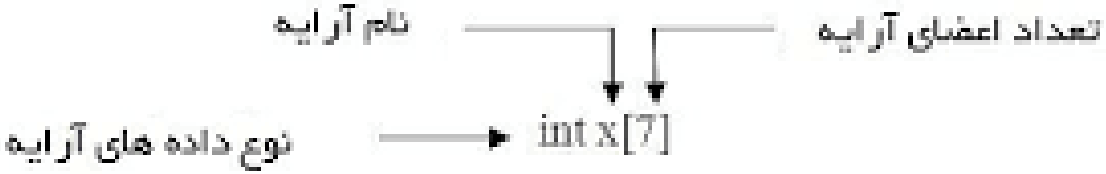
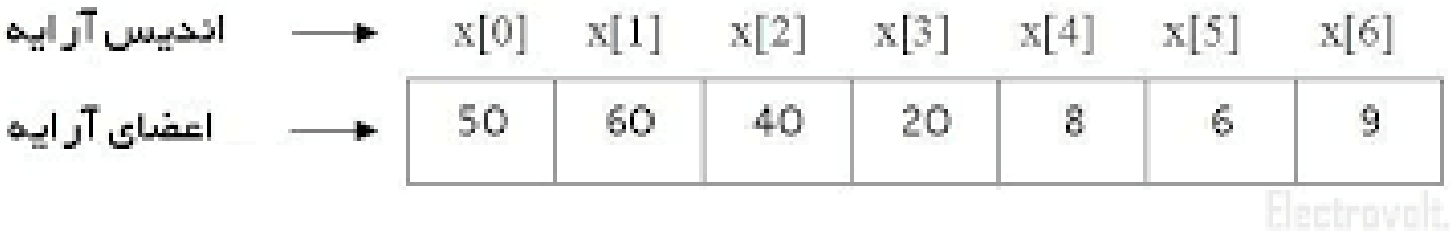
# ساختمان‌های داده

# Data Structures

# آرایه ها

# Arrays



# آرایه یک بعدی

- یک آرایه یک شامل تعدادی عنصر از یک نوع دادهای (data type) می باشد.
- هر آرایه دارای نام، نوع داده ای و اندازه می باشد.
- تعریف در سی شارپ:

type[]  name = new type[size]

int[] A = new int[10]  تعریف یک آرایه ۱۰ عنصری از اعداد صحیح

- خانه های آرایه بصورت پشت سرهم در حافظه قرار می گیرند.

# آرایه یک بعدی

- تعریف در پاسکال:

A : Array[$L_1$..$U_1$] type;

A: Array[2..11] of integer;

- آرایه ای با ۱۰ عنصر (11-2+1=10) از نوع عدد صحیح
- هر نوع داده ای به اندازه $n$ بایت فضا اشغال می کند.
- موقعیت شروع این آرایه در حافظه : $\alpha$
- خانه های آرایه بصورت پشت سرهم در حافظه قرار می گیرند.

# آرایه یک بعدی

- آدرس خانه iام و تعداد عناصر آرایه از فرمول های زیر به دست می آید:

تعداد عناصر آرایه A = $(U_1-L_1) + 1$

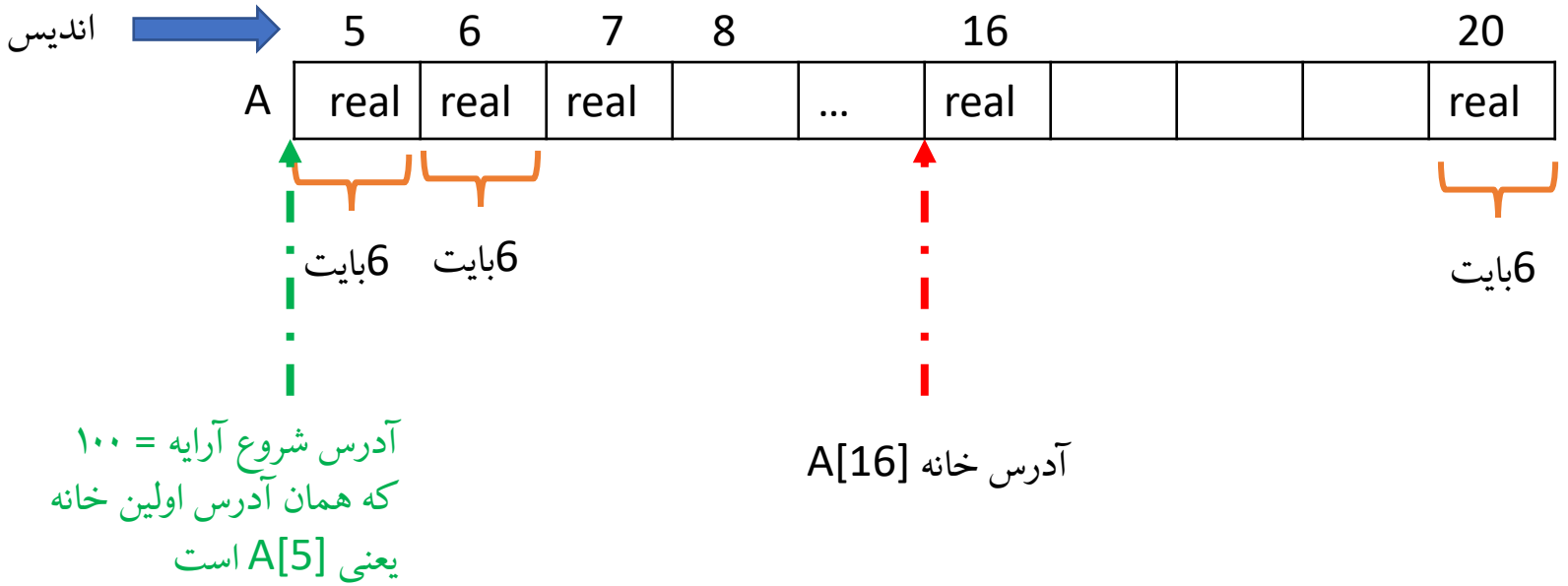
آدرس خانه A[i] = $(i-L_1) \times n + \alpha$

# آرایه یک بعدی

❖مثال : آرایه زیر تعریف شده است. اگر این آرایه از آدرس ۱۰۰ حافظه به بعد قرار گرفته باشد، آدرس خانه A[16] کدام است؟ تعداد عناصر آرایه را نیز به دست آورید.

