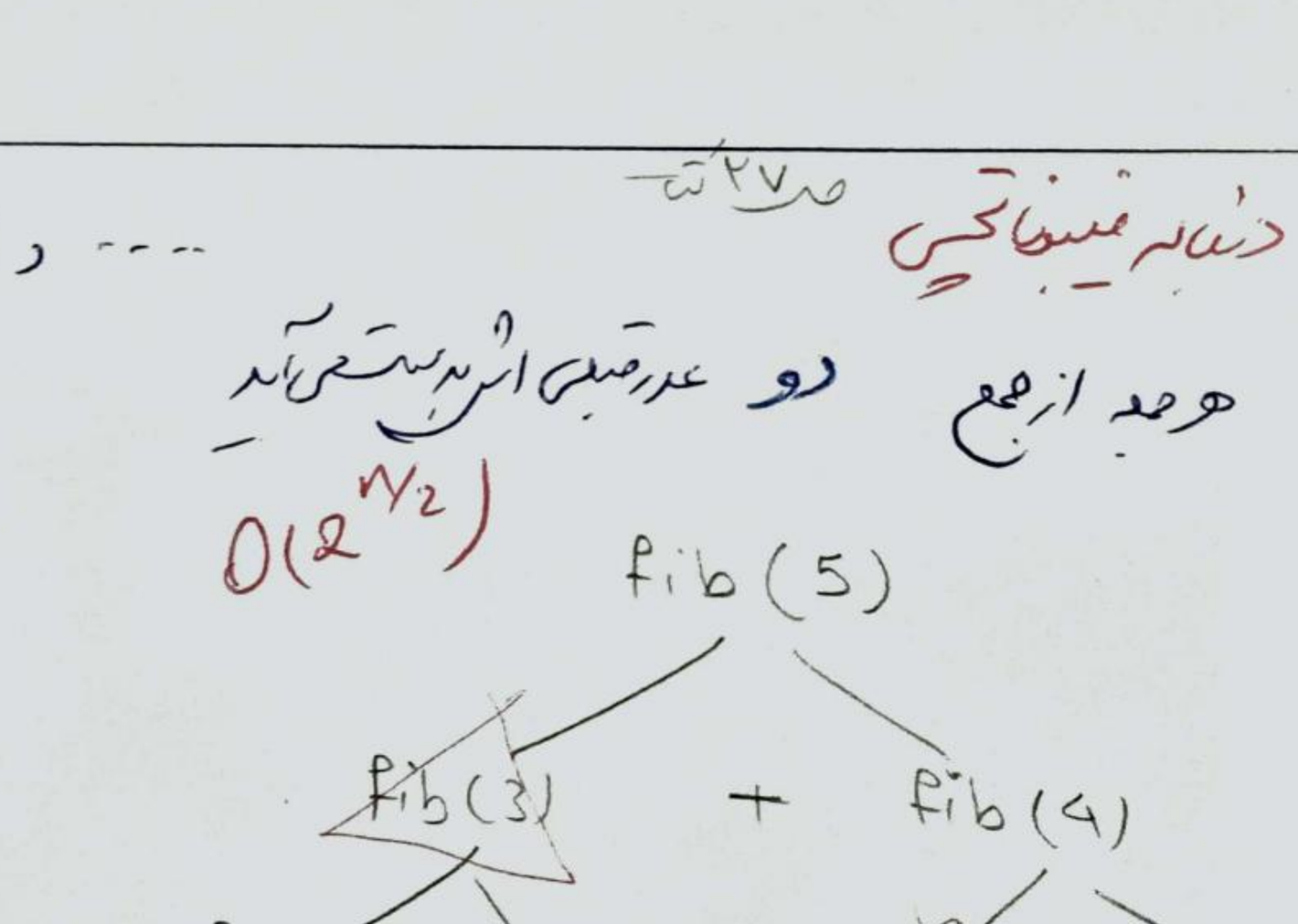
0, 1, 1, 2, 3, 5, 8, ...

