مرکز آموزش الکترونیکی دانشگاه علم و صنعت ایران

طراحی و تحلیل الگوریتمها

فصل سوم: روشهای تقسیم و غلبه

(جلسه اول)

اهداف یادگیری

- آشنایی با اصول کلی روشهای تقسیم و غلبه
- آشنایی با نحوه محاسبه پیچیدگی زمانی روشهای تقسیم و غلبه
- بررسی مسائلی چون:
  - جستجوی دودویی
  - یافتن بزرگترین و کوچکترین عنصر در آرایه
  - ضرب چند جمله ایها
  - مرتب سازی ادغامی (Merge Sort)
  - مرتب سازی سریع (Quick Sort)
  - انتخاب k امین عنصر در یک آرایه نامرتب
  - ضرب ماتریسها به روش استراسن

کلیات

- این روش از ضرب المثل معروف تفرقه بینداز و حکومت کن نشات گرفته است
- در این روش مسئله ای به اندازه n، به k $(1 < k \leq n)$ زیر مسئله تفکیک می شود.
- هر یک از زیر مسائل جداگانه حل شده و پاسخ هر کدام تعیین می گردد.
- سپس از ترکیب این پاسخها جواب نهایی مسئله تعیین می شود.

کلیات(ادامه)

- اگر اندازه هر یک از زیر مسائل بزرگ باشد، به طوریکه حل آنها بدیهی نباشد، هر یک از آنها نیز با همین روش حل می شوند.
- هر یک از زیر مسائل (اگر اندازه آنها بزرگ باشد) به شیوه بازگشتی حل می شوند.
- مثال: تعداد اعداد اول بزرگتر مساوی ۰ و کوچکتر از ۱۰۰ چند تاست؟
- اندازه مسئله (n): تعداد اعدادی که باید بین آنها به دنبال عدد اول بگردیم. در این مسئله ۱۰۰

کلیات(ادامه)

- بزرگ بودن مسئله: پاسخ بدیهی نباشد. ۱۰۰ عدد فوق، برای تعیین جواب باید جستجو شوند. بنابراین پاسخ بدیهی نیست.
- کوچک بودن مسئله یا بدیهی بودن پاسخ: اگر صرفاً یک عدد در اختیار داشته باشیم، به سرعت می توان گفت اول است یا خیر. مانند ۱۳ یا ۵۶
- تعیین کوچک بودن یا بزرگ بودن اندازه مسئله: بستگی به نوع مسئله دارد. ولی معمولاً مسئله ای به اندازه ۱، به عنوان مسئله کوچک شناخته می شود.

کلیات(ادامه)

- تقسیم مسئله: تفکیک ورودی به بخشهای کوچکتر (معمولاً) جدا از هم. در مثال قبل می توان اعداد را به ۱۰ دسته به صورت زیر تقسیم کرد:(به جای ۱۰ می توان از هر عدد دیگری استفاده کرد)
- 0-9, 10-19, …, 80-89, 90-99
- حل هر زیر مسئله: هر زیر مسئله جدا از سایر زیر مسائل بررسی می شود: به عنوان مثال می خواهیم تعداد اعداد اول در بازه ۹-۰ (دسته اول) را حساب کنیم.

کلیات(ادامه)

- چون باز هم جواب بدیهی نیست آن را به ۱۰ دسته تقسیم می کنیم که اینبار هر دسته دقیقاً ۱ عدد دارد و به سرعت می توان تشخیص داد اول است یا خیر
- ترکیب: پس از حل هر زیر مسئله، برای بدست آوردن جواب نهایی، این پاسخها باید با هم ترکیب شوند.
- ترکیب در هر مسئله ای ممکن است متفاوت باشد.
- در مثال ارائه شده، ترکیب عبارتست از عملگر جمع (+)

یک الگوریتم کلی برای روش تقسیم و غلبه

```
Algorithm  D&C(low, high)
{  if small(low, high) then
     return G(low, high)
   else {
    mid=Divide (low, high)
    return combine(D&C(low, mid),
    D&C(mid+1, high))
    }
}
```