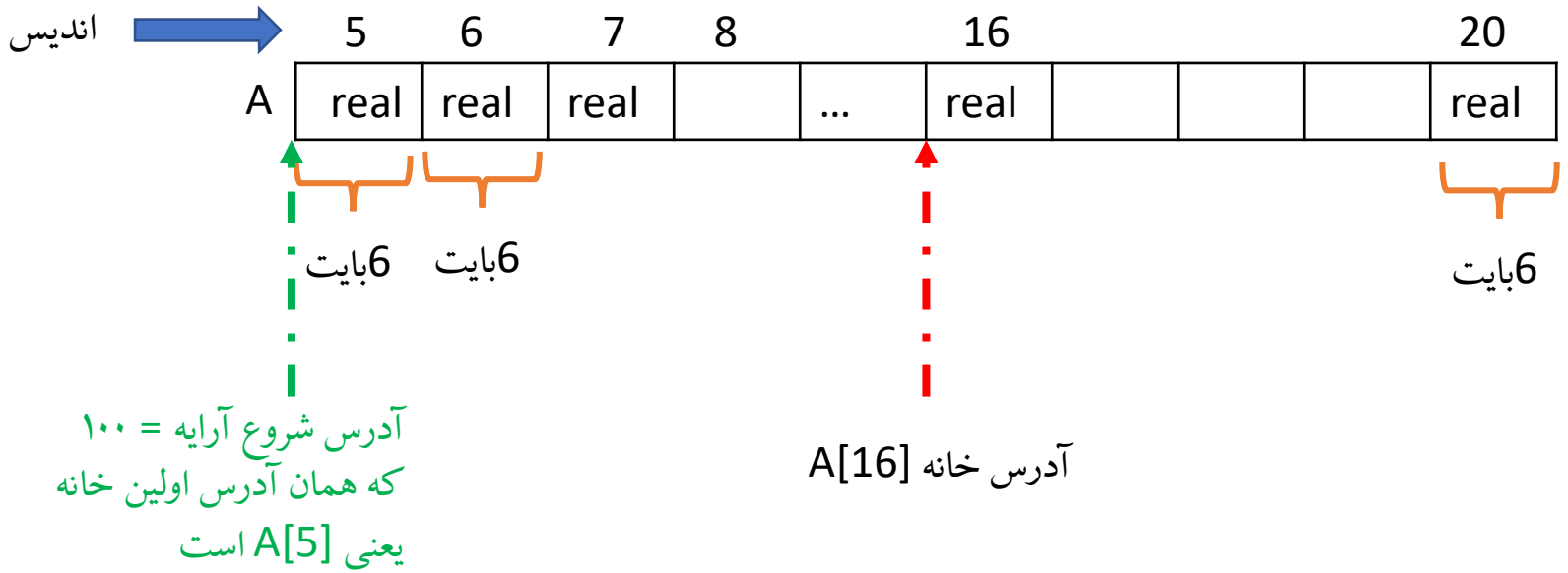
- فرض کنید که نوع داده ای real به مقدار ۶ بایت حافظه اشغال نماید.

A: Array[5..20] of real;



# آرایه یک بعدی

A: Array[5..20] of real;

