مرتب سازی

Sort

مساله مرتب سازی

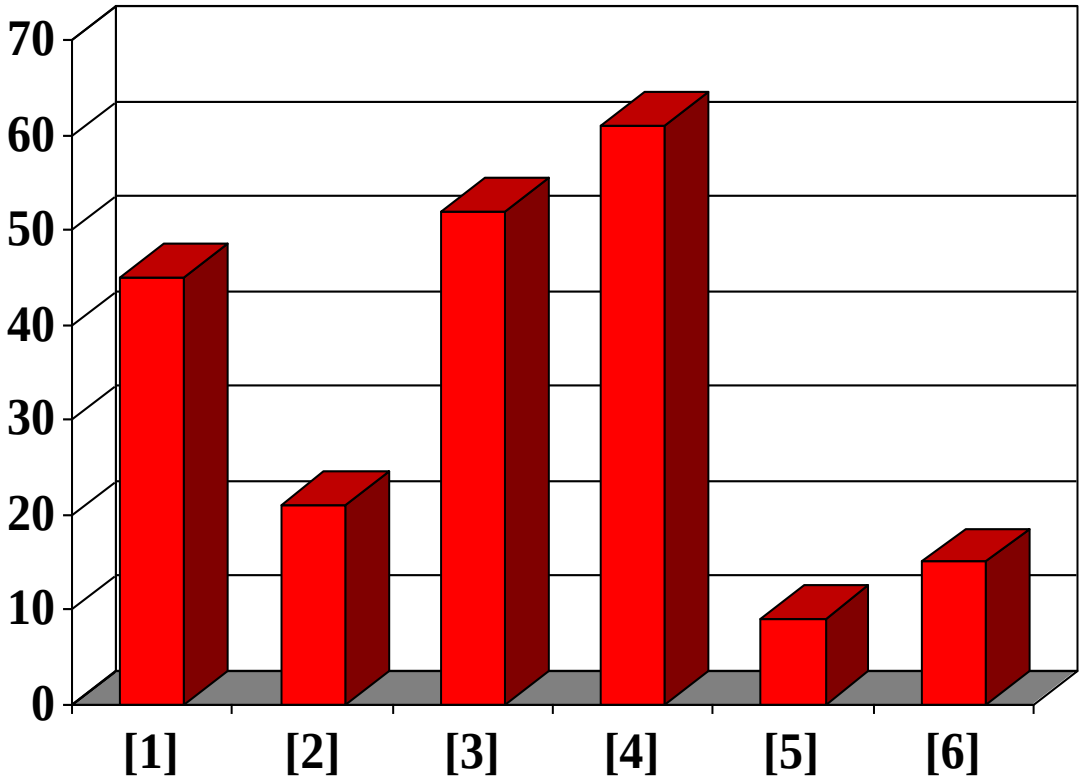
❖مشکل رایج: فهرستی از مقادیر را مرتب کنید، از کمترین به بالاترین.

- لیست نمرات امتحان
- کلمات فرهنگ لغت به ترتیب حروف الفبا
- اسامی دانش آموزان به ترتیب حروف الفبا ذکر شده است
- سوابق دانش آموز بر اساس شماره شناسه مرتب شده است
- به طور کلی، لیستی از رکوردهایی که دارای کلید هستند به ما داده می شود. این کلیدها برای تعیین ترتیب اقلام موجود در لیست استفاده می شوند.

الگوریتم های مرتب سازی مرتبه مربع

- به ما n رکورد داده می شود تا مرتب کنیم.
- تعدادی الگوریتم مرتب سازی ساده وجود دارد که بدترین و متوسط عملکرد آن‌ها $O(n^2)$ درجه دوم است:
  - مرتب سازی انتخابی Selection Sort
  - مرتب سازی درج Insertion Sort
  - مرتب سازی حبابی Bubble Sort

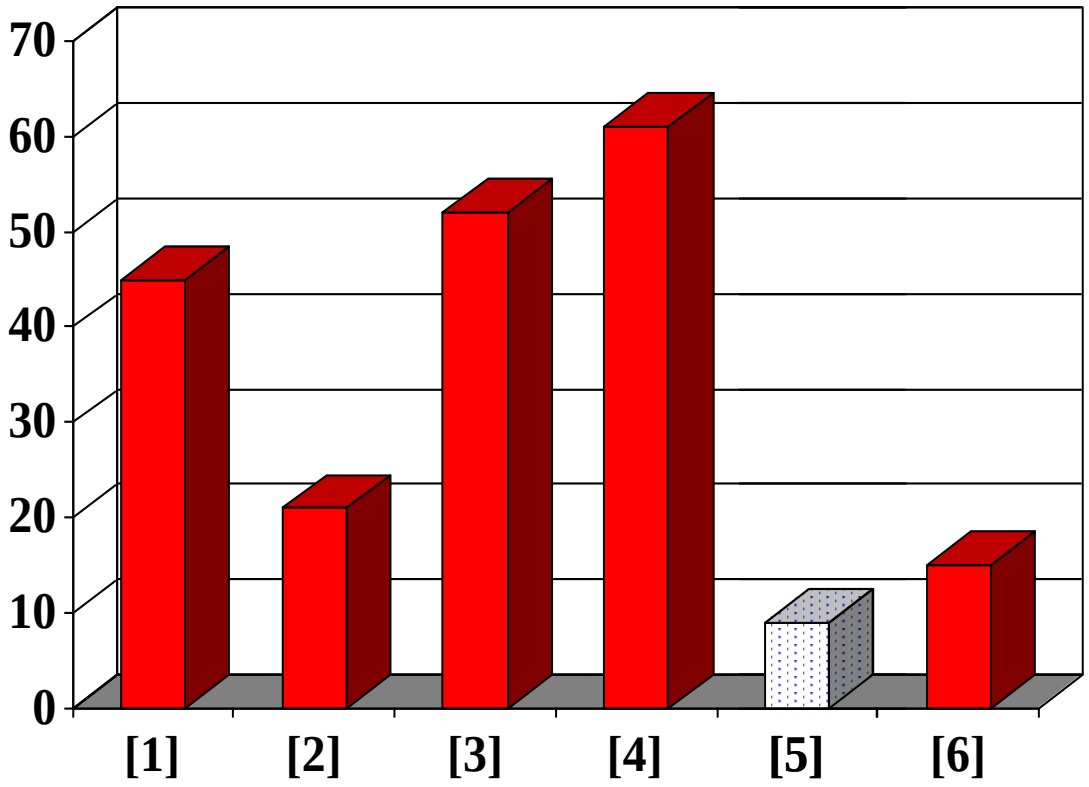
مرتب کردن یک آرایه از اعداد صحیح



- مثال: آرایه ای از شش عدد صحیح به ما داده می شود که می خواهیم آنها را از کوچکترین به بزرگ مرتب کنیم