محاسبه پیچیدگی زمانی الگوریتمهای تقسیم و غلبه

- فرض کنید T(n) زمان اجرای الگوریتم D&C(1,n) باشد. (منظور از n اندازه مسئله است.)
- همچنین فرض کنید f(n) زمان ترکیب پاسخها باشد. (تابع combine در الگوریتم قبل)
- در این صورت زمان اجرای D&C(1,mid) که در آن mid=n/2 است، عبارت خواهد بود از T(n/2)

محاسبه پیچیدگی زمانی الگوریتمهای تقسیم و غلبه(ادامه)

- و چون در الگوریتم قبل ۲ بار D&C() و هر بار با اندازه ورودی n/2 فراخوانی شده بود، پس
- $T(n)=T(n/2)+T(n/2)+f(n)=2T(n/2)+f(n)$
- و اگر n عدد کوچکی باشد و فرض کنیم پاسخ آن در زمان g(n) قابل محاسبه باشد، آنگاه $T(n)=g(n)$

چند نکته

- در یک الگوریتم تقسیم و غلبه، همیشه مسئله به ۲ قسمت تقسیم نمی شود. مانند مثال تعیین تعداد اول که مسئله به ۱۰ قسمت تقسیم شد.
- در این حالت باز هم هر یک از زیر مسایل جداگانه و به صورت بازگشتی حل می شوند.
- اگر همان مثال تعیین اعداد اول را این بار برای بازه 1- n در نظر بگیریم:

$$T(n)=10\,T(n/10)+1$$

$$T(n)=g(n) \quad n=1$$

جستجوی دودویی

- مسئله: آرایه مرتب شده (فرض کنید به صورت صعودی) A[1..n] و کلید x مفروض است. هدف یافتن x در این آرایه است به قسمی که اگر بود، اندیس خانه ای که x در آن است مشخص شود و در غیر اینصورت ۱-
  - راه حل اول: جستجوی خطی با O(n) !
  - راه حل دوم: استفاده از روش تقسیم و غلبه (جستجوی دودویی)

جستجوی دودویی(ادامه)

- ابتدا عنصر وسط آرایه را تعیین و با X مقایسه می کنیم:
  - اگر عنصر وسط برابر با X باشد، پاسخ مسئله مشخص شده و اندیس خانه وسط به عنوان جواب تعیین می شود.
  - اگر X بزرگتر از عنصر وسط باشد، پس به طور حتم X نمی تواند در نیمه سمت چپ آرایه باشد (چون آرایه مرتب است) لذا جستجو با همین روش در نیمه سمت راست ادامه می یابد.
  - در غیر اینصورت X در نیمه سمت چپ است، و نیمه سمت چپ باید به همین شیوه بررسی شود
- فرایند فوق آنقدر ادامه پیدا می کند تا یا X پیدا شود و یا اینکه بازه مورد جستجو خالی شود که در این صورت X در آرایه وجود ندارد

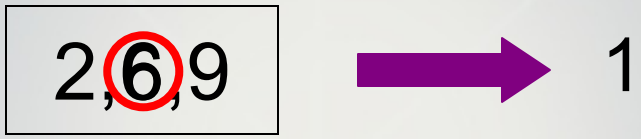
الگوریتم جستجوی دودویی

```
int bin-search(int A[],int n,int x,int low, int high)
{ if (low>high) return -1;
  else{
      int mid=(low+high)/2;
       if(A[mid]==x) return mid;
      else if(A[mid]<x)
            return bin-search(A,n,x,mid+1,high);
      else return bin-search(A,n,x,low,mid-1);
       }
}
```

مثال

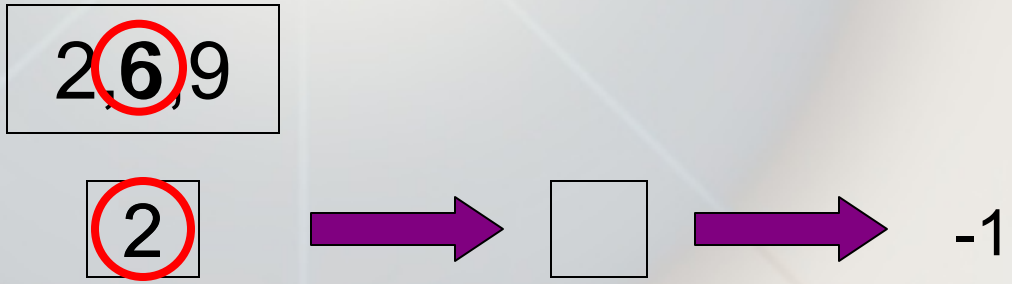
جستجوی موفق:

2,6,9,12,14,18,22,27 x=6

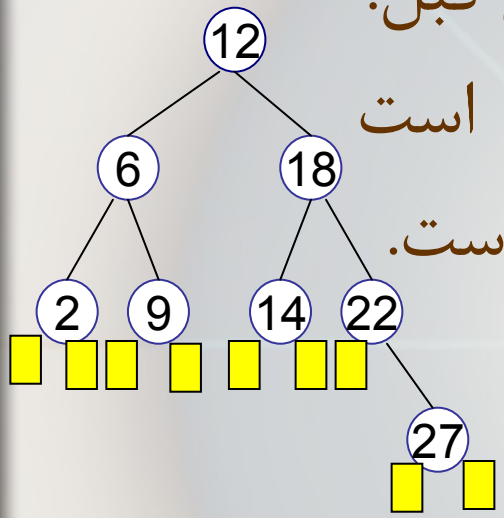


جستجوی ناموفق

2,6,9,12,14,18,22,27 X=1



تحلیل زمانی الگوریتم جستجوی دودویی



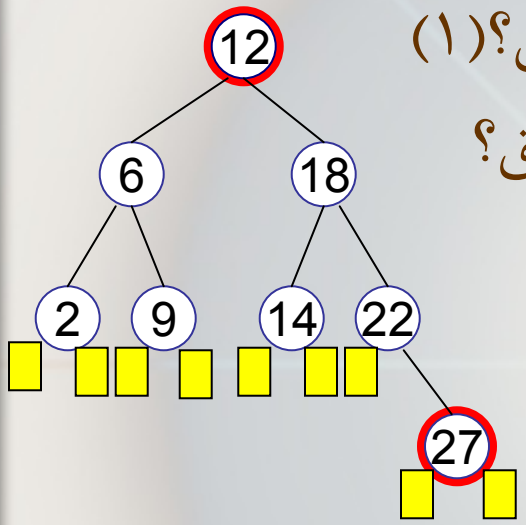
- یک درخت دودویی جستجو برای مثال قبل:
- تعداد جستجوها برای یافتن ۱۲ برابر ۱ است
- تعداد جستجوها برای یافتن ۶ برابر ۲ است.
- میانگین جستجوی موفق:

$$(1+2+2+3+3+3+3+4)/8=2.7$$

- میانگین جستجوی ناموفق:

$$(3+3+3+3+3+3+3+4+4)/9=3.2$$

تحلیل زمانی الگوریتم جستجوی دودویی



- حداقل تعداد مقایسه برای جستجوی موفق؟(۱)
- حداکثر تعداد مقایسه برای جستجوی موفق؟

$$2^{k-1} \le n < 2^k$$

(برای درختی به ارتفاع k، k می باشد.)

- بهترین زمان اجرا در جستجوی موفق: $\Omega(1)$
- بدترین زمان اجرا در جستجوی موفق: $O(\log n)$
- در جستجوی ناموفق همیشه باید تا پایین ترین سطح پیش رفت
- زمان اجرای الگوریتم در جستجوی ناموفق: $\theta(\log n)$

محاسبه میانگین زمان اجرای الگوریتم جستجوی دودویی

- گره داخلی در یک درخت دودویی: گرهی که برگ نباشد
- گره خارجی در درخت دودویی: هر گرهی که برگ باشد
- I: طول مسیرهای داخلی (مجموع فاصله هر گره داخلی تا ریشه) $I = \sum d(x)$
- E: طول مسیرهای خارجی (مجموع فاصله هر گره خارجی تا ریشه) $E = \sum d(x)$

محاسبه میانگین زمان اجرای(ادامه)