هر جمله از جمع دو عدد قبلی اش به دست می آید

$$f(n) = \begin{cases} 0 & n=0 \\ 1 & n=1 \\ f(n-1)+f(n-2) & n \geq 2 \end{cases}$$

$O(2^{n/2})$

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



۱۵ بار صدا زده شده

جلد ۶

در این مثال دیدیم که هر بار جمله ها شکسته می شود و به جمله های کوچکتر تقسیم می شوند
پس از روش تقسیم و غلبه استفاده کردیم بعد استفاده به صورتی که این شکل را به عنوان کوچکتر (زیر مسئله ها)
می بینیم . حال از یک آرایه دنباله فیبوناچی را پیاده سازی می کنیم.

روش بالا به پایین Top/down

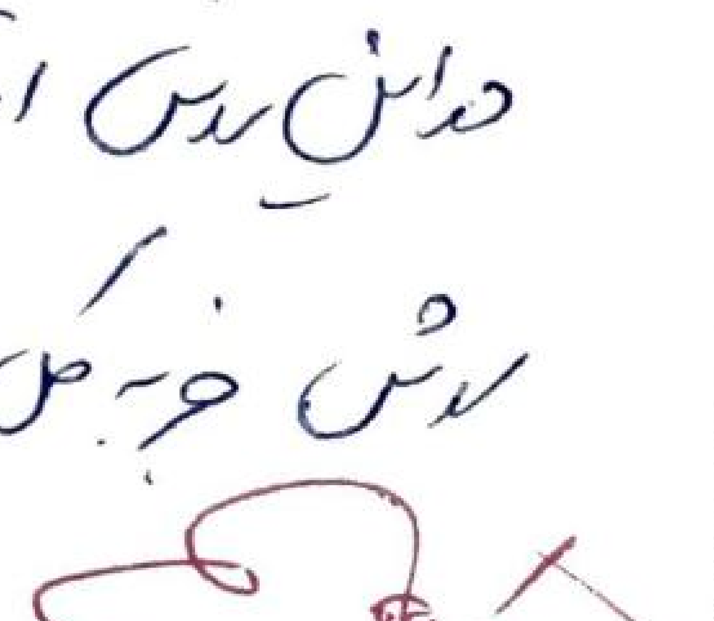
F[5] = ?

```
int fib (int n)
{
    int i, f[0..n];
    f[0] = 0;
    if(n>0) {
     f[1] = 1;
      for (i=2; i<=n; i++)
          f[i] = f[i-1] + f[i-2];
    }
    return f[n];
}
```

$O(n)$

f[0] = 0
f[1] = 1
f[2] = f[1] + f[0]
f[3] = f[2] + f[1]
f[4] = f[3] + f[2]
f[5] = f[4] + f[3]

در این روش ابتدا عنصر های کوچکتر حل می شوند مثل F[1], F[2] تا به عنصر بزرگتر می رسیم F[5]
روش حل از پایین به بالا که به آن روش پویا یا Dynamic prog می گوییم.



الگوریتم فیبوناچی بازگشتی تقسیم و غلبه دارای مرتبه $O(2^{\frac{n}{2}})$
الگوریتم فیبوناچی با روش پویا دارای مرتبه $O(n)$

که روش پویا بار ایجاد شده کمتری است و کمتر تحمل می کند

روش‌های پایین به بالا - جهت بازگشتی پیاده‌سازی می‌شود

۱۰
جستجوی خطی

در برابر آرایه‌ای نامرتب. این جستجو انجام می‌شود و روش جستجو به این ترتیب است که از اندیس صفر شروع (حلقه) با عنصر اول مقایسه می‌شود در صورت عدم تساوی با عنصر دوم مقایسه می‌شود

بدترین حالت: عنصر مورد جستجو آخرین عنصر آرایه باشد. n مقایسه
بهترین حالت: عنصر مورد جستجو اولین عنصر آرایه باشد. 1 مقایسه
در حالت متوسط: $\frac{n+1}{2}$ مقایسه

مرتبه اجرایی $O(n)$

```
int SeqSearch (items S[n], items x)
{
    int location;
    while(location<=n      S[location]!=x)
        Location = location+1;
    if (location>n)
      return 0;
    else
      return location;
}
```

مثال: در آرایه زیر عدد 43 را جستجو کنید به روش جستجوی ترتیبی.