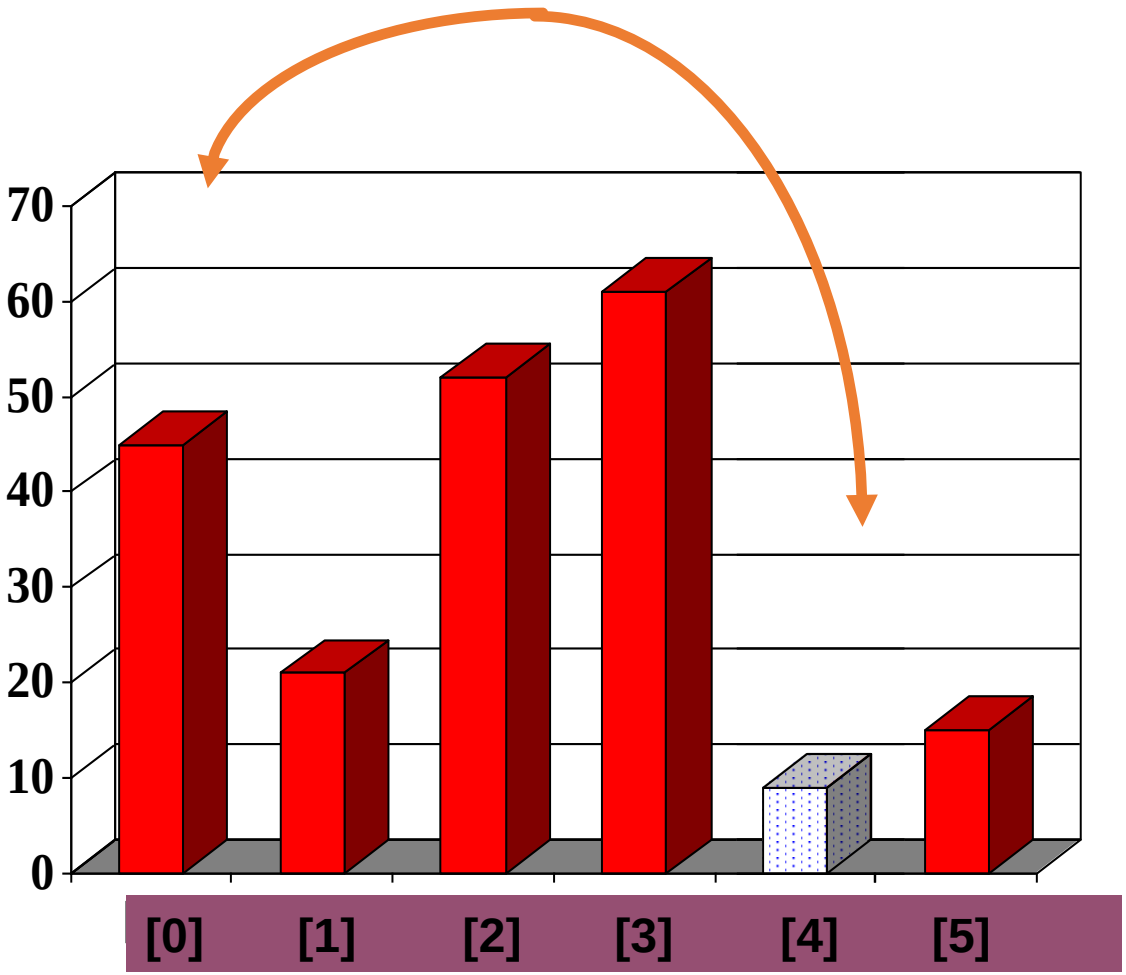
مرتب سازی انتخابی

- شروع با یافتن کوچکترین ورودی

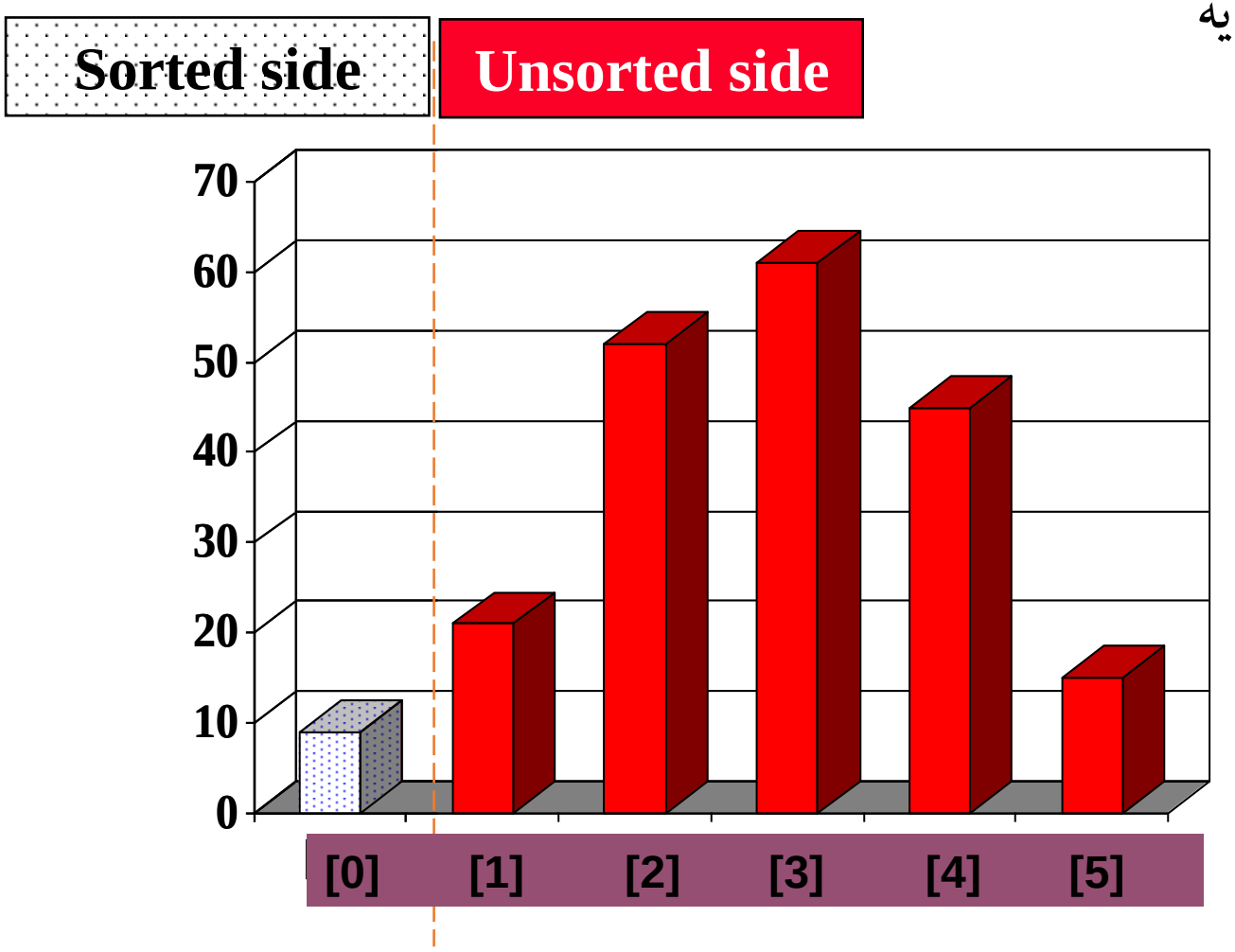


مرتب سازی انتخابی

- جابجایی آن با اولین ورودی

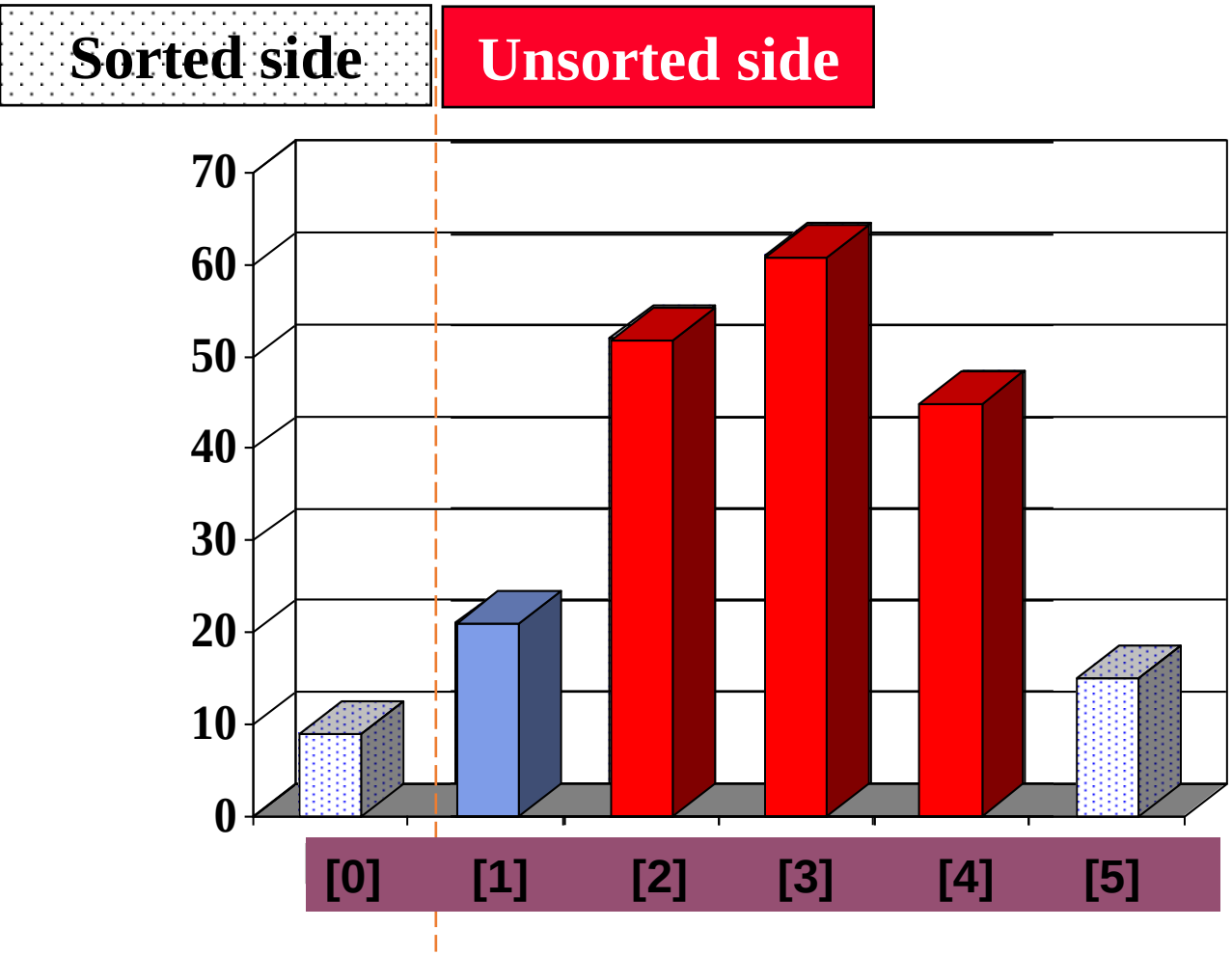


مرتب سازی انتخابی



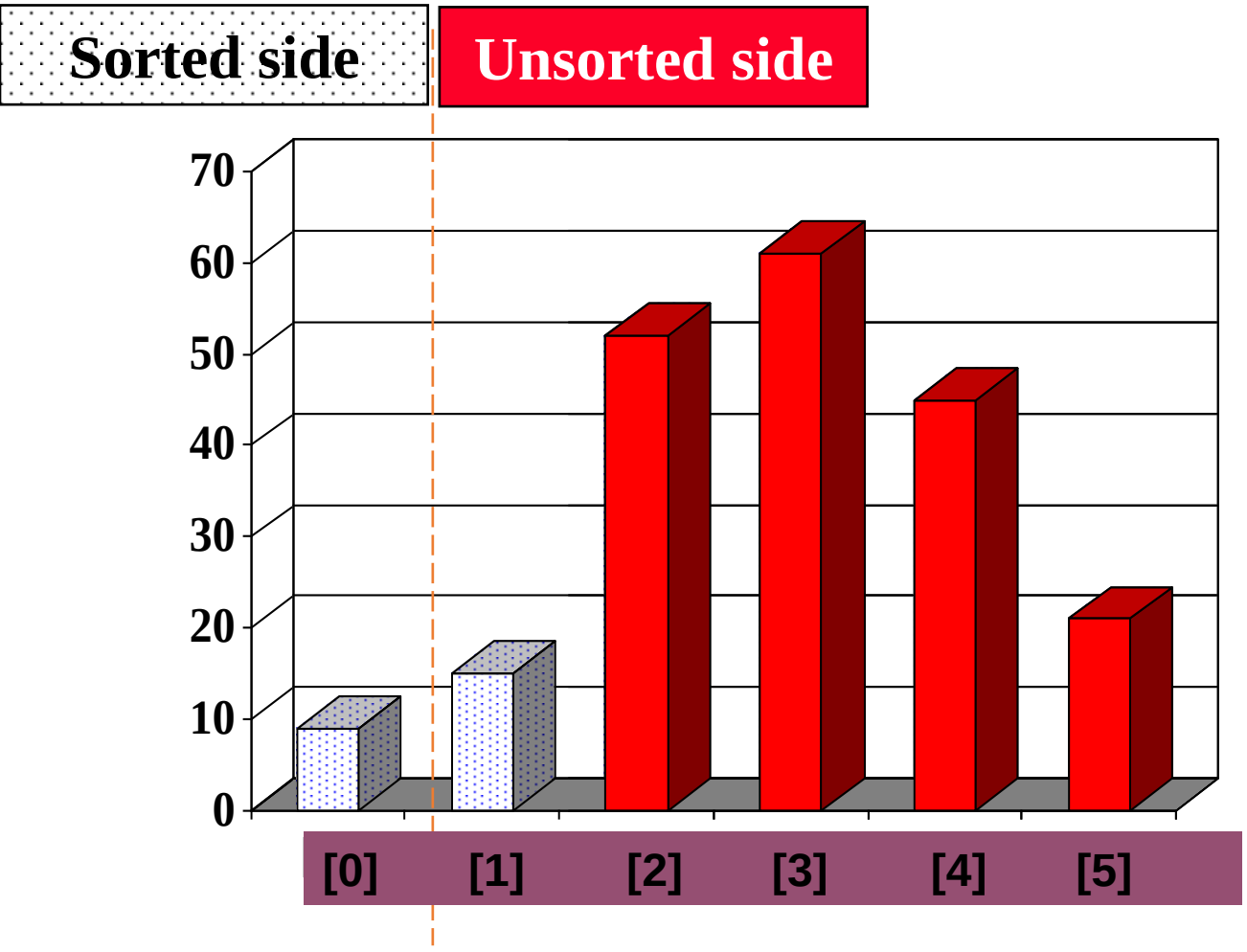
- هم اکنون بخشی از آرایه مرتب شده است

مرتب سازی انتخابی



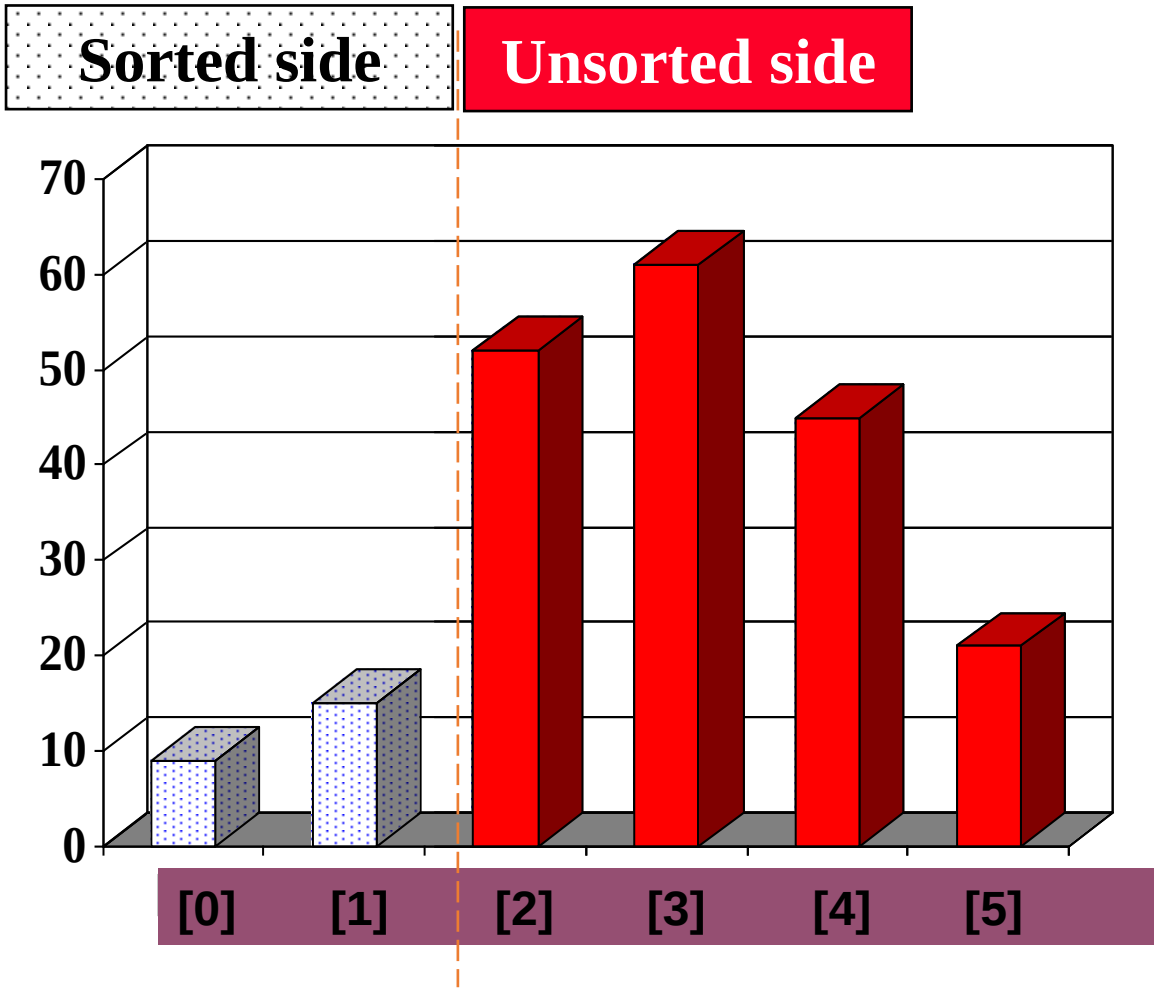
- کوچکترین عدد در سمت نامرتب را پیدا کنید

مرتب سازی انتخابی



- آن را با اولین عدد سمت نامرتب جابجا نمایید.