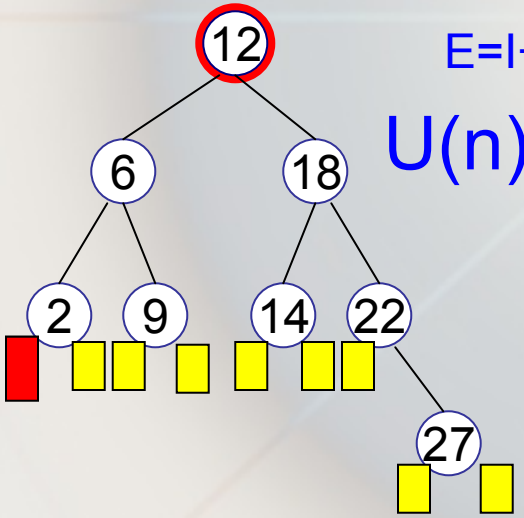
- رابطه مقابل در یک درخت جستجوی دودویی با n گره برقرار است: $E=I+2n$
- فرض کنید S(n) میانگین تعداد مقایسه برای جستجوی موفق باشد
- فرض کنید U(n) میانگین تعداد مقایسه برای جستجوی ناموفق باشد

$$S(n)=\frac{\sum[d(x)+1]}{n} \qquad U(n)=\frac{\sum d(x)}{n+1}$$

محاسبه میانگین زمان اجرای(ادامه)

$$S(n)=\frac{\sum[d(x)+1]}{n} \qquad U(n)=\frac{\sum d(x)}{n+1}$$

$$S(n)=(I+n)/n \qquad U(n)=E/(n+1)$$



$E=I+2n \Rightarrow S(n)=(1+1/n)U(n)-1$

$U(n)=\theta(\log n) \Rightarrow S(n)=O(\log n)$

تحلیل زمانی جستجوی دودویی

مرتبه زمانی / حالتها	بهترین حالت	حالت میانگین	بدترین حالت
جستجوی موفق	$\Omega(1)$	$\Theta(\log n)$	$O(\log n)$
جستجوی ناموفق	$\Omega(\log n)$	$\Theta(\log n)$	$O(\log n)$

یافتن کوچکترین و بزرگترین عنصر یک آرایه

- یک الگوریتم ساده:

```
Void  simple_max_min(int A[],int n,int &max,int &min)
{ max=min=A[0];
  for(int i=1;i<n;i++)
    If (A[i]>max) max=A[i]
     else if(A[i]<min) min=A[i]
}
```

حداقل مقایسه ها: n-1

حداکثر مقایسه ها: 2(n-1)

تعداد مقایسه ها: 2(n-1)