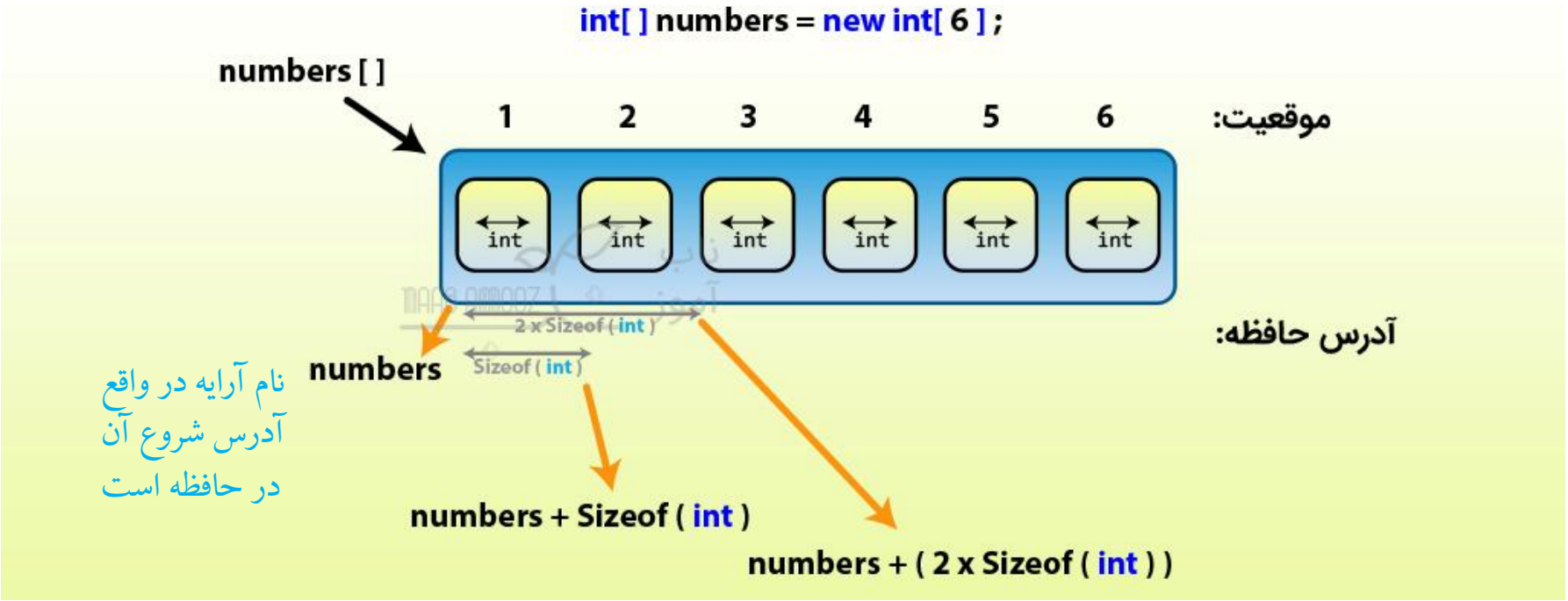



**آدرس خانه A[16]=(16-5)*6+100 = 66 + 100 = 166**

**تعداد عناصر آرایه A=(20-5+1) = 16**

**فضایی که کلا آرایه اشغال کرده است=(20-5+1)*6 = 96 Bytes**

# آرایه یک بعدی




- تابع Sizeof : تابعی است که فضایی اشغال شده توسط یک نوع داده (data type) را بر می گرداند
- مثلا برای int در سی شارپ ۴ بایت است.

# روش‌های جستجو در آرایه یک بعدی

- در آرایه های نا مرتب، جستجو به صورت خطی (ترتیبی) انجام می شود.
  - در این نوع جستجو عنصر مورد با هر یک از عناصر آرایه مقایسه می شود.
- در آرایه های مرتب، جستجو به صورت دودویی (Binary) انجام می شود.

# جستجوی خطی

- تابع زیر مشخص می کند که آیا عددی مانند m در کدام خانه آرایه وجود دارد. اگر وجود نداشته باشد عدد 1- را بر می گرداند. طول آرایه برابر n است.

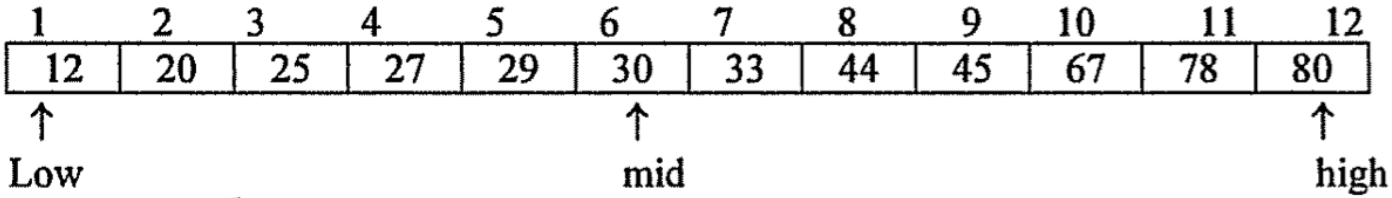
```
int  Lsearch(int x[], int n, int m)
{
      int i;
      for( i=0; i<n ; i++ )
          if( m == x[i] )
                return i;
      return -1;
}
```

# جستجوی دودویی

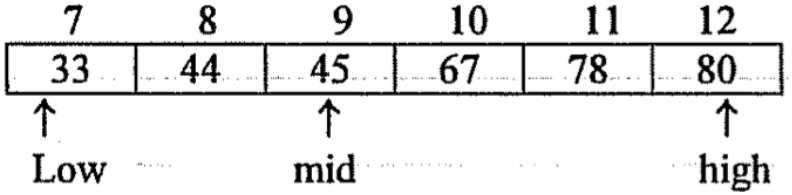
- جستجوی دودویی فقط در آرایه های مرتب استفاده می شود
- در این روش عنصر مورد نظر با خانه وسط آرایه مقایسه می شود:
  - اگر با این خانه برابر بود جستجو تمام می شود
  - اگر عنصر مورد جستجو از خانه وسط بزرگتر بود جستجو در بخش بالایی آرایه
  - و در غیر اینصورت جستجو در بخش پائینی آرایه انجام می شود
- فرض کرده ایم آرایه به صورت صعودی مرتب شده است
- این رویه تا یافتن عنصر مورد نظر یا بررسی کل خانه های آرایه ادامه می یابد.

# جستجوی دودویی

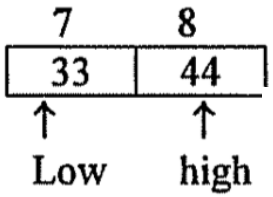
- مثال : در لیست زیر می خواهیم ببینیم عدد 44 در کدام خانه قرار دارد؟




ابتدا خانه وسطی (6) را با عدد 44 مقایسه می‌کنیم. چون 30 کمتر از 44 است پس نیمه بالایی آرایه یعنی از خانه 7 تا 12 را فقط نگاه می‌کنیم. برای این منظور Low را برابر mid + 1 قرار می‌دهیم :




حال خانه وسط یعنی خانه 9 را که حاوی عدد 45 است را با 44 مقایسه می‌کنیم. چون 44 کمتر از 45 است پس نیمه پائین این آرایه را فقط نگاه می‌کنیم. برای اینکار high را برابر mid - 1 قرار می‌دهیم.