1	2	3	4	5	6	7	8	9
21	7	39	41	15	43	10	9	11

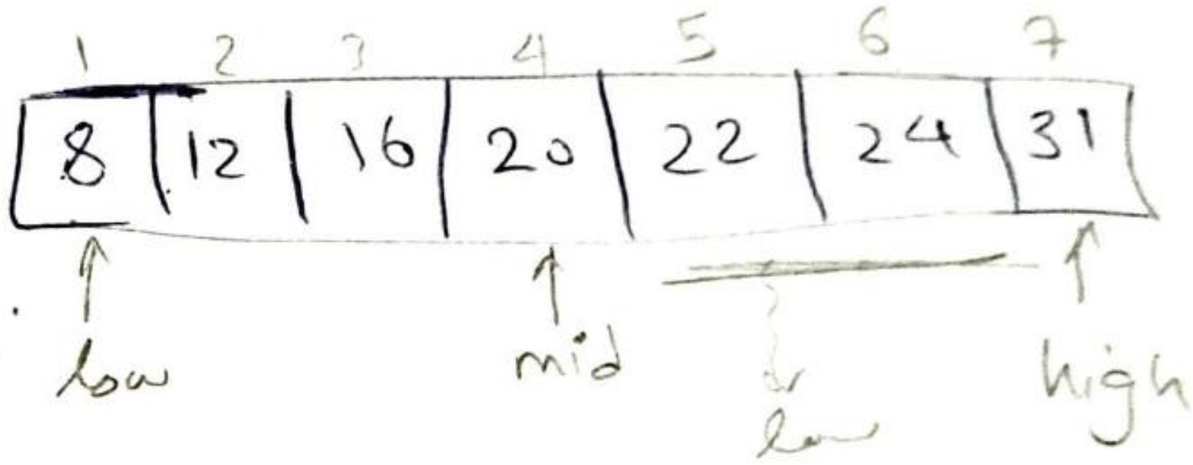
برای پیدا کردن 43 نیاز بود 6 مقایسه هیم اگر 43 آخرین عنصر بود نیاز بود 9 مقایسه و از اینجا می‌توان مقایسه روش مناسبی نیست برای همین از روش جستجوی دودویی استفاده می‌کنیم

جستجوی دودویی صد ۱۰۴

در این روش عنصر مورد جستجو (کلید) با عنصر وسط آرایه مقایسه می‌شود. اگر کلید برابر با عنصر وسط باشد عمل جستجو پایان می‌یابد. اگر کلید بزرگتر از عنصر وسط باشد عمل جستجو در نیمه بالایی آرایه و اگر کلید کوچکتر از عنصر وسط باشد عمل جستجو در نیمه پایینی انجام می‌شود.

مرتبه اجرایی $O(\log_2^n)$

مثال: جستجوی عدد 24



$mid = \frac{1+7}{2} = 4$

$x[4] = 20$

$m > x[mid]$

$24 > x[4]$

$low = mid + 1 = 4 + 1 = 5$

$high = 7$

$mid = \frac{7+5}{2} = 6$

$m = x[6]$

با ۲ مقایسه پیدا شد.

```
int bsearch(int x[], int n, int m)
{
    int low, high, mid;
    low = 0; high = n-1;
    while (low <= high){
        mid = ((low+high)/2;
        if(m < x[mid])
            high = mid-1;
        else if(m > x[mid])
            low = mid+1;
        else
            return mid;
    }
    return -1;
}
```