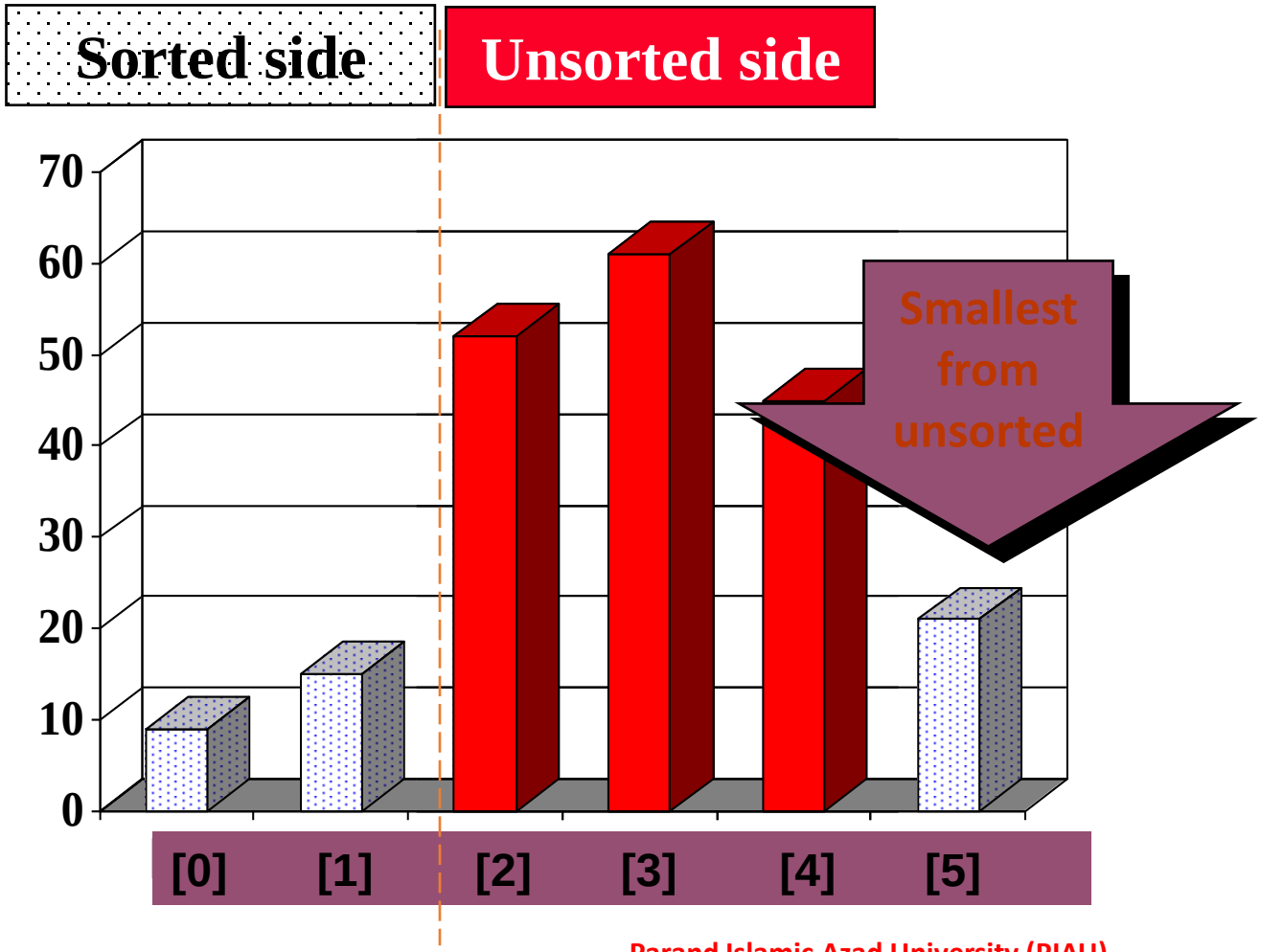
مرتب سازی انتخابی



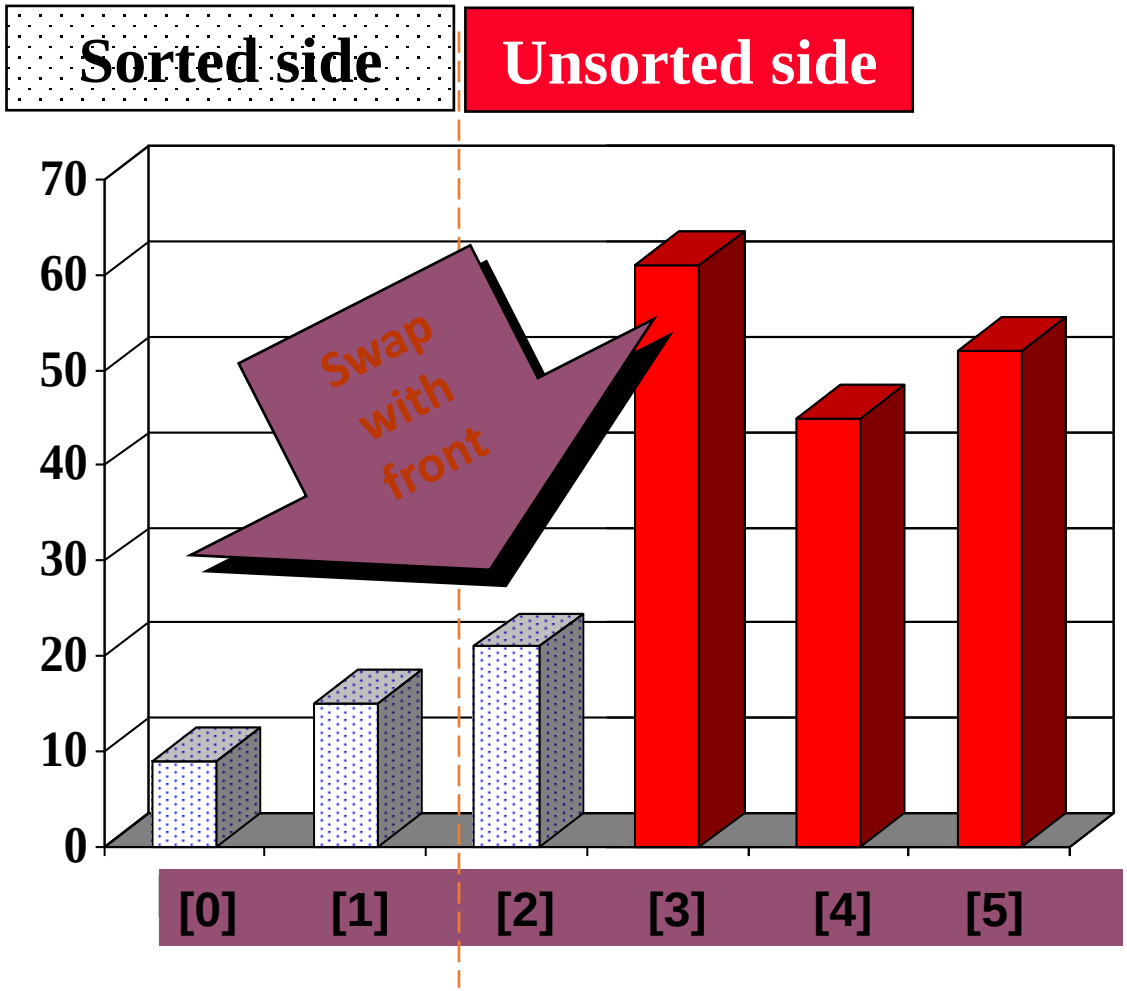
- حال بخش مرتب شده را گسترش می دهیم

مرتب سازی انتخابی

- همین فرآیند را ادامه میدهیم



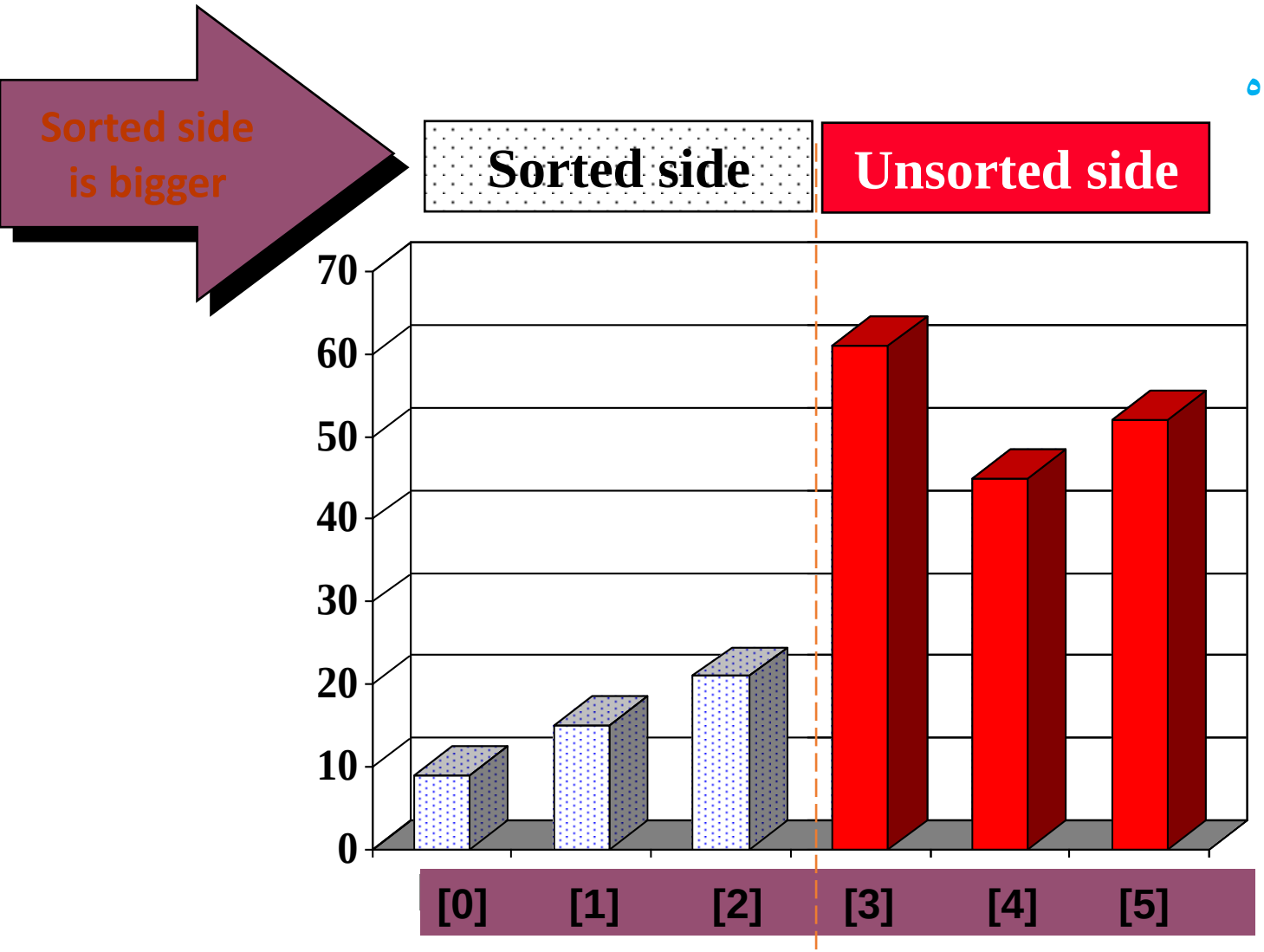
مرتب سازی انتخابی



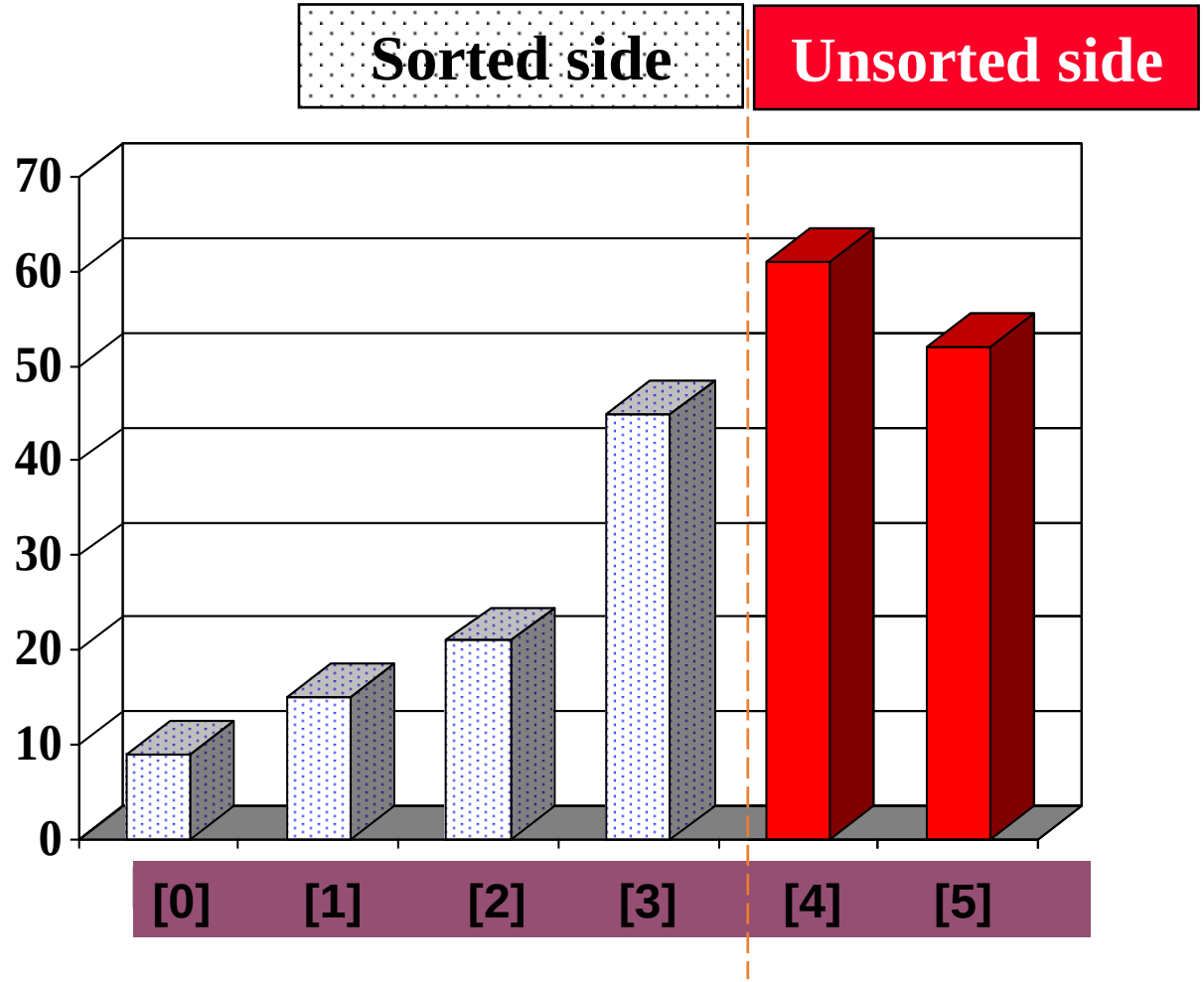
- همین فرآیند را ادامه میدهیم

مرتب سازی انتخابی

- سمت مرتب شده بزرگتر می شود.

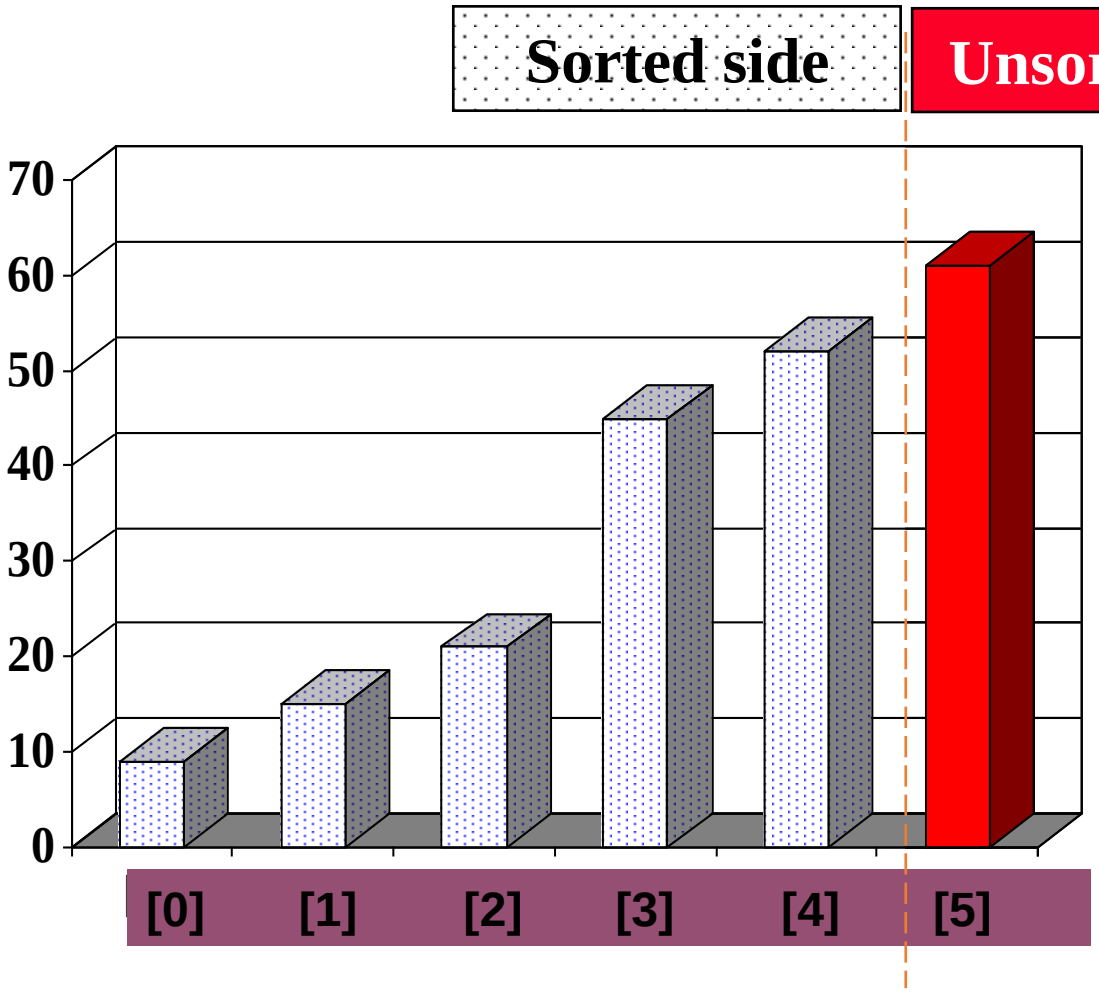


مرتب سازی انتخابی



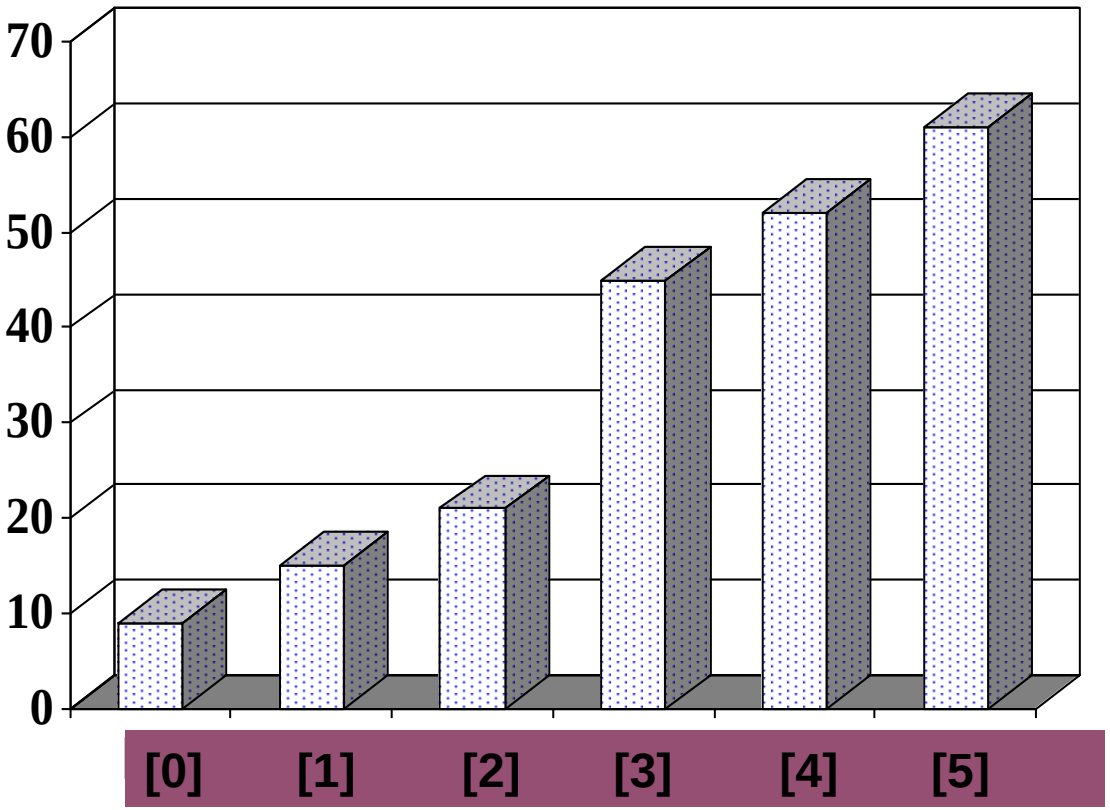
- فرآیند به اضافه کردن یک عدد بیشتر به سمت مرتب شده ادامه می دهد.

مرتب سازی انتخابی



- زمانی که سمت نامرتب فقط یک عدد دارد، می توان فرآیند را متوقف نمود.

مرتب سازی انتخابی



- آرایه هم اکنون به صورت مرتب شده می باشد.
- در واقع ما به صورت تکراری در هر مرحله کوچکترین عدد را یافته و آن را با ابتدای بخش نامرتب جابجا می نماییم.

الگوریتم مرتب سازی انتخابی

```
template <class Item>
void selection_sort(Item data[ ], size_t n)
{
    size_t i, j, smallest;
    Item temp;

    if(n < 2) return;  // nothing to sort!!

    for(i = 0; i < n-1 ; ++i)
    {
      // find smallest in unsorted part of array
      smallest = i;
      for(j = i+1; j < n; ++j)
          if(data[smallest] > data[j]) smallest = j;

      // put it at front of unsorted part of array (swap)
      temp = data[i];
      data[i] = data[smallest];
      data[smallest] = temp;
    }
}
```

تحلیل زمانی الگوریتم مرتب سازی انتخابی

```
template <class Item>
void selection_sort(Item data[ ], size_t n)
{
    size_t i, j, smallest;
    Item temp;

    if(n < 2) return;  // nothing to sort!!

    for(i = 0; i < n-1 ; ++i)
    {
      // find smallest in unsorted part of array
     smallest = i;
     for(j = i+1; j < n; ++j
        if(data[smallest] > data[j]) smallest = j;

     // put it at front of unsorted part of array (swap)
     temp = data[i];
     data[i] = data[smallest];
     data[smallest] = temp;
    }
}
```

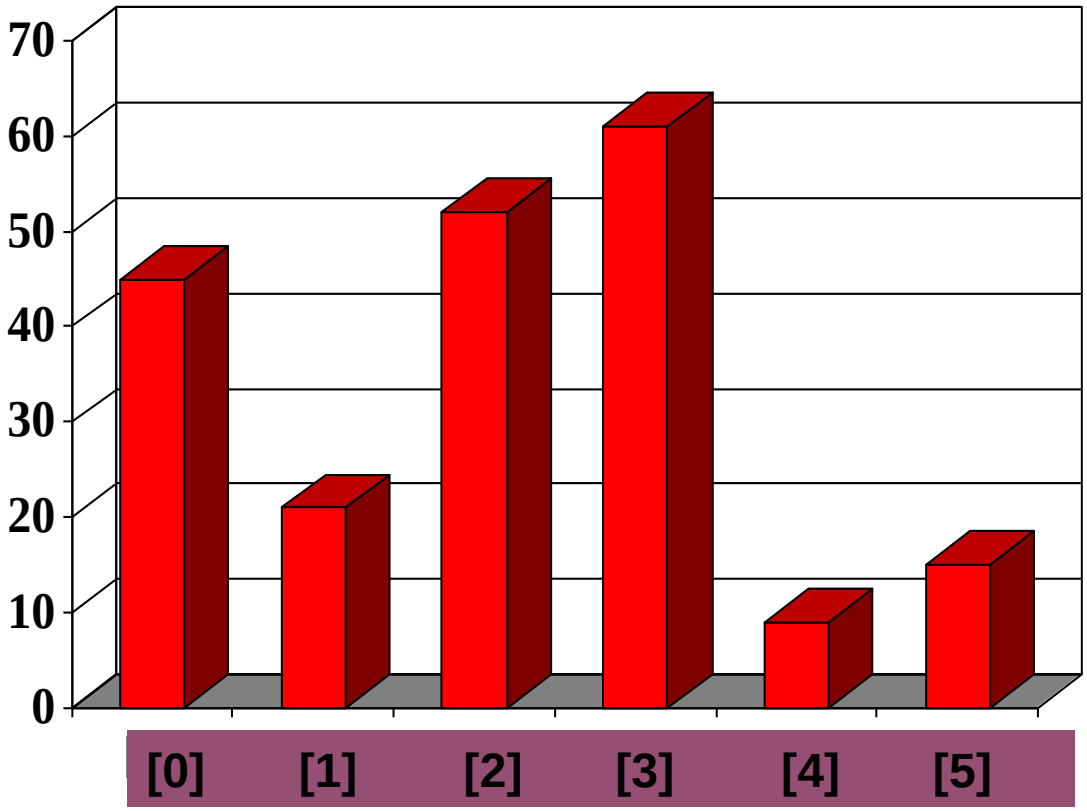
Outer loop: O(n)

Inner loop: O(n)

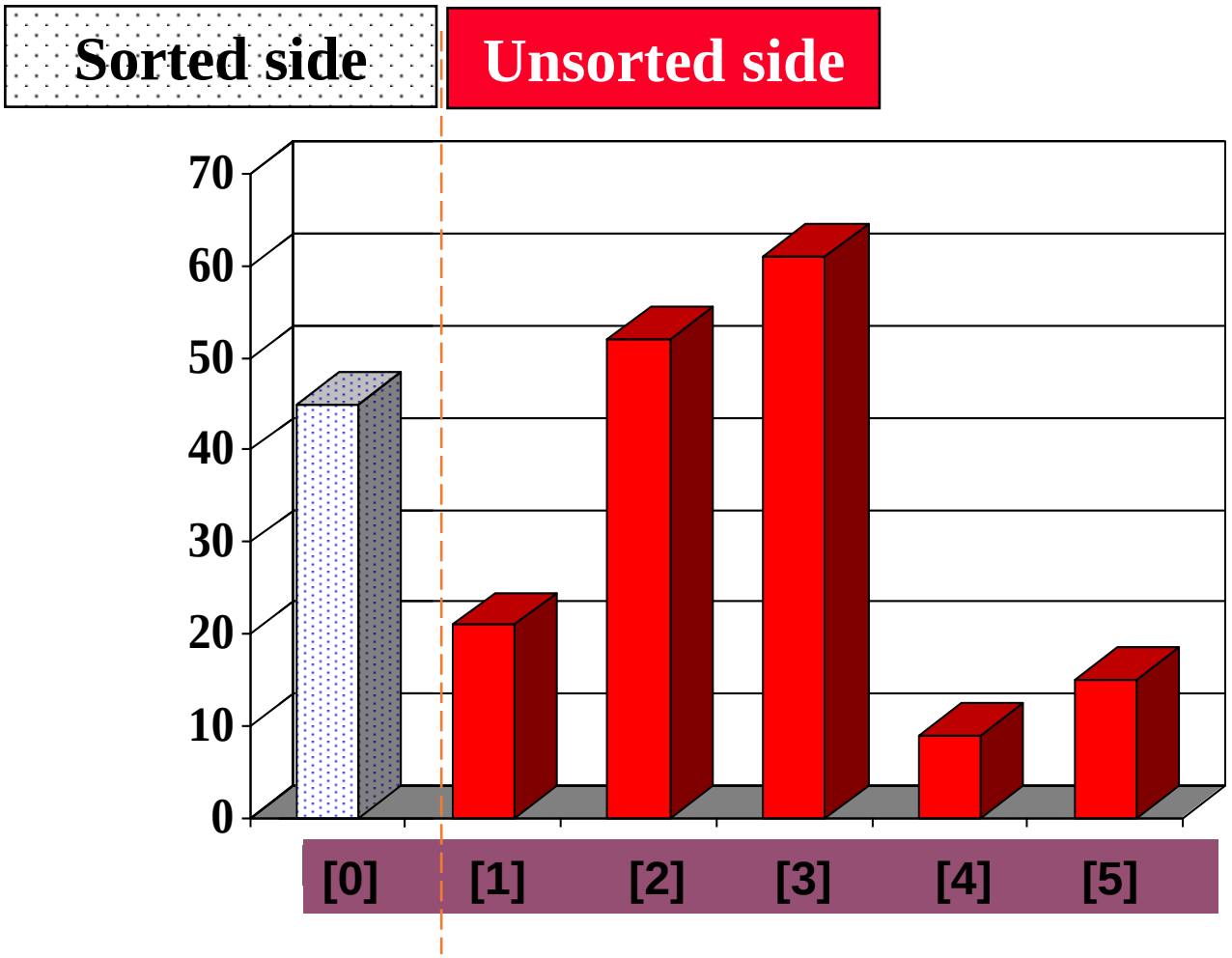
Overall : $O(n^2)$

مرتب سازی درجی

- الگوریتم مرتب سازی درجی، آرایه را به صورت سمت مرتب شده و مرتب نشده می بیند.

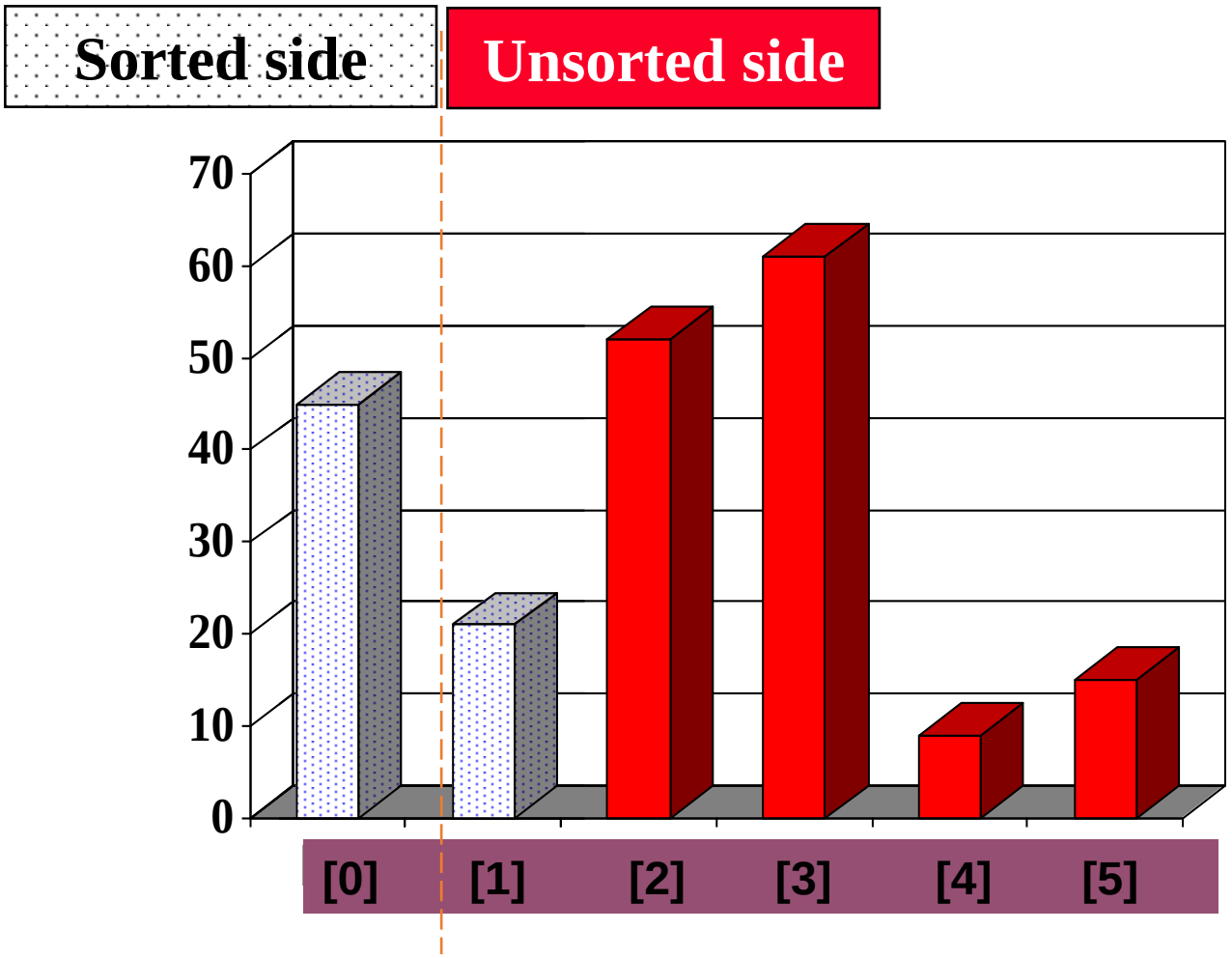


مرتب سازی درجی



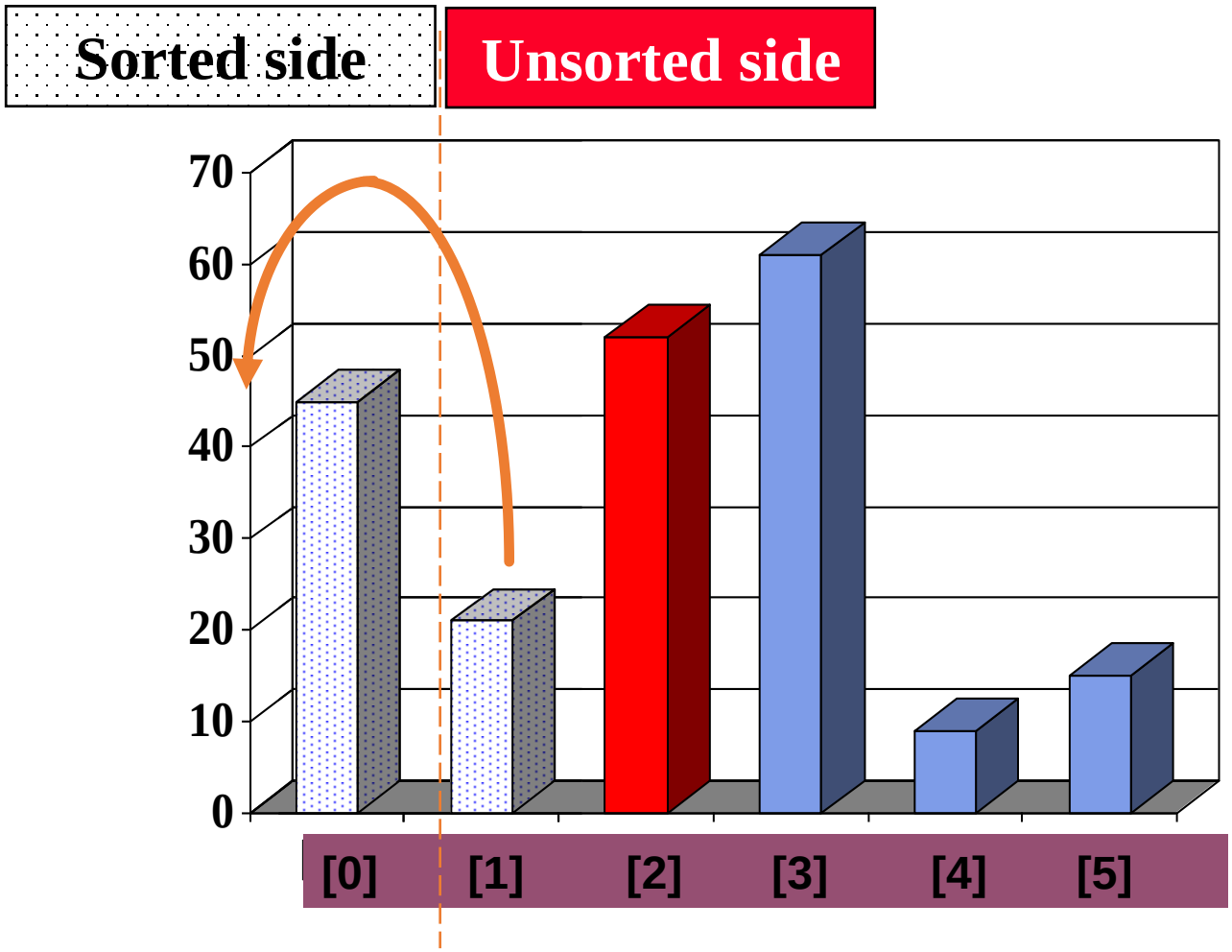
- بخش مرتب شده تنها با یک عنصر (عدد) شروع می شود که الزاما هم کوچکترین عدد نیست.

مرتب سازی درجی



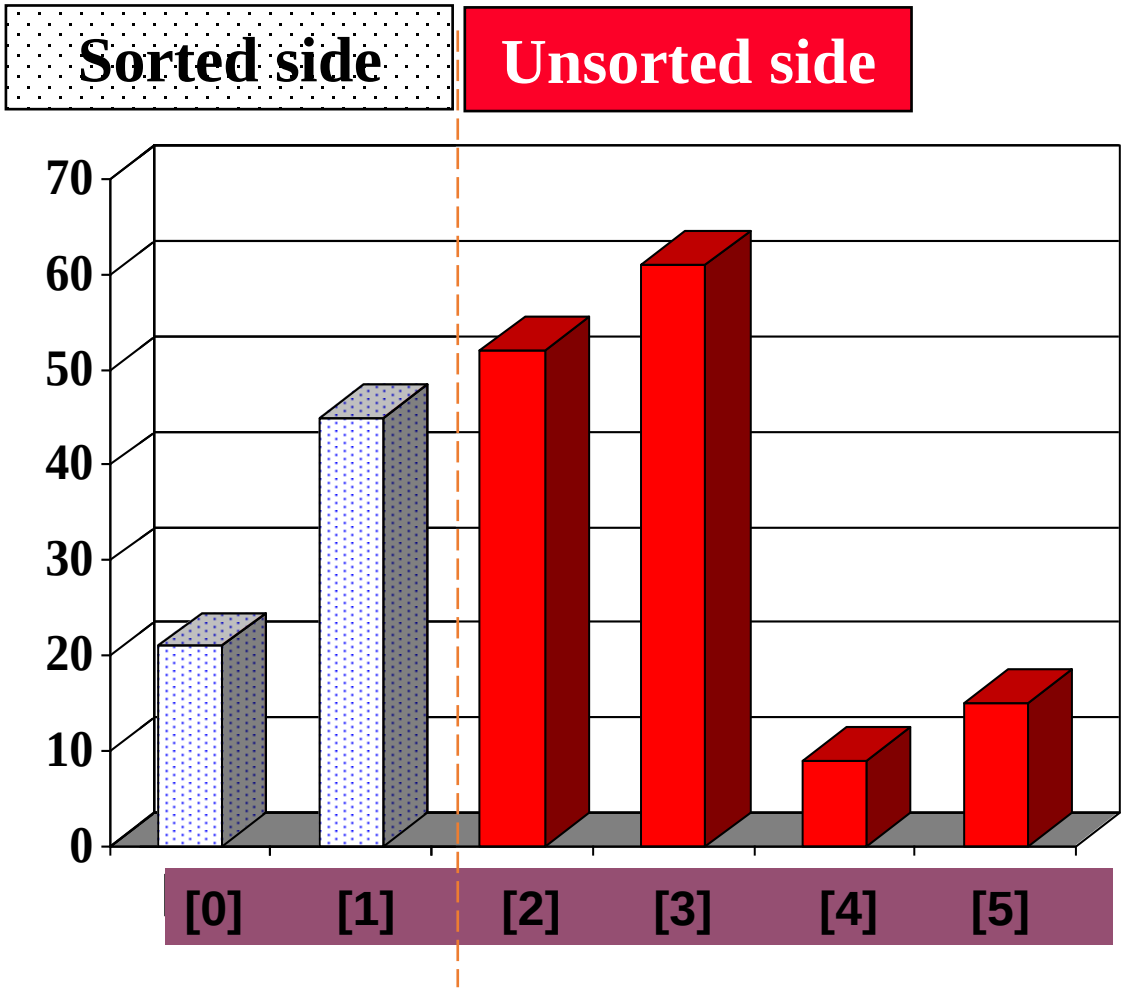
- بخش مرتب شده با دریافت اولین عدد از بخش مرتب نشده شروع به رشد می نماید.

مرتب سازی درجی



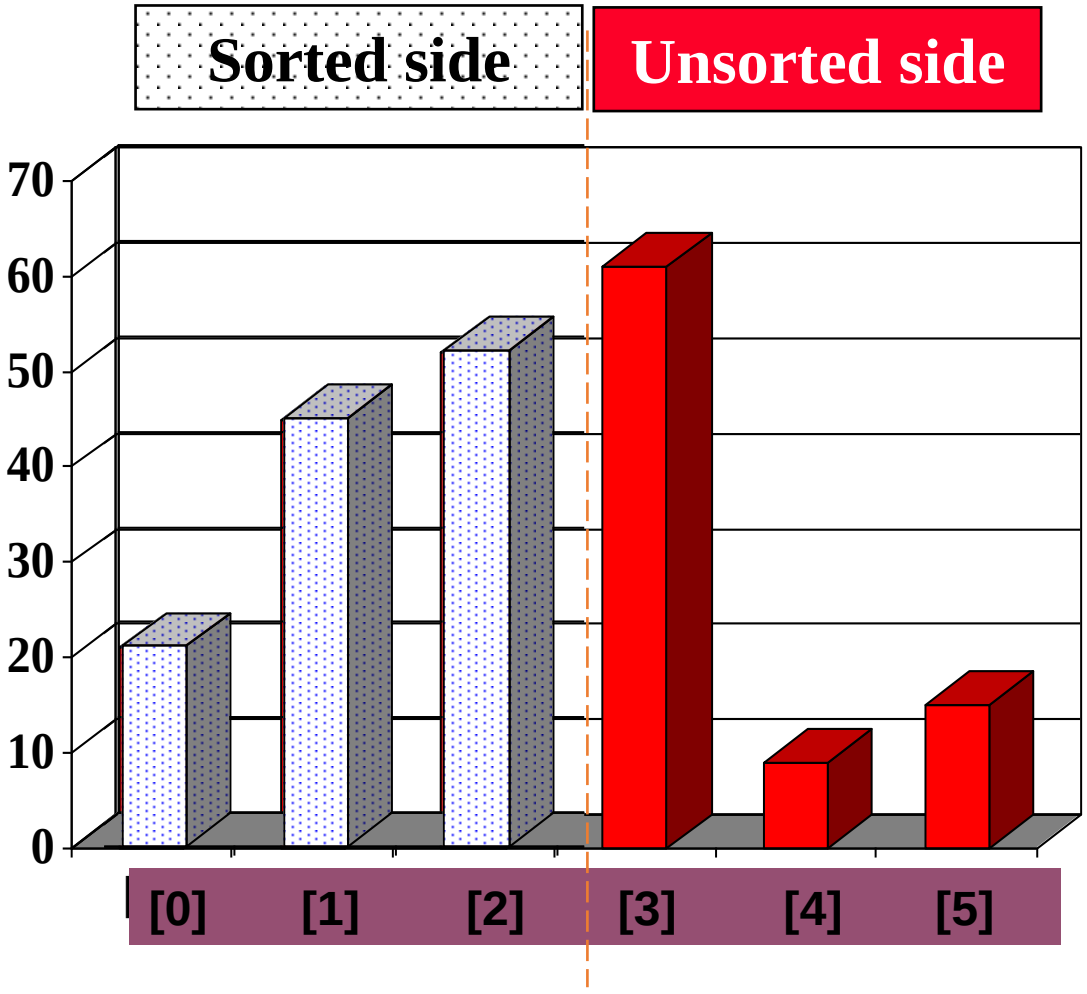
- ... و آن عدد را در جایی در بخش مرتب شده قرار می دهد که ترتیب به صورت درستی حاصل شود.

مرتب سازی درجی



- ... و آن عدد را در جایی در بخش مرتب شده قرار می دهد که ترتیب به صورت درستی حاصل شود.

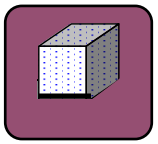
مرتب سازی درجی

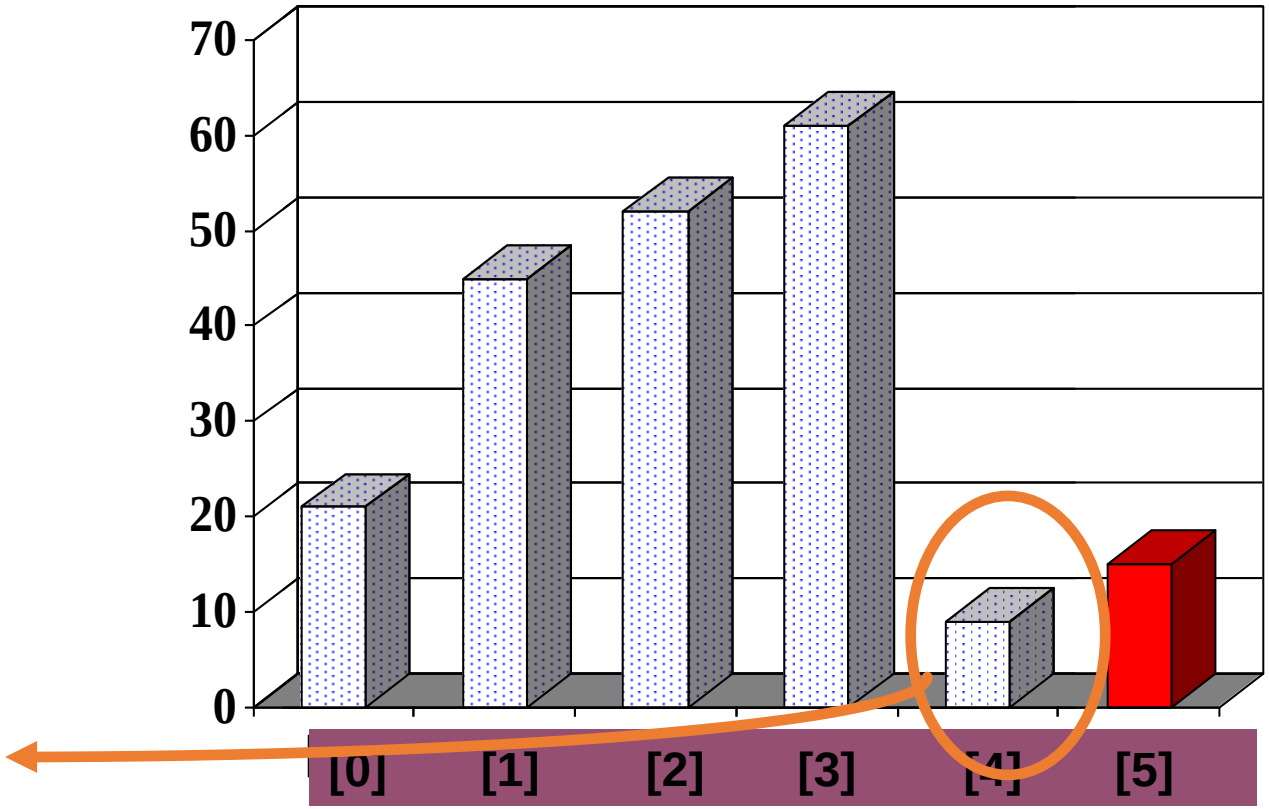


- گاهی اوقات ما خوش شانس هستیم و نیازی به جابجایی این عدد جدید نیست.

مرتب سازی درجی

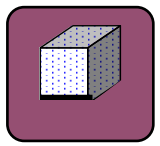
- این عدد جدید را در یک خانه جداگانه کپی نمایید.

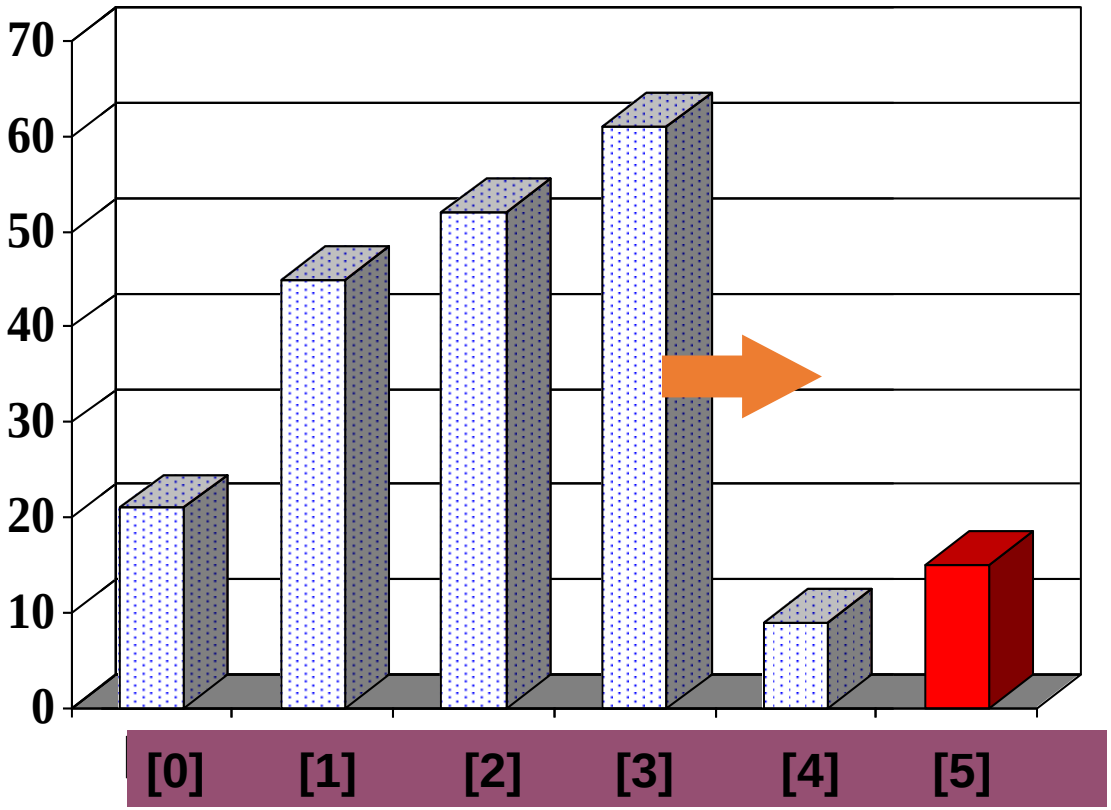




مرتب سازی درجی

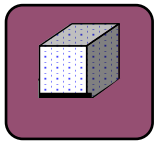
- عناصر بخش مرتب شده را یکی یکی به سمت راست شیف دهید، تا جا برای این خانه جدید باز شود.

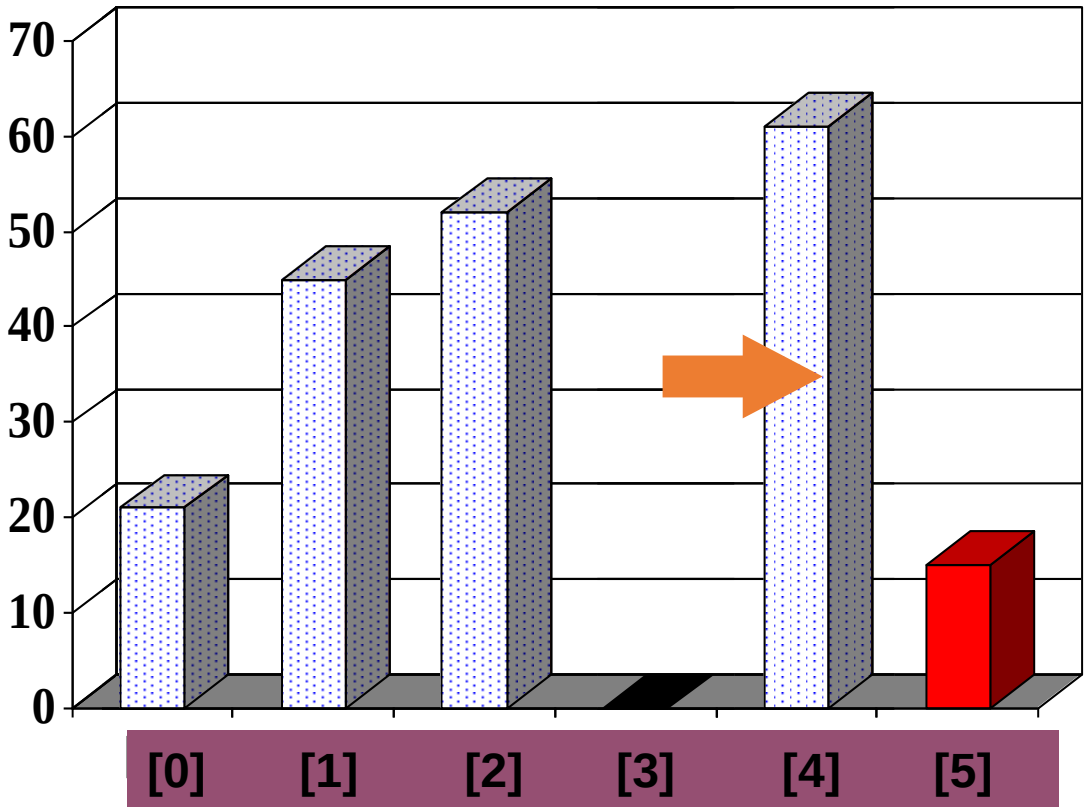




مرتب سازی درجی

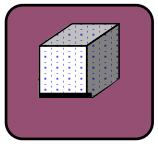
- عناصر بخش مرتب شده را یکی یکی به سمت راست شیف دهید، تا جا برای این خانه جدید باز شود.

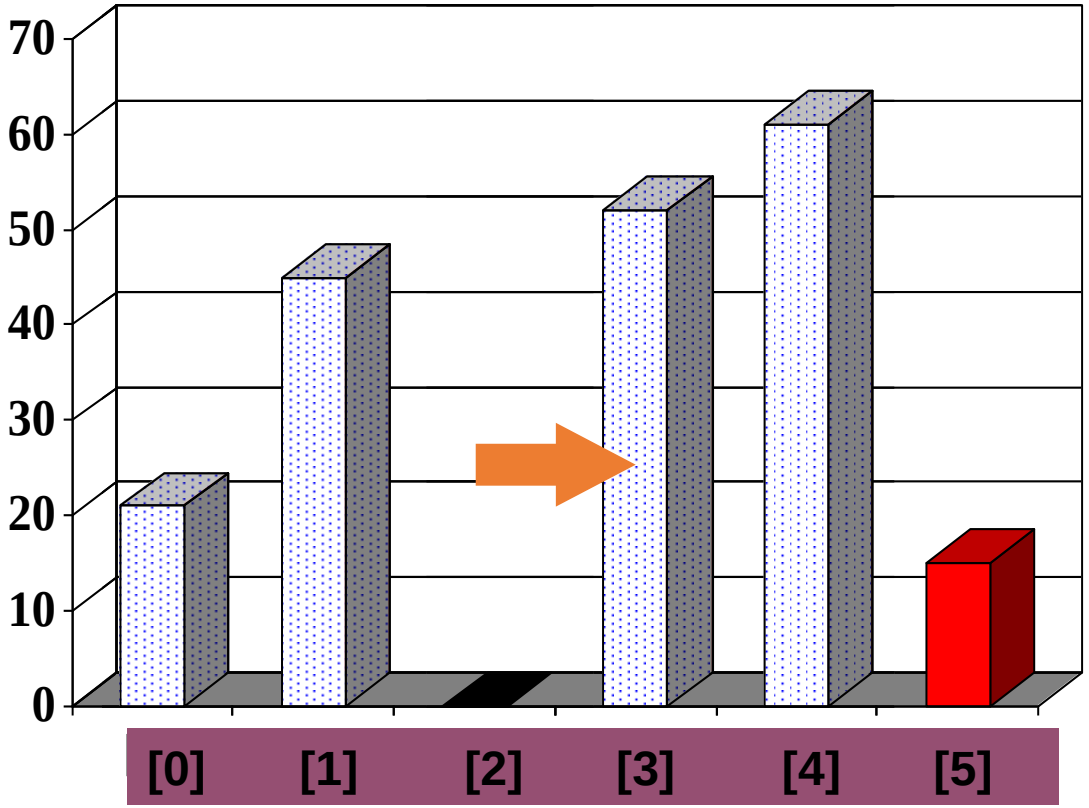




مرتب سازی درجی

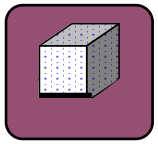
- به شیفت دادن ادامه میدهیم.

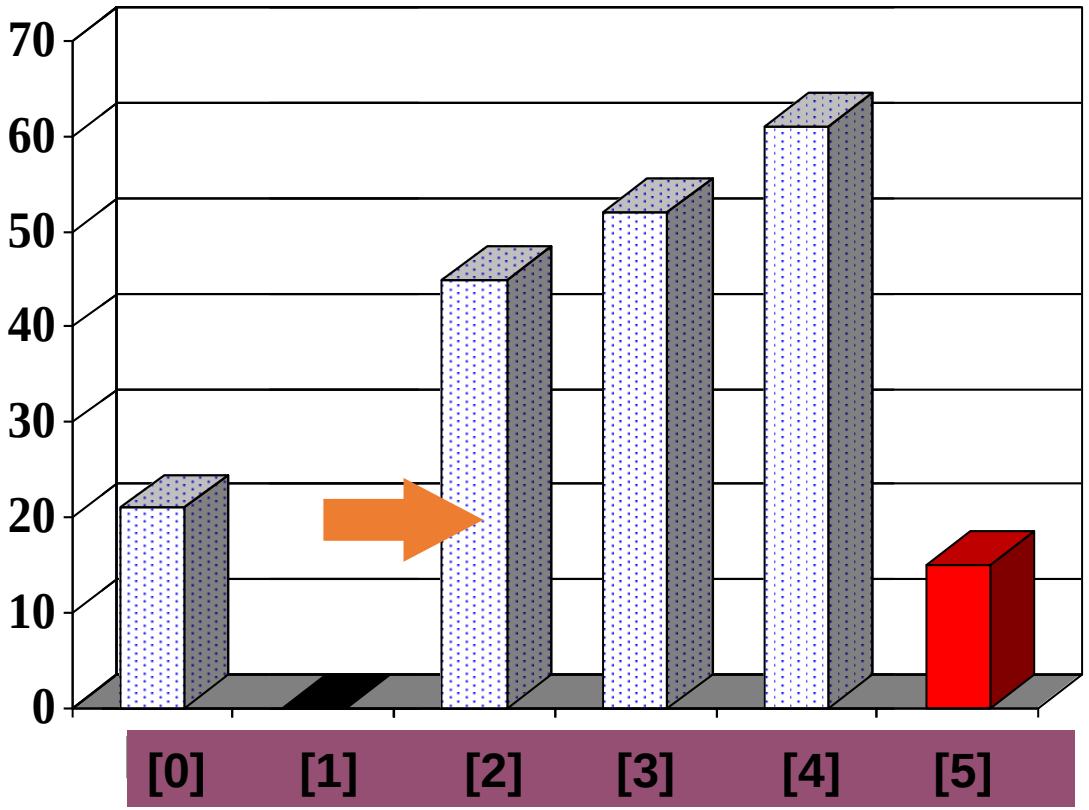




مرتب سازی درجی

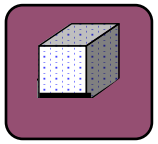
- به شیفت دادن ادامه میدهیم.

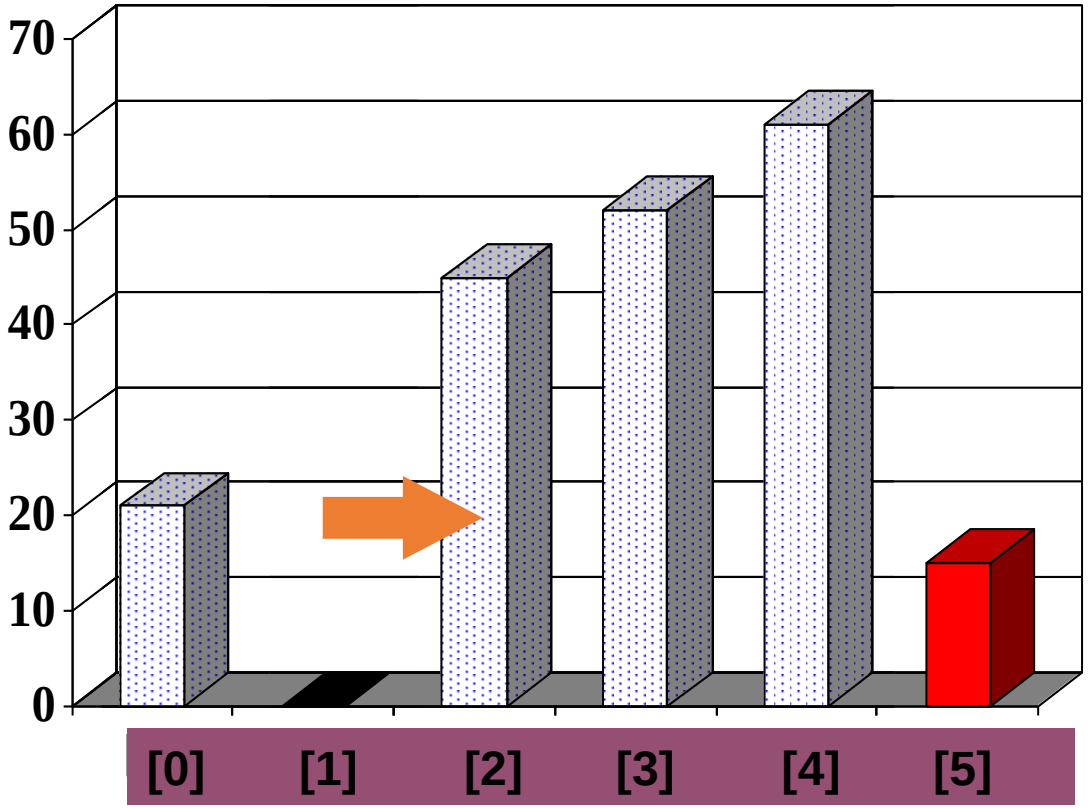




مرتب سازی درجی

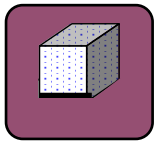
- به شیفت دادن ادامه میدهیم.

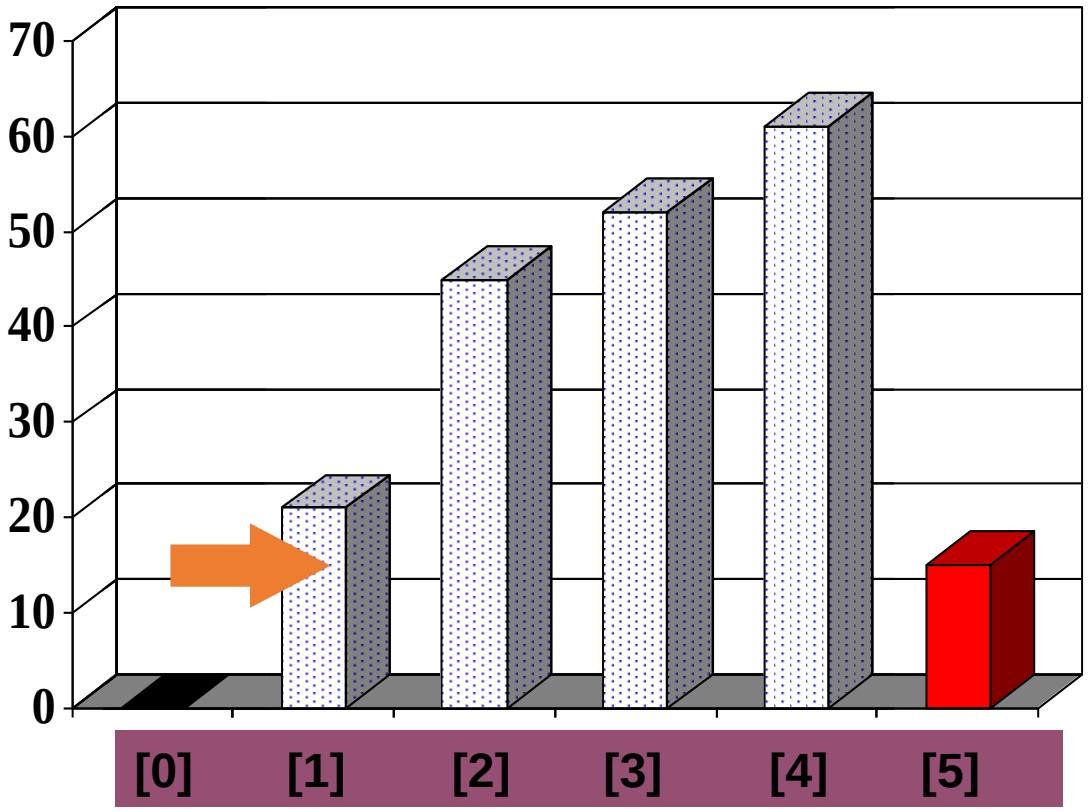




مرتب سازی درجی

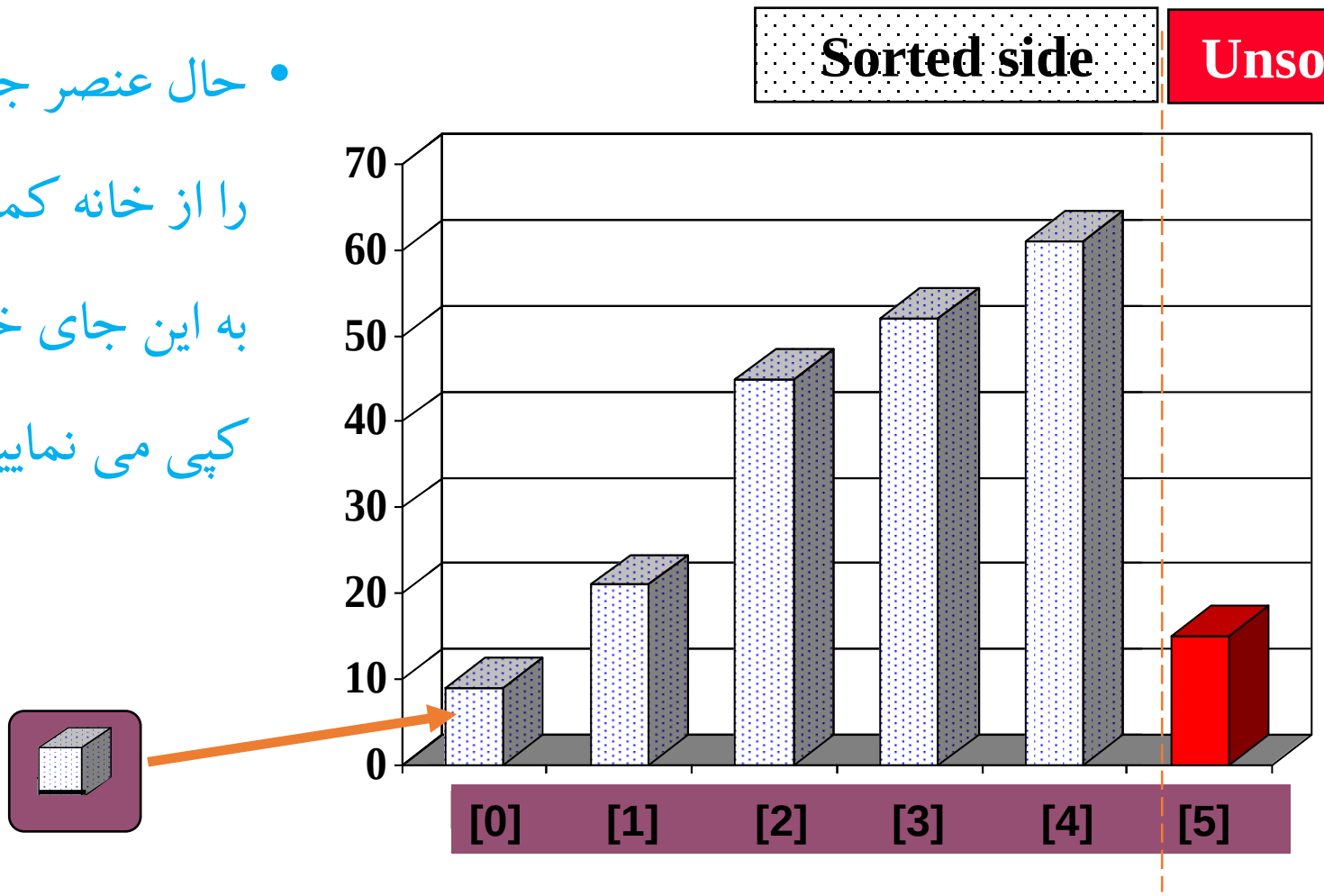
- تا اینکه به محل مناسب برای قرار گرفته عنصر جدید برسیم.





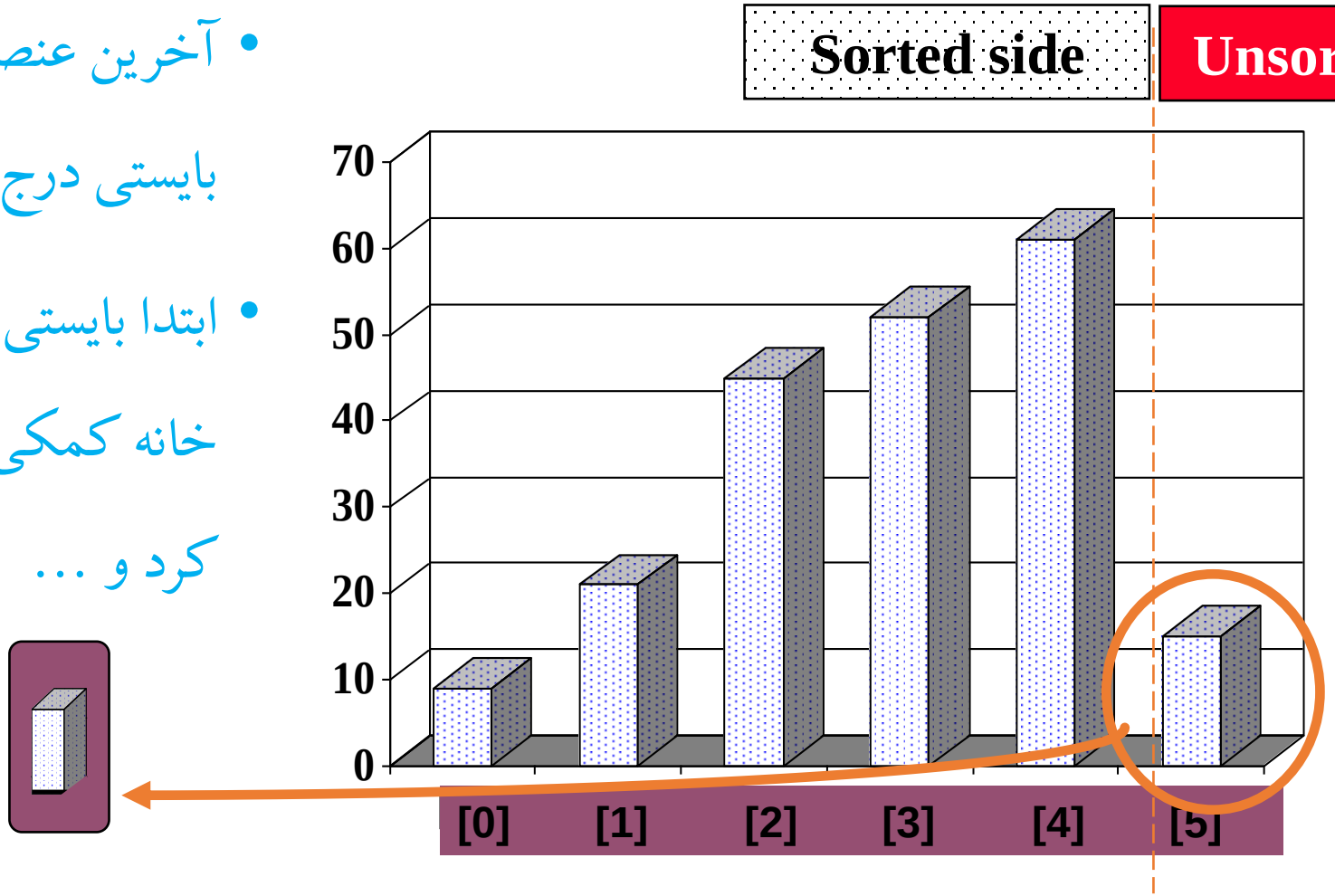
مرتب سازی درجی

- حال عنصر جدید را از خانه کمکی به این جای خالی کپی می نماییم.



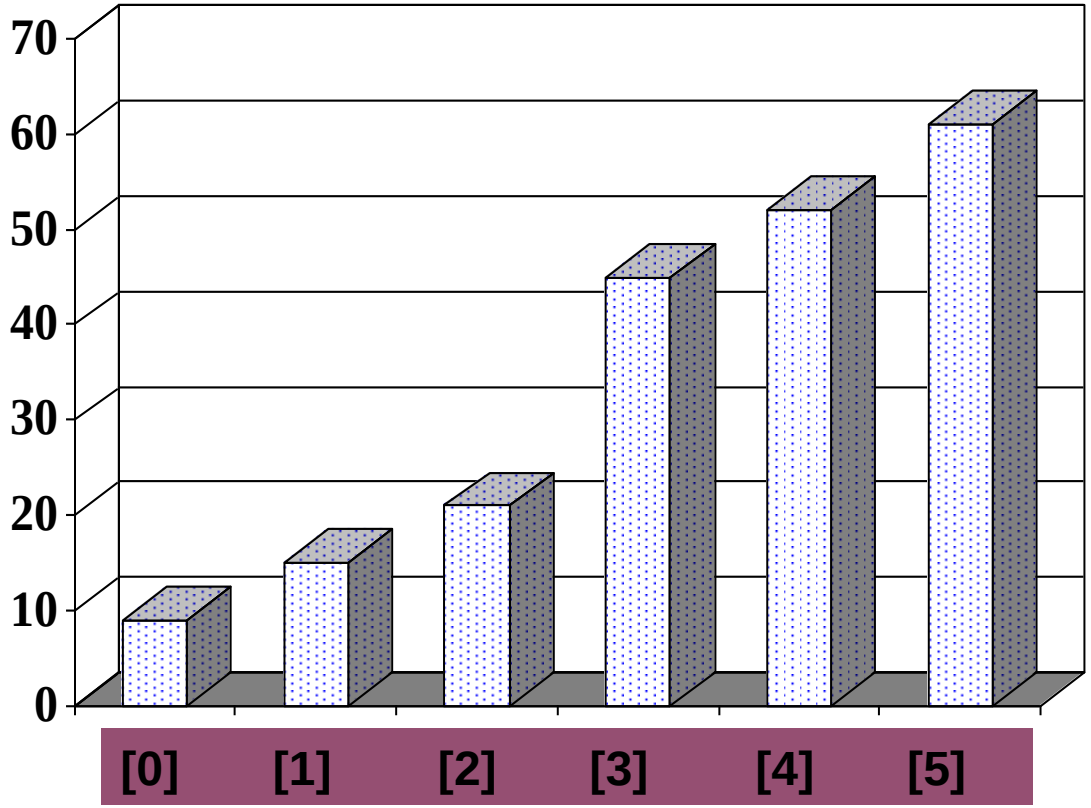
مرتب سازی درجی

- آخرین عنصر هم بایستی درج شود.
- ابتدا بایستی آنرا در خانه کمکی کپی کرد و ...



مرتب سازی درجی

- آرایه مرتب شده



الگوریتم مرتب درجی

```
template <class Item>
void insertion_sort(Item data[ ], size_t n)
{
    size_t i, j;
    Item temp;

    if(n < 2) return;  // nothing to sort!!

    for(i = 1; i < n; ++i)
    {
      // take next item at front of unsorted part of array
      // and insert it in appropriate location in sorted part of array
      temp = data[i];
      for(j = i; data[j-1] > temp and j > 0; --j)
         data[j] = data[j-1];  // shift element forward

      data[j] = temp;
     }
}
```

تحلیل زمانی الگوریتم مرتب درجی

```
template <class Item>
void insertion_sort(Item data[ ], size_t n)
{
    size_t i, j;
    Item temp;

    if(n < 2) return;  // nothing to sort!!

    for(i = 1; i < n; ++i)
    {
      // take next item at front of unsorted part of array
      // and insert it in appropriate location in sorted part of array
      temp = data[i];
      for(j = i; data[j-1] > temp and j > 0; --j)
        data[j] = data[j-1];  // shift element forward

      data[j] = temp;
    }
}
```

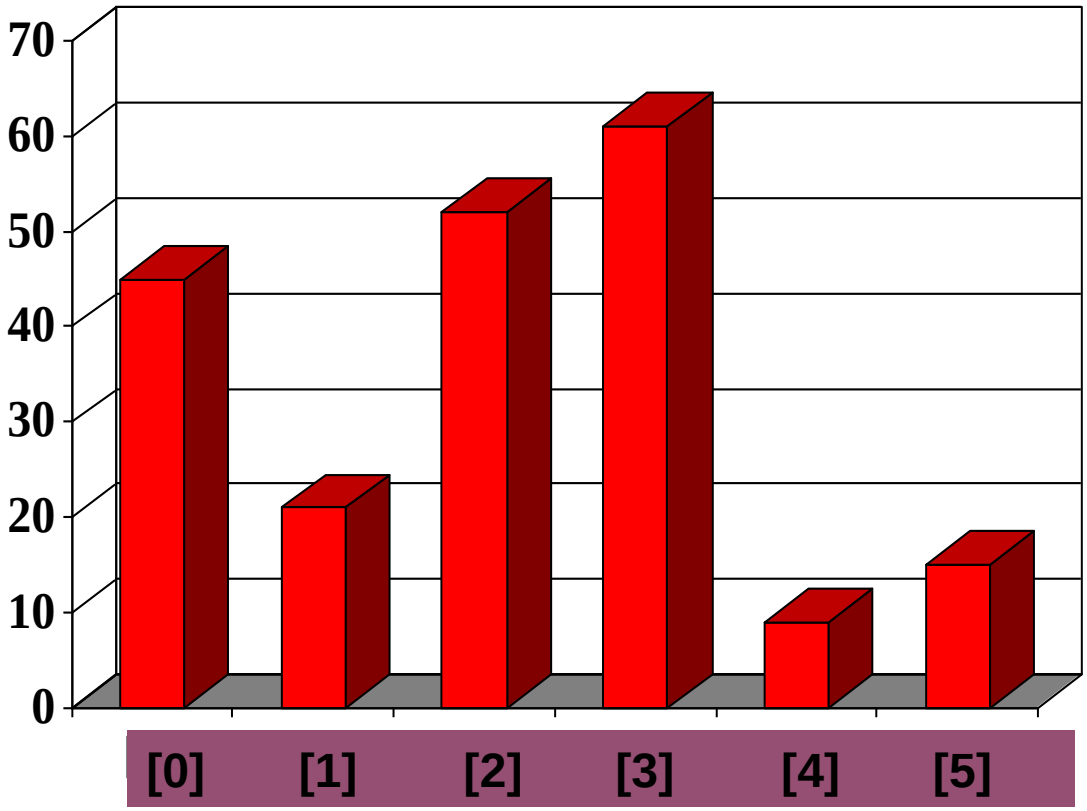
Outer loop: O(n)

Inner loop: O(n)

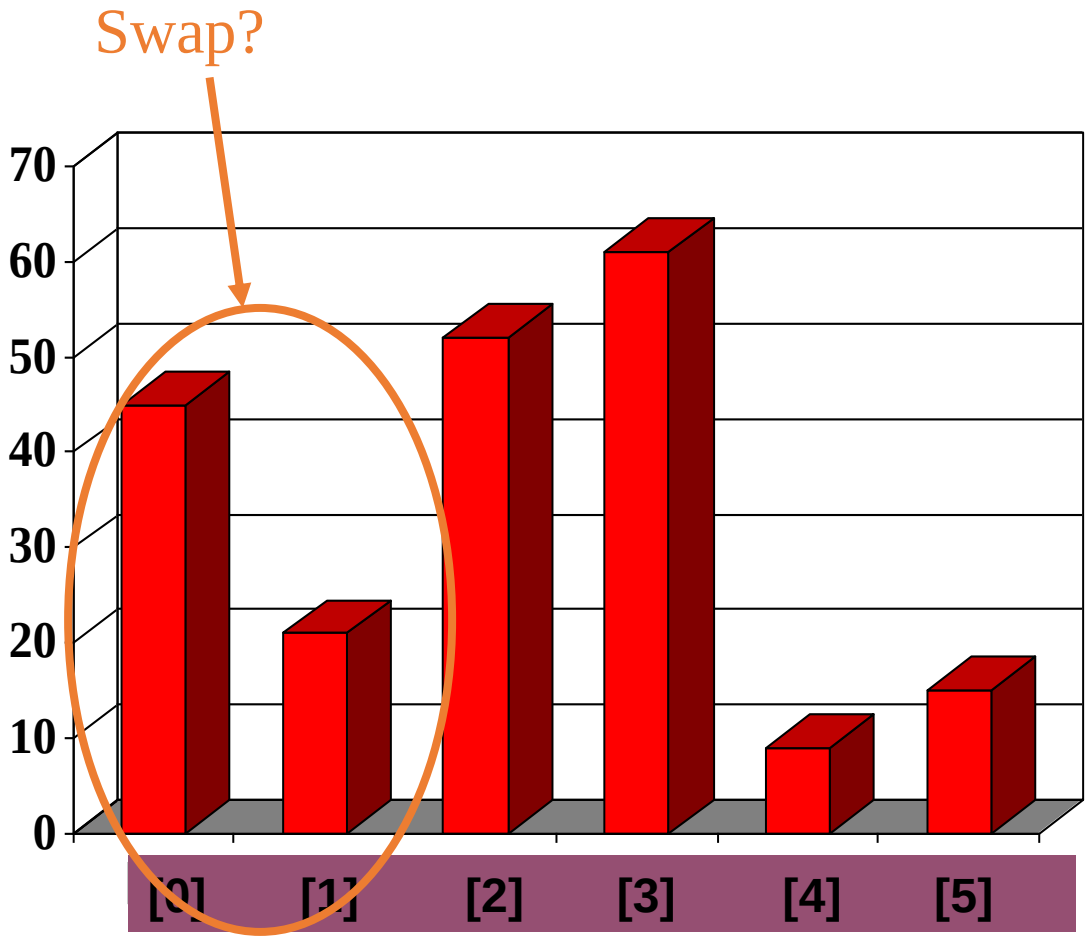
Overall : $O(n^2)$

مرتب سازی حبابی

- الگوریتم مرتب سازی حبابی به جفت اعداد پشت سر هم در آرایه نگاه می کند و در صورت نیاز جای آنها را با هم عوض می نماید.

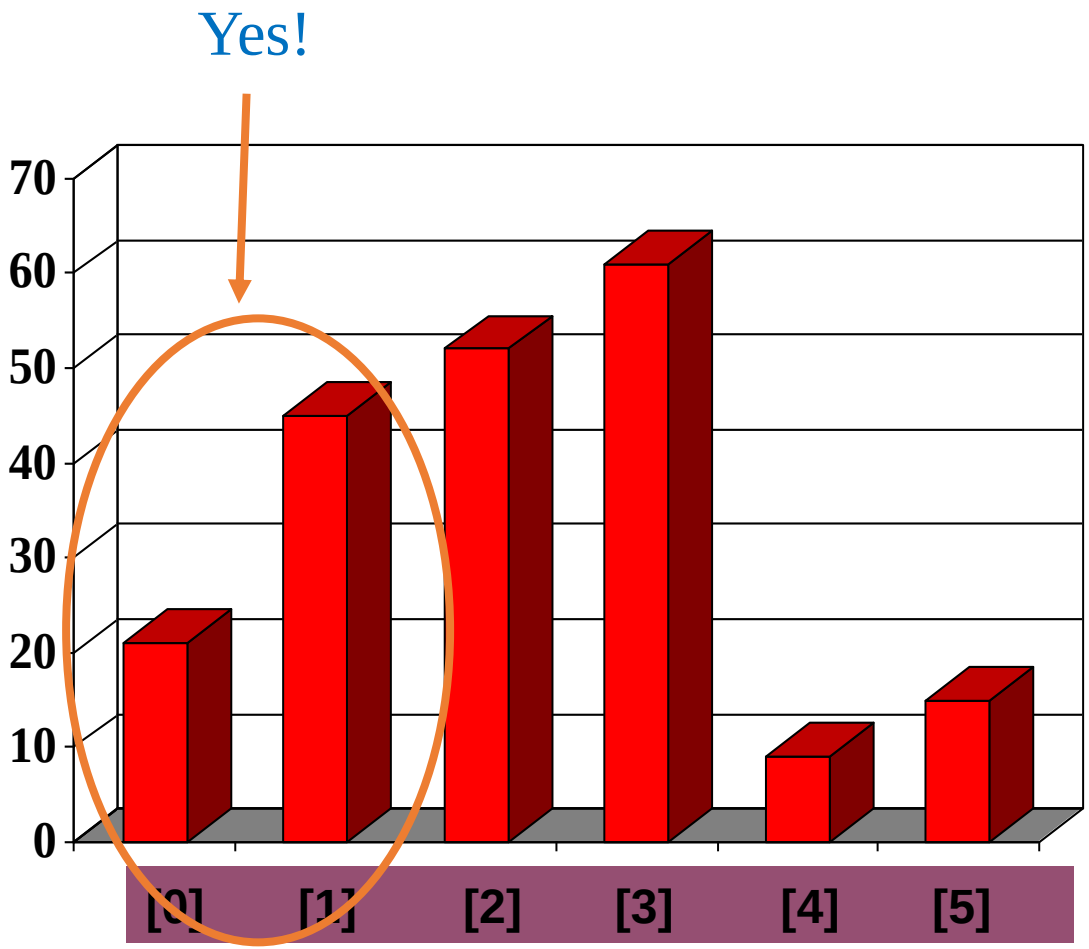


مرتب سازی حبابی



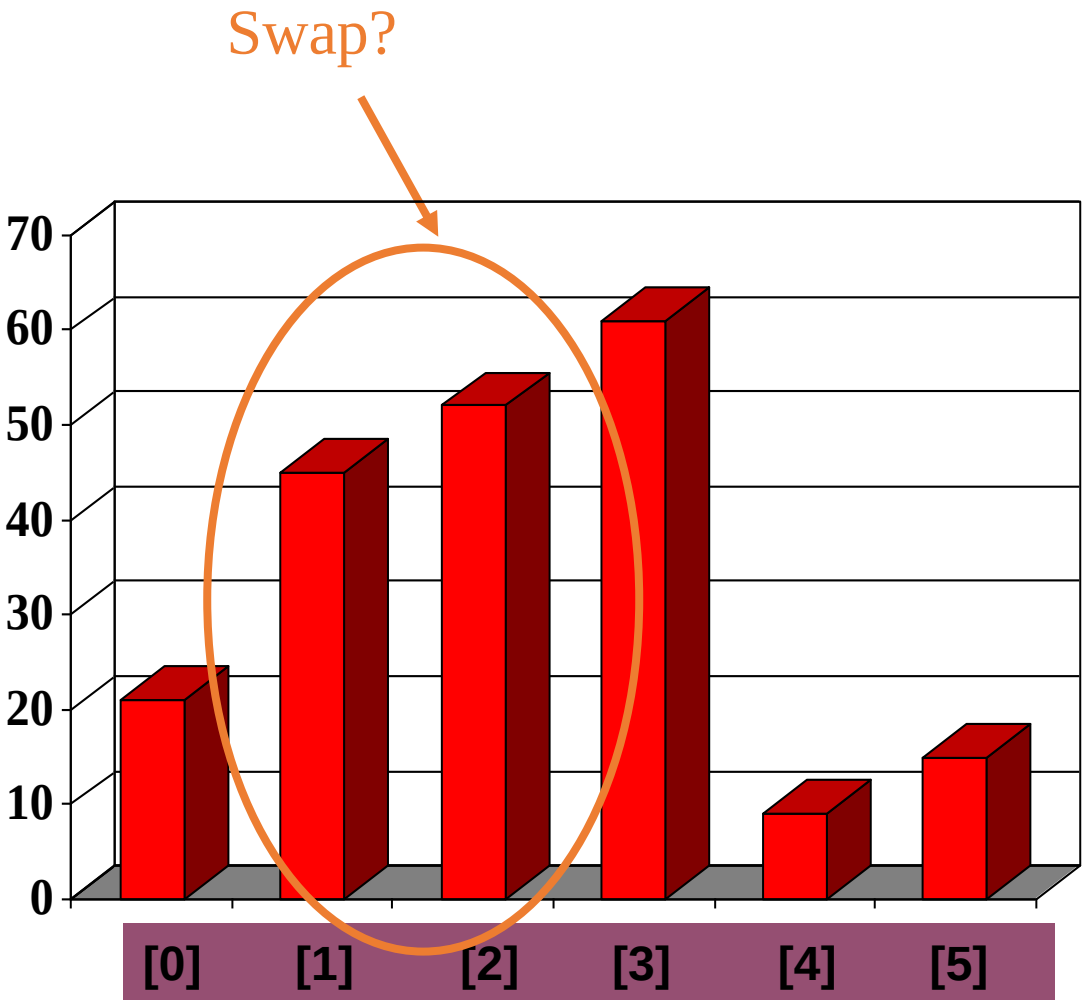
- الگوریتم مرتب سازی حبابی به جفت اعداد پشت سر هم در آرایه نگاه می کند و در صورت نیاز جای آنها را با هم عوض می نماید.

مرتب سازی حبابی



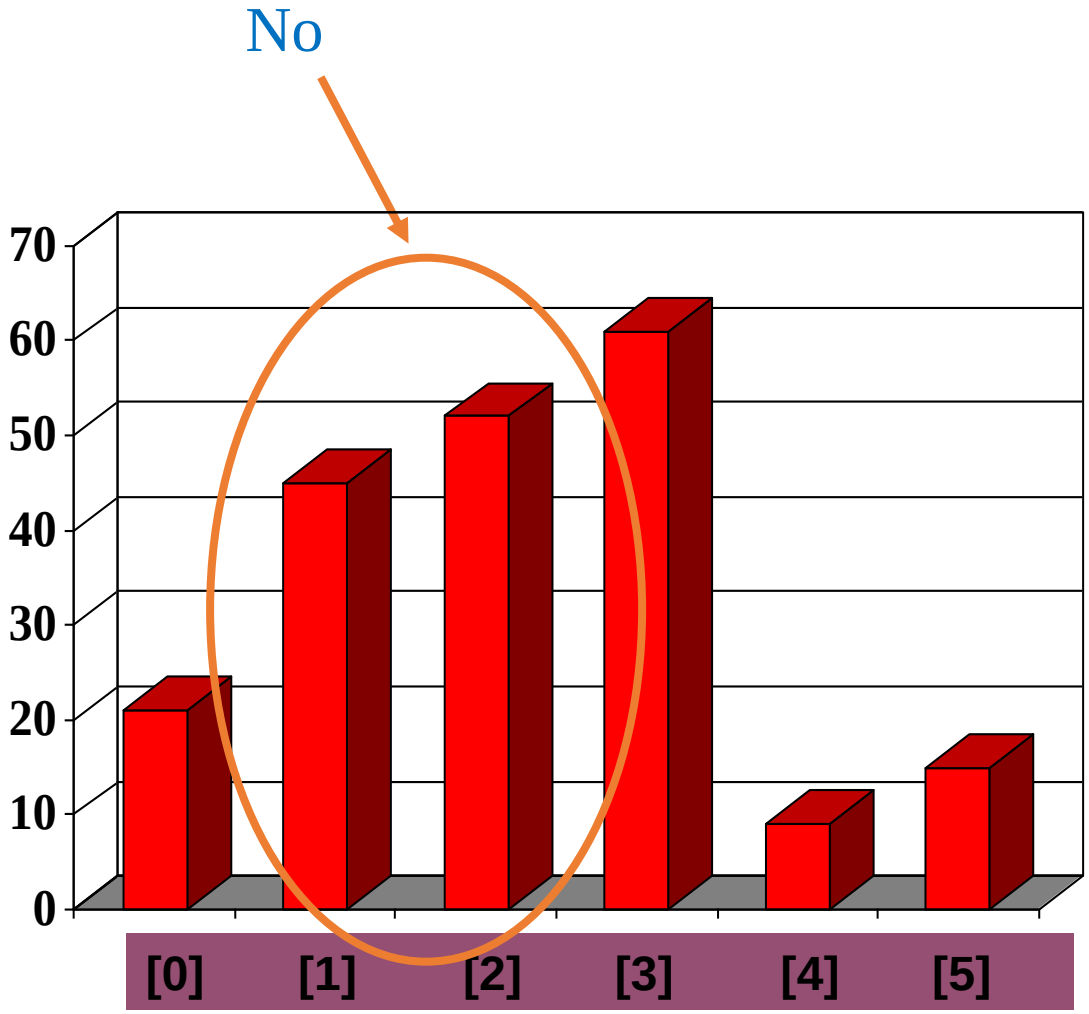
- الگوریتم مرتب سازی حبابی به جفت اعداد پشت سر هم در آرایه نگاه می کند و در صورت نیاز جای آنها را با هم عوض می نماید.

مرتب سازی حبابی



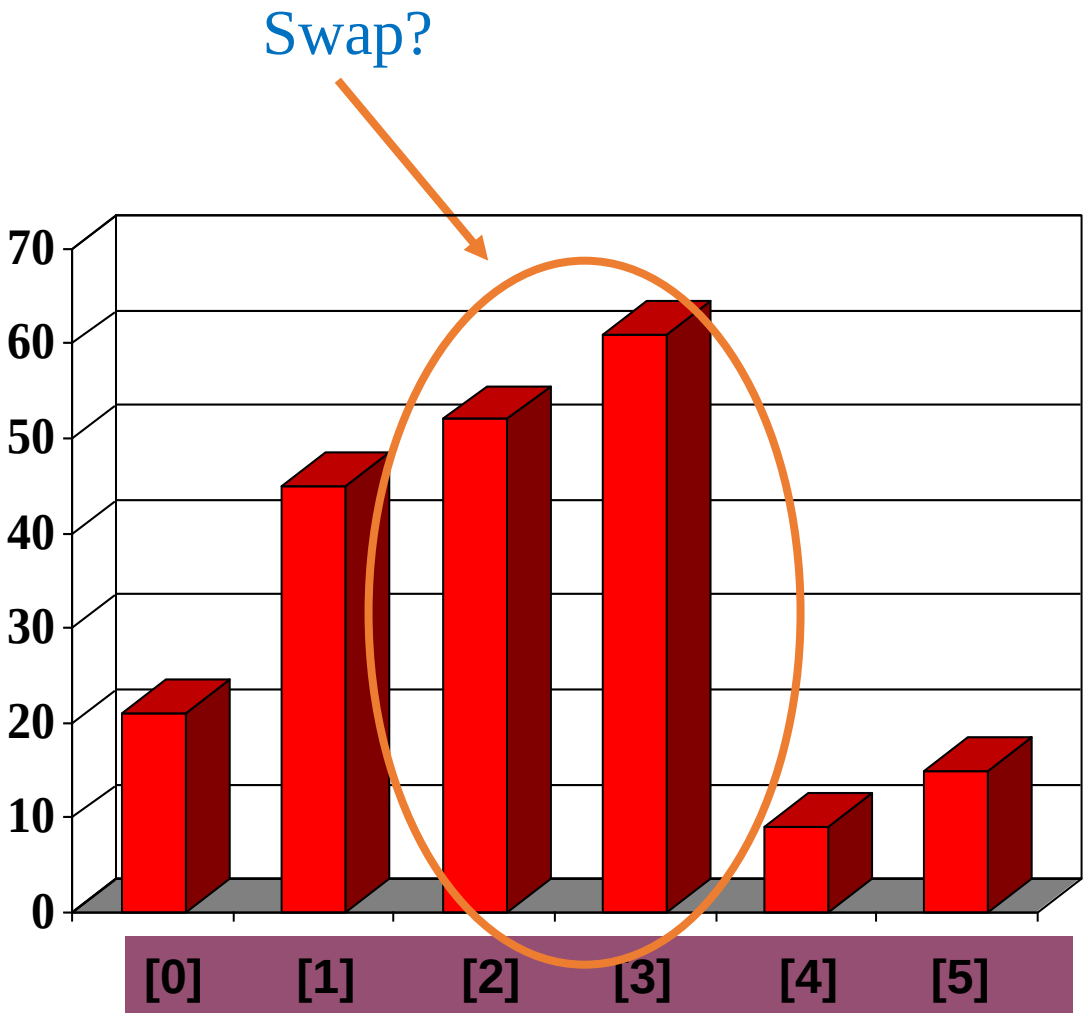
- الگوریتم مرتب سازی حبابی به جفت اعداد پشت سر هم در آرایه نگاه می کند و در صورت نیاز جای آنها را با هم عوض می نماید.

مرتب سازی حبابی



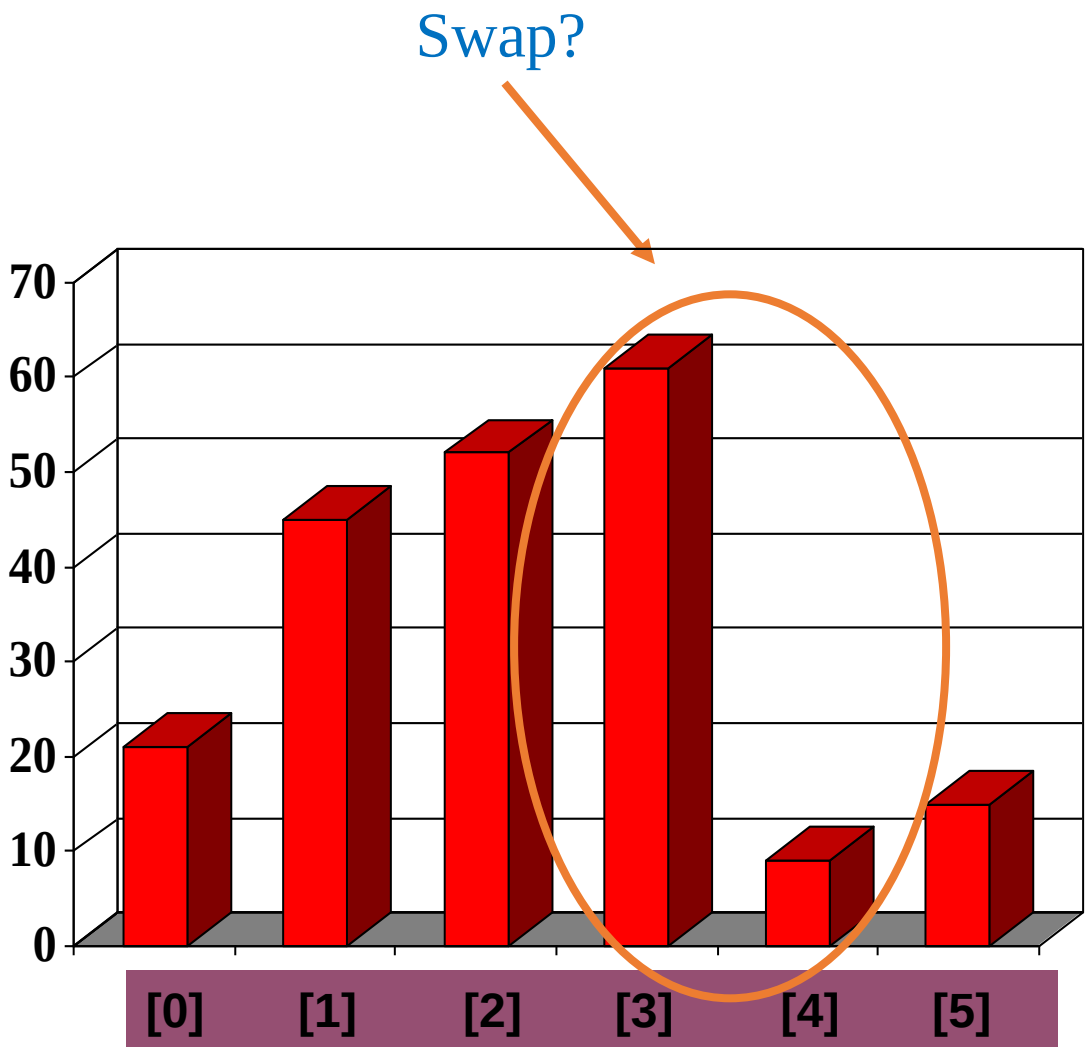
- الگوریتم مرتب سازی حبابی به جفت اعداد پشت سر هم در آرایه نگاه می کند و در صورت نیاز جای آنها را با هم عوض می نماید.

مرتب سازی حبابی



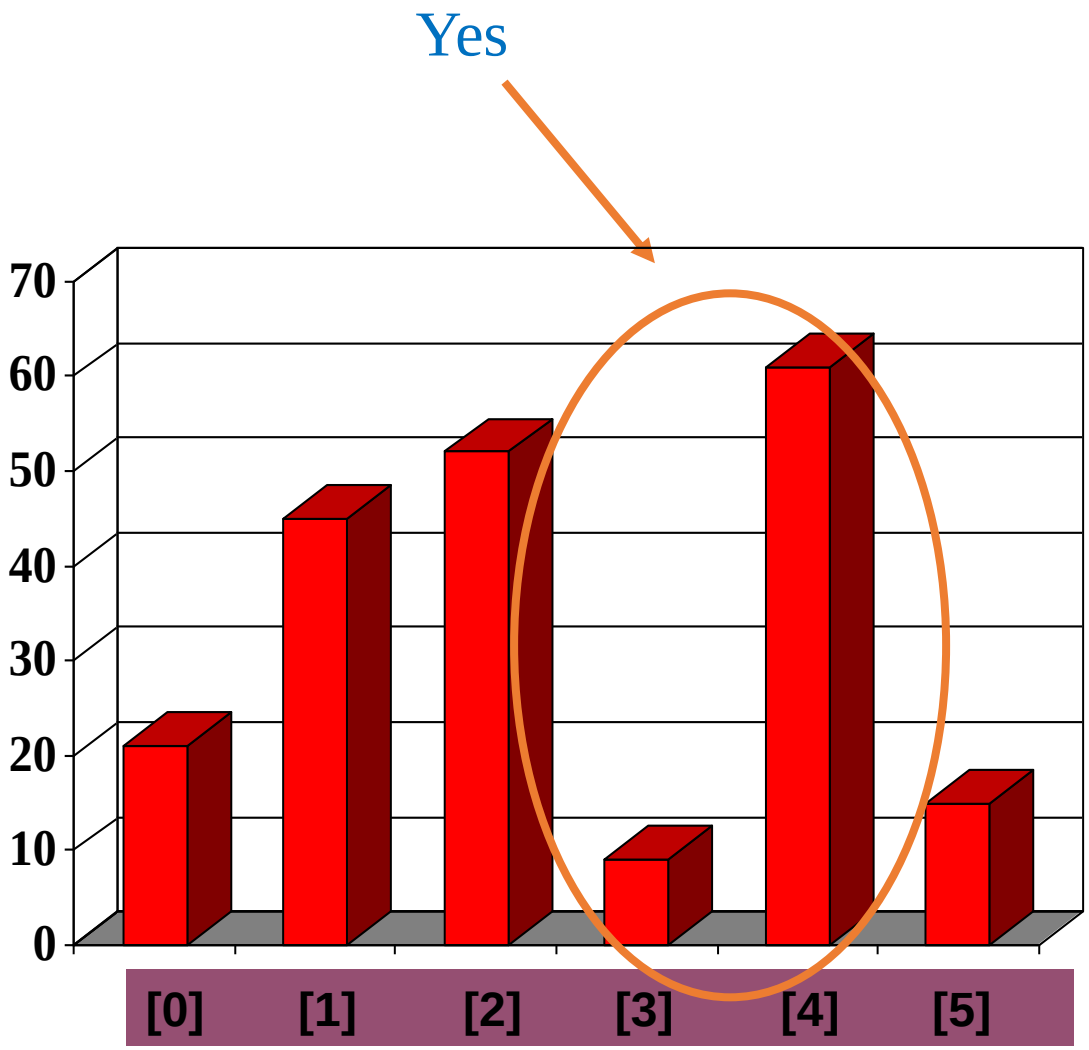
- الگوریتم مرتب سازی حبابی به جفت اعداد پشت سر هم در آرایه نگاه می کند و در صورت نیاز جای آنها را با هم عوض می نماید.

مرتب سازی حبابی



- الگوریتم مرتب سازی حبابی به جفت اعداد پشت سر هم در آرایه نگاه می کند و در صورت نیاز جای آنها را با هم عوض می نماید.

مرتب سازی حبابی



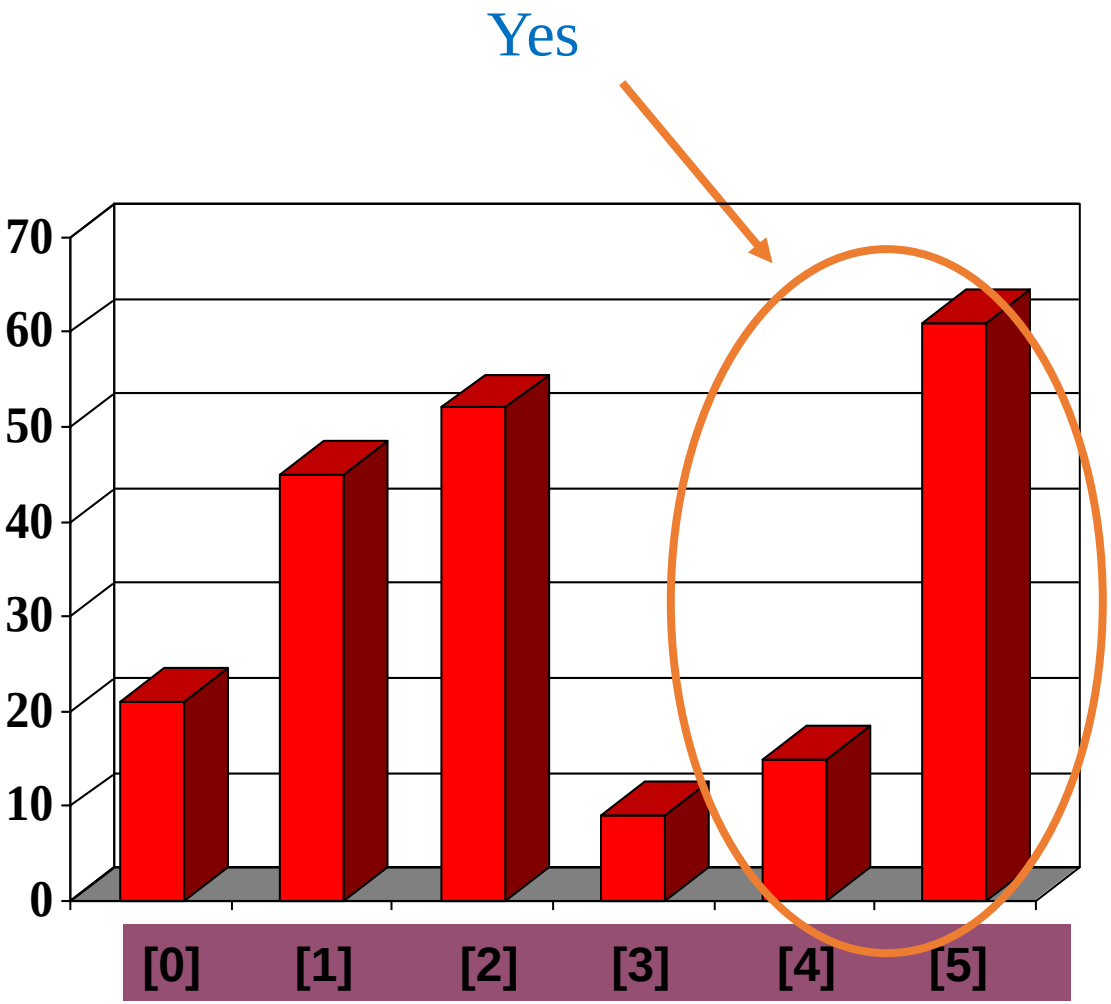
- الگوریتم مرتب سازی حبابی به جفت اعداد پشت سر هم در آرایه نگاه می کند و در صورت نیاز جای آنها را با هم عوض می نماید.

مرتب سازی حبابی

- الگوریتم مرتب سازی حبابی به جفت اعداد پشت سر هم در آرایه نگاه می کند و در صورت نیاز جای آنها را با هم عوض می نماید.

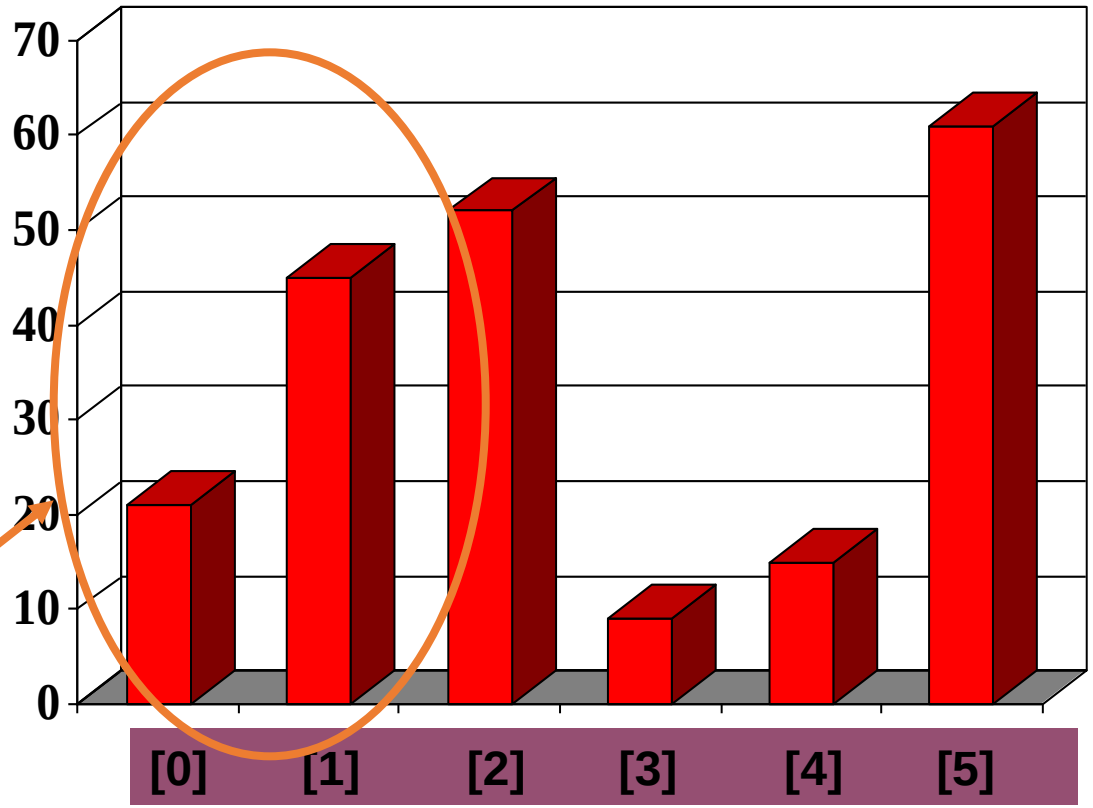
مرتب سازی حبابی

- حال بزرگترین عدد در جای خود قرار دارد.



مرتب سازی حبابی

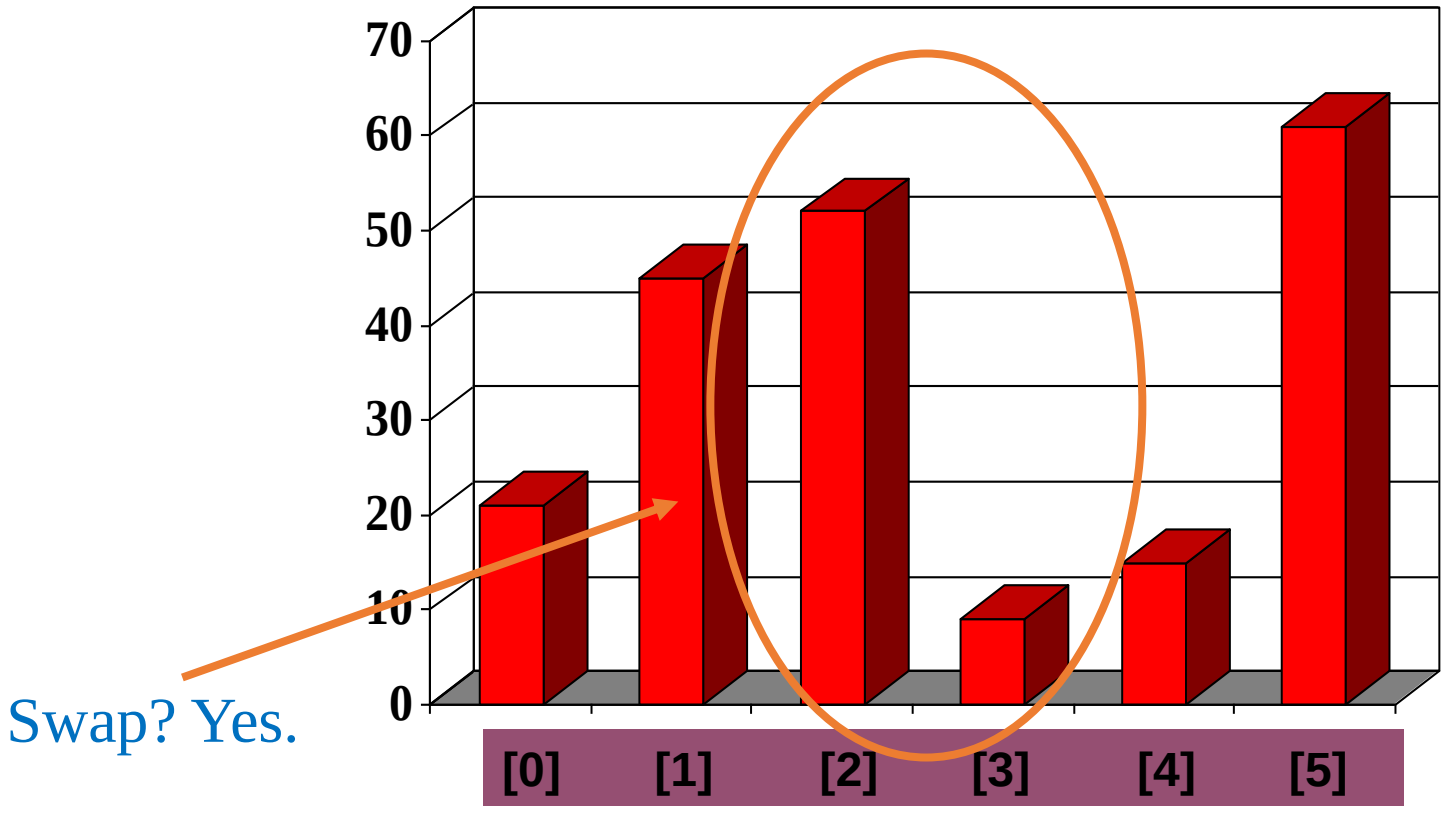
- تکرار کنید



Swap? No.