حال خانه شماره 7 و سپس خانه شماره 8 را با کلید 44 مقایسه می‌کنیم و در می‌یابیم که عدد 44 در خانه شماره 8 قرار دارد.

# جستجوی دودویی

- تابع زیر مشخص می کند که آیا عددی مانند m در کدام خانه آرایه وجود دارد. اگر وجود نداشته باشد عدد 1- را بر می گرداند. طول آرایه برابر n است.

```
int  Bsearch(int x[], int n, int m)
{
      int 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;
}
```

# جستجوی دودویی بازگشتی

- تابع زیر مشخص می کند که آیا عددی مانند m در کدام خانه آرایه وجود دارد. اگر وجود نداشته باشد عدد 1- را بر می گرداند. طول آرایه برابر n است.

```
int  BS(int x[], int Low, int High, int m)
{
      int  mid = (low + High) /2;
      if( low<=high){
           if( m < x[mid] )
                return(BS(x, Low, mid-1, m));
           else if (m>x[mid])
                return(BS(x, mid+1, High, m)
            else
                return mid;
        }
      else
            return  -1;
}
```

# نکات جستجوی آرایه یک بعدی

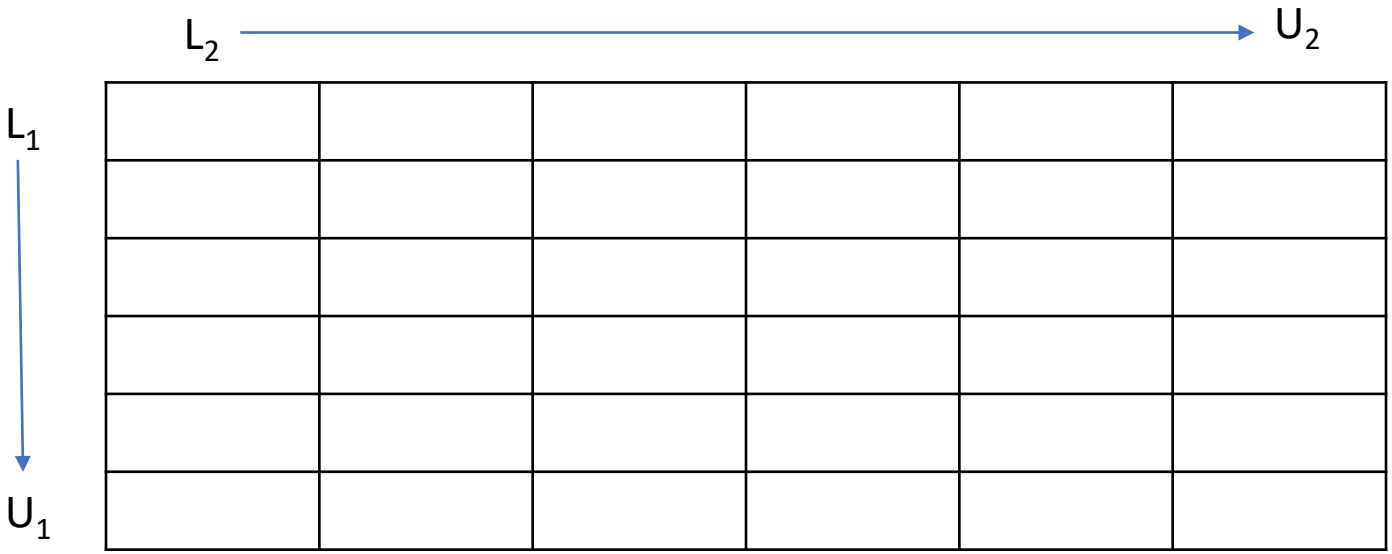
- مرتبه اجرایی جستجوی خطی : O(n) است
  - چون در بدترین حالت می بایست تمام n خانه آرایه بررسی شوند.
- مرتبه اجرایی جستجوی دودویی : O(logn) است.
  - زیرا هر بار نصف آرایه مورد بررسی قرار می گیرد
- در جستجوی دودویی، در بدترین حالت با $\lfloor \log_2 n \rfloor + 1$ عمل مقایسه می توانیم کلید مورد نظر را پیدا کنیم.

| تعداد مقایسه | بهترین حالت | حالت متوسط | بدترین حالت |
|---|---|---|---|
| در جستجوی ترتیبی | O(1) | O (n) | O (n) |
| در جستجوی باینری | O(1) | O (log n) | O (log n) |

# آرایه دو بعدی

- نحوه تعریف

A : Array[$L_1$..$U_1$, $L_2$..$U_2$] of type



# آرایه دو بعدی

- نحوه تعریف

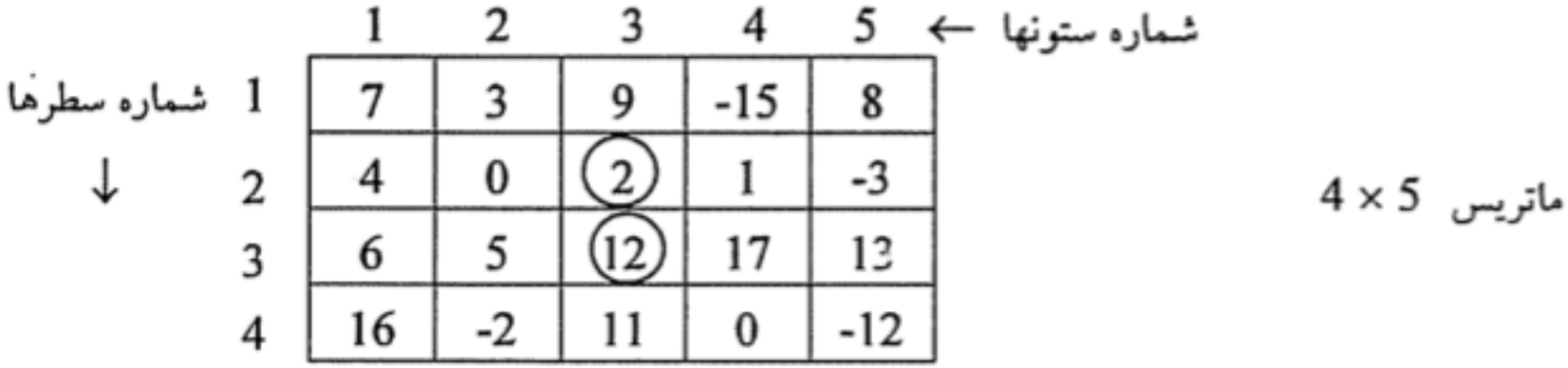
A : Array[$L_1..U_1$, $L_2..U_2$] of type

- هر عنصر type به اندازه n بایت فضا اشغال می کند.
- فرمول محاسبه تعداد عناصر آرایه دو بعدی:

تعداد عناصر آرایه = $(U_1-L_1+1)*(U_2-L_2+1)$

# آرایه دو بعدی

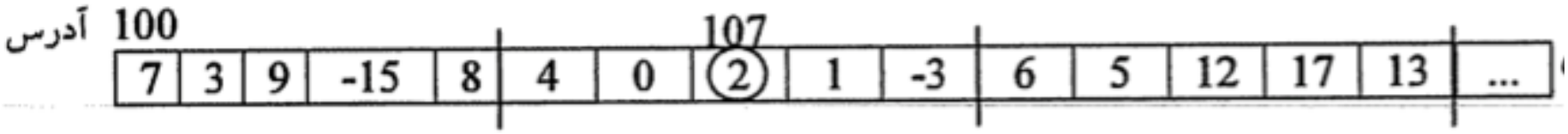
- نحوه ذخیره سازی
  - به دو صورت سطری
  - و ستونی امکان پذیر است.
- به ماتریس 4x5 زیر دقت کنید:



# آرایه دو بعدی

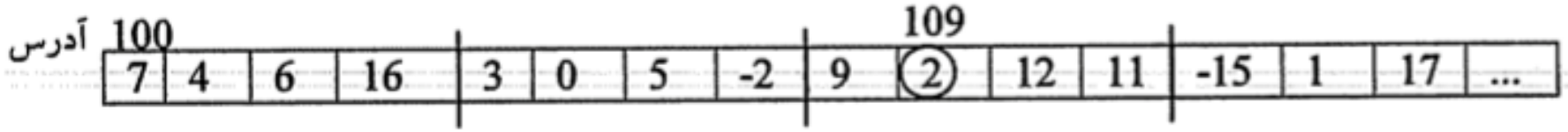
| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 7 | 3 | 9 | -15 | 8 |
| 2 | 4 | 0 | (2) | 1 | -3 |
| 3 | 6 | 5 | (12) | 17 | 13 |
| 4 | 16 | -2 | 11 | 0 | -12 |

## روش سطری




اگر آدرس شروع آرایه در حافظه عدد ۱۰۰ باشد و هر خانه یک بایت فضا بخواهد، آنگاه عنصر سطر ۲ و ستون ۳ (یعنی عدد ۲) در روش سطری در خانه ۱۰۷ حافظه خواهد بود.

## روش ستونی



# آرایه دو بعدی

- در حالت کلی در آرایه دو بعدی تعریف شده، آدرس خانه A[I, j] از فرمول های زیر به دست می آید:
  - α: آدرس شروع آرایه در حافظه است.
  - n : تعداد فضای اشغال شده توسط هر نوع داده ای (type) است.

.

$$\text{محل } A[i,j] \text{ در روش سطری} = [(i - L_1)(U_2 - L_2 + 1) + (j - L_2)] * n + \alpha$$

$$\text{محل } A[i,j] \text{ در روش ستونی} = [(j - L_2)(U_1 - L_1 + 1) + (i - L_1)] * n + \alpha$$

# آرایه دو بعدی

❑ مثال :

در آرایه M [ -2 .. 17 , 3 .. 8] آدرس خانهٔ M [7 , 5] در روش ستونی چه می‌شود؟

جواب : $\alpha$ را برابر صفر و n را برابر 1 فرض می‌کنیم :

$$A[i,j] \text{ محل} = (j - L2)(U1 - L1 + 1) + (i - L1)$$

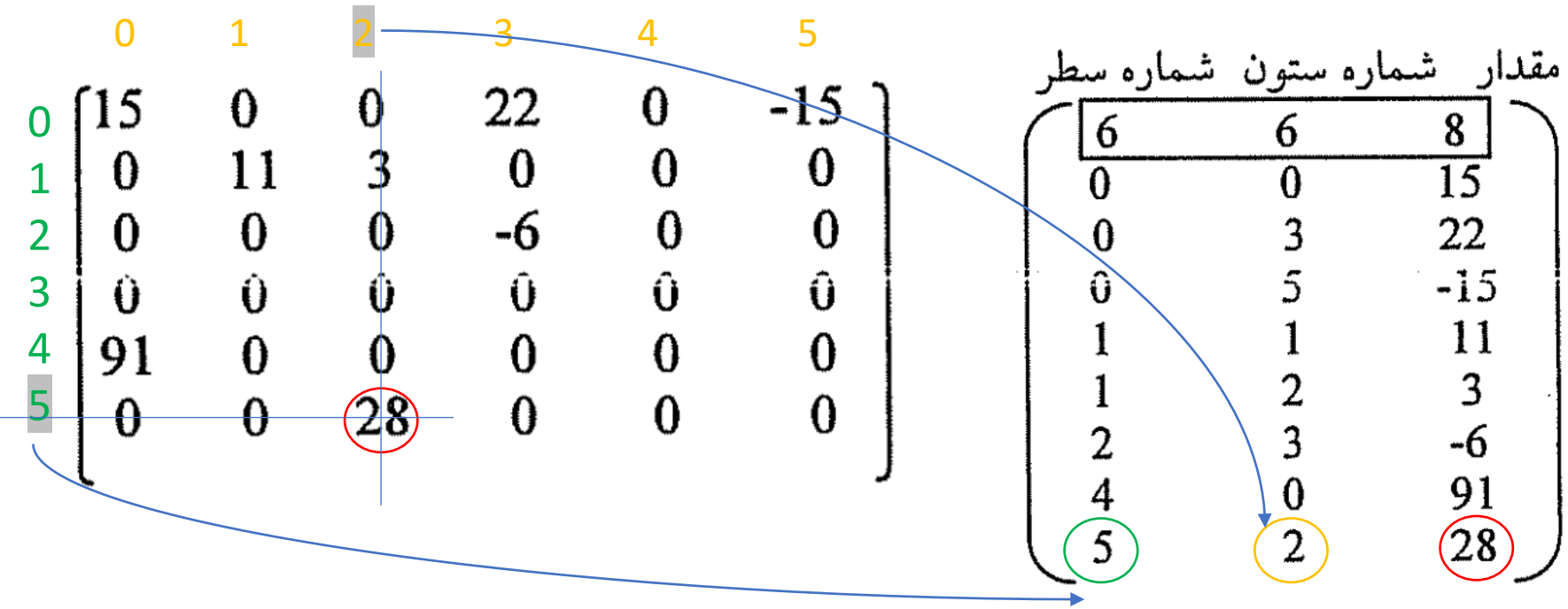
$$A[7,5] \text{ محل} = (5 - 3)(17 - (-2) + 1) + (7 - (-2)) = 2 \times 20 + 9 = 49$$

# ماتریس‌های اسپارس

- ماتریسی که عناصر صفر آن زیاد باشد، ماتریس اسپارس (پراکنده - خلوت) نامیده می شود.
- برای نمایش یک ماتریس اسپارس می توان از یک ماتریس کوچکتر ۳ ستونی استفاده کرد
  - ستون اول: شماره سطر،
  - ستون دوم: شماره ستون
  - ستون سوم: مقدار عنصر غیر صفر
- بدین ترتیب در مصرف حافظه صرفه جویی می شود.
- به عبارتی دیگر شر عنصر از ماتریس اسپارس بوسیله مکانش یعني j و i و مقدارش به صورت (i, j, value) مشخص می شود.

# ماتریس‌های اسپارس

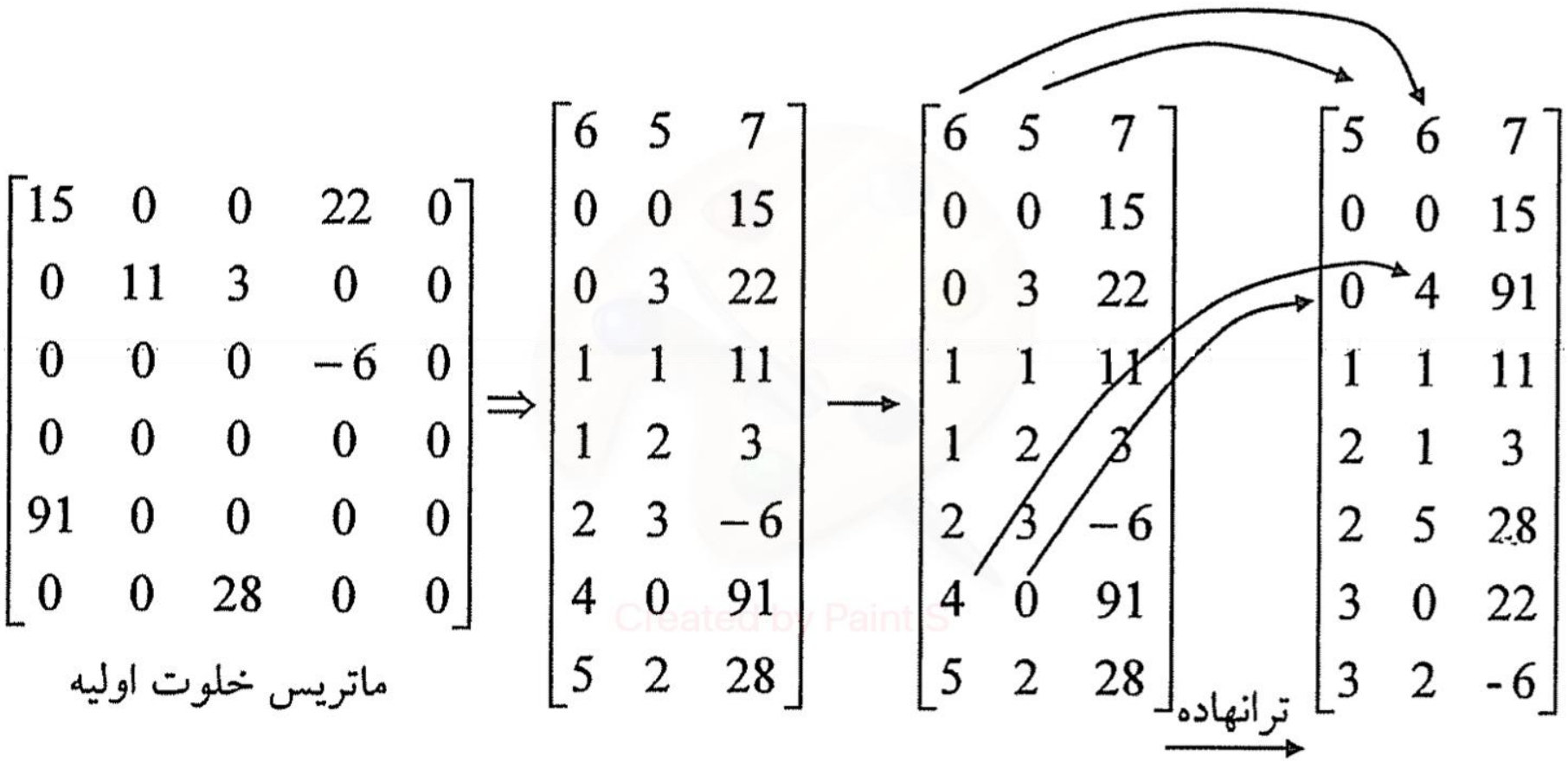
- نمایش ماتریس اسپارس زیر به چه صورت خواهد شد؟



# ترانهاده ماتریس اسپارس

- ترانهاده (Transpose) یک ماتریس $A_{mn}$ را به صورت $A^T_{nm}$ نمایش می دهیم.
- ترانهاده یک ماتریس به سادگی با جابجایی سطر اول با ستون اول، سطر دوم با ستون دوم و الی آخر به دست می آید.
- جهت ترانهاده کردن آن از روش زیر استفاده می کنیم.
  - ابتدا سطر اول (هدر) که بیانگر ابعاد ماتریس خلوت و تعداد عناصر غیر صفر آن است را با جابجا کردن تعداد سطر و ستونها در ترانهاده می نویسیم.
  - سپس در ستون وسطی به دنبال اعداد و گشته و آنها را در ستون اول ترانهاده می نویسیم، ستون اول متناظر در نمایش اولیه را در ستون وسط ترانهاده وارد می کنیم.
  - بعد از آن در ستون وسطی به دنبال اعداد ۱ می گردیم و الی آخر.
- (فرض کرده ایم شماره سطر و ستونها از صفر شروع می شوند)

# ترانهاده ماتریس اسپارس

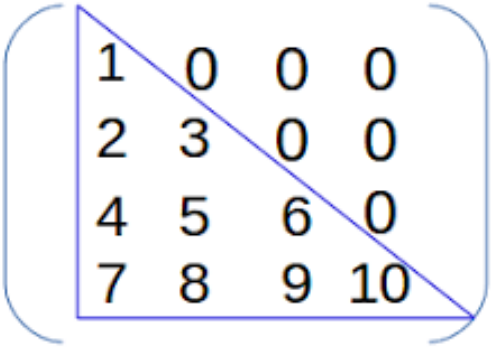


# ماتریس‌های خاص

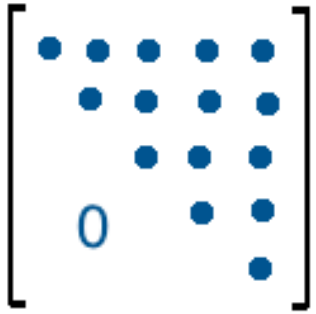
- ماتریس های بالا مثلثی
- ماتریس های پایین مثلثی
- ماتریس های قطری
- ماتریس های سه قطری

# ماتریس‌های بالا (پایین) مثلثی

- ماتریسی است که تمام اعضای پایین (بالای) قطر اصلی آن صفر است.
- مشابه ماتریس اسپارس اگر بصورت دو بعدی ذخیره شود، مقدار زیادی حافظه هدر خواهد رفت.




ماتریس پایین مثلثی




ماتریس بالا مثلثی


ماتریس پایین مثلثی

# ماتریس‌های پایین مثلثی

❑ برای صرفه جویی در حافظه (به حدود ۵۰٪) می توان ماتریس پایین مثلثی را به صورت آرایه ای یک بعدی ذخیره سازی نمود.

$$A_{4\times4} = \begin{bmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| $a_{11}$ | $a_{21}$ | $a_{22}$ | $a_{31}$ | $a_{32}$ | $a_{33}$ | $a_{41}$ | $a_{42}$ | $a_{43}$ | $a_{44}$ |

آرایه یک بعدی B

❑ حال بایستی رابطه ای بین A[i, j] و B[k] برقرار نمود.

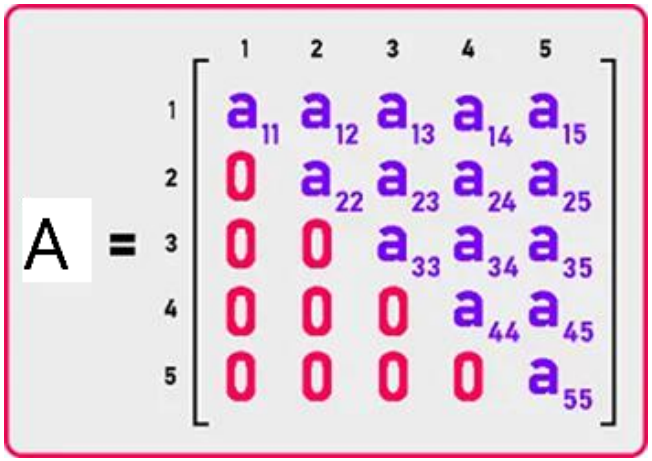
- یعنی عنصر A[i, j] در کدام خانه B قرار دارد.

عنصر $a_{ij}$ در سطر i ام قرار گرفته و قبل از آن $1+2+3+\ldots+(i-1)=\frac{i(i-1)}{2}$ عنصر در سطرهای قبلی قرار گرفته‌اند. حال این عنصر $a_{ij}$ در سطر iام به اندازه j خانه جلو آمده است لذا :

عنصر A[i,j] ماتریس پائین مثلثی معادل است با عنصر $B[\frac{i(i-1)}{2}+j]$

# ماتریس‌های بالا مثلثی

- برای صرفه جویی در حافظه (به حدود ۵۰٪) می توان ماتریس بالا مثلثی را به صورت آرایه ای یک بعدی ذخیره سازی نمود.




| a11 | a12 | a13 | a14 | a15 | a22 | a23 | . | . | a55 |
|---|---|---|---|---|---|---|---|---|---|

آرایه یک بعدی B

- حال بایستی رابطه ای بین A[i, j] و B[k] برقرار نمود.
  - یعنی عنصر A[i, j] در کدام خانه B قرار دارد.

$$k = j + (i-1)(n - \frac{i}{2})$$

# ماتریس‌های سه قطری

- مشابه ماتریس های اسپارس است که همه عناصر به جز قطر اصلی و قطر بالا و قطر پایین زیر قطر اصلی صفر است.
- مشابه ماتریس های بالا (پایین) مثلثی می توان عناصر آن را در یک آرایه یک بعدی به صورت زیر ذخیره نمود.
- سپس بایستی رابطه ای بین A[i, j] و B[k] به دست آورد.

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ 0 & 0 & 0 & a_{54} & a_{55} \end{bmatrix}$$

| k =1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $a_{11}$ | $a_{12}$ | $a_{21}$ | $a_{22}$ | $a_{23}$ | $a_{32}$ | $a_{33}$ | $a_{34}$ | $a_{43}$ | $a_{44}$ | $a_{45}$ | $a_{54}$ | $a_{55}$ |

# ماتریس‌های سه قطری

$$\begin{bmatrix} a_{11} \to a_{12} & 0 & 0 & 0 \\ a_{21} \to a_{22} \to a_{23} & 0 & 0 & \\ 0 & a_{32} \to a_{33} \to a_{34} & 0 & \\ 0 & 0 & a_{43} \to a_{44} \to a_{45} & \\ 0 & 0 & 0 & a_{54} \to a_{55} \end{bmatrix}$$

| k =1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $a_{11}$ | $a_{12}$ | $a_{21}$ | $a_{22}$ | $a_{23}$ | $a_{32}$ | $a_{33}$ | $a_{34}$ | $a_{43}$ | $a_{44}$ | $a_{45}$ | $a_{54}$ | $a_{55}$ |

سطر اول به طور ویژه فقط ۲ عنصر غیرصفر دارد. پس بدیهی است که تعداد عناصر غیرصفر موجود در سطرهای قبل از سطر iام برابر است با :2+ 3(i - 2). از طرف دیگر در سطر iام عنصر $a_{ij}$ ، عنصر 2+(j-i)ام غیرصفر آن سطر است. لذا :

$$\text{اندیس معادل} = 3(i-2) + 2 + (j-i) + 2 = 2i + j - 2 \qquad A[i,j] \text{ در آرایه } B$$

عنصر A[i,j] ماتریس سه قطری معادل است با عنصر B[2i + j - 2]

# تمرینات سری سوم

۱) آرایه A در پاسکال به صورت زیر تعریف شده است. اگر آدرس شروع آرایه در حافظه عدد ۱۰۰۰ باشد، آدرس عنصر A[14] را به دست آورید.

A= Array[3..50] of Integer;

۲) اگر آرایه ای به صورت A[-12..16] در آدرس ۵۰۰ حافظه ذخیره شده باشد، و طول عناصر ۶ بایت باشد، آدرس A[-6] را به دست آورید.

۳)آرایه دو بعدی زیر در آدرس ۵۰۰۰ به بعد حافظه ذخیره شده است. آدرس عنصر list[2, 8] به روش سطری را محاسبه کنید. (فضای هر real را ۶ بایت در نظر بگیرید)

list = Array[-3..4, 5..20] of real;

# تمرینات سری سوم

۴) نمایش ماتریس اسپارس $\begin{bmatrix} 0 & 0 & 2 \\ 0 & 2 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ در کامپیوتر بهتر است به چه صورت باشد؟

# تمرینات سری سوم

۵) اگر عناصر غیرصفر در یک ماتریس سه‌قطری A به ابعاد $n \times n$ را به صورت سطری در یک آرایه یک‌بعدی b ذخیره نماییم فرمول محل ذخیره عنصر A[i, j] در حافظه چه خواهد بود؟ (به فرض $\alpha$ آدرس شروع ماتریس A در حافظه باشد).

۶) تعداد عناصر غیرصفر در یک ماتریس پایین مثلثی (ماتریسی که در آن عناصر بالای قطر اصلی برابر صفر باشند) با ابعاد n×n چیست؟ (کارشناسی ناپیوسته ـ علمی کاربردی ـ آزاد ۸٤)