میانگین مقایسه ها: 3n/2-1

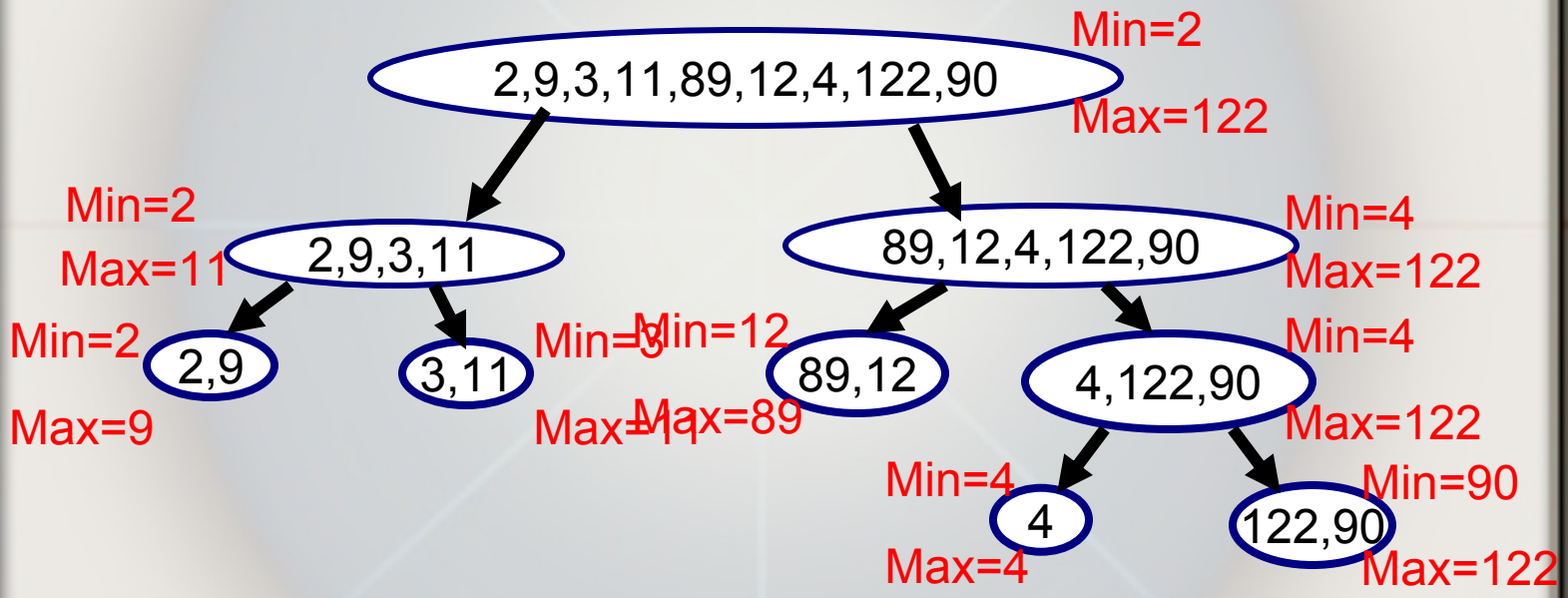
$\theta(n)$

- حداقل، حداکثر و میانگین مقایسه ها چه موقع اتفاق می افتد؟

یافتن کوچکترین و بزرگترین عنصر با روش تقسیم و غلبه

- ایده:
  - آرایه را به دو نیم تقسیم کن
  - بزرگترین و کوچکترین عنصر نیمه سمت چپ را محاسبه کن (با روش تقسیم و غلبه) و آنها را به ترتیب max1 و min1 بنام
  - بزرگترین و کوچکترین عنصر نیمه سمت راست را محاسبه کن (با روش تقسیم و غلبه) و آنها را به ترتیب max2 و min2 بنام
  - بزرگترین عدد برابر است با MAX(max1,max2) و کوچکترین عدد برابر است با MIN(min1, min2)

مثال


2,9,3,11,89,12,4,122,90
Min=2
Max=122
2,9,3,11
Min=2
Max=11
89,12,4,122,90
Min=4
Max=122
2,9
Min=2
Max=9
3,11
Min=3
Max=11
89,12
Min=12
Max=89
4,122,90
Min=4
Max=122
4
Min=4
Max=4
122,90
Min=90
Max=122

الگوریتم تقسیم و غلبه برای این مسئله

```
Void Find_min_max(int low,int high, int min, int max)
{ int mid,min1,min2,max1,max2;
  If(high-low)<2) if(A[low]>A[high]){max=A[low]; min=A[high];}
  else if(high=low+1){
  If(A[low]>A[high]){max=A[low]; min=A[high];}
  else{max=A[high]; min=A[low];}}
  else
  mid=(low+high)/2;
  Find_min_max(low,mid,min1,max1);
  Find_min_max(mid+1,high,min2,max2);
```

تحلیل زمانی الگوریتم تقسیم و غلبه

- فرض کنید T(n) زمان اجرای Find_min_max به ازای low=0 و high=n-1 باشد (آرایه ای به طول n)

```
{  int mid,min1,min2,max1,max2;

   mid=(low+high)/2;
   Find_min_max(low,mid,min1,max1);
   Find_min_max(mid+1,high,min2,max2);
   if(max1>max2) max=max1;
   else max=max2;
   if(min1<min2) min=min1;
   else min=min2;
   if(A[low]<A[high]) { max=A[high]; min=A[low];}
   }
   else  max=A[low]; min=A[high];

}

else
If (low=high) min=max=A[low];
else if (high==low+1){
   T(1)=T(2)=1
```

$T(n)=2T(n/2)$

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

T(n/2)

O(1)

T(n/2)

T(n)=θ(n)

T(1)=T(2)=1

محاسبه تعداد مقایسه ها در الگوریتم Find_min_max

- فرض کنید T(n) تعداد مقایسه های انجام شده در الگوریتم Find_min_max به ازای low=0 و high=n-1 باشد (آرایه ای به طول n)

$$\begin{cases} T(n)=2T(n/2)+2 \\ T(1)=0 \\ T(2)=1 \end{cases} \Rightarrow T(n)=3n/2-2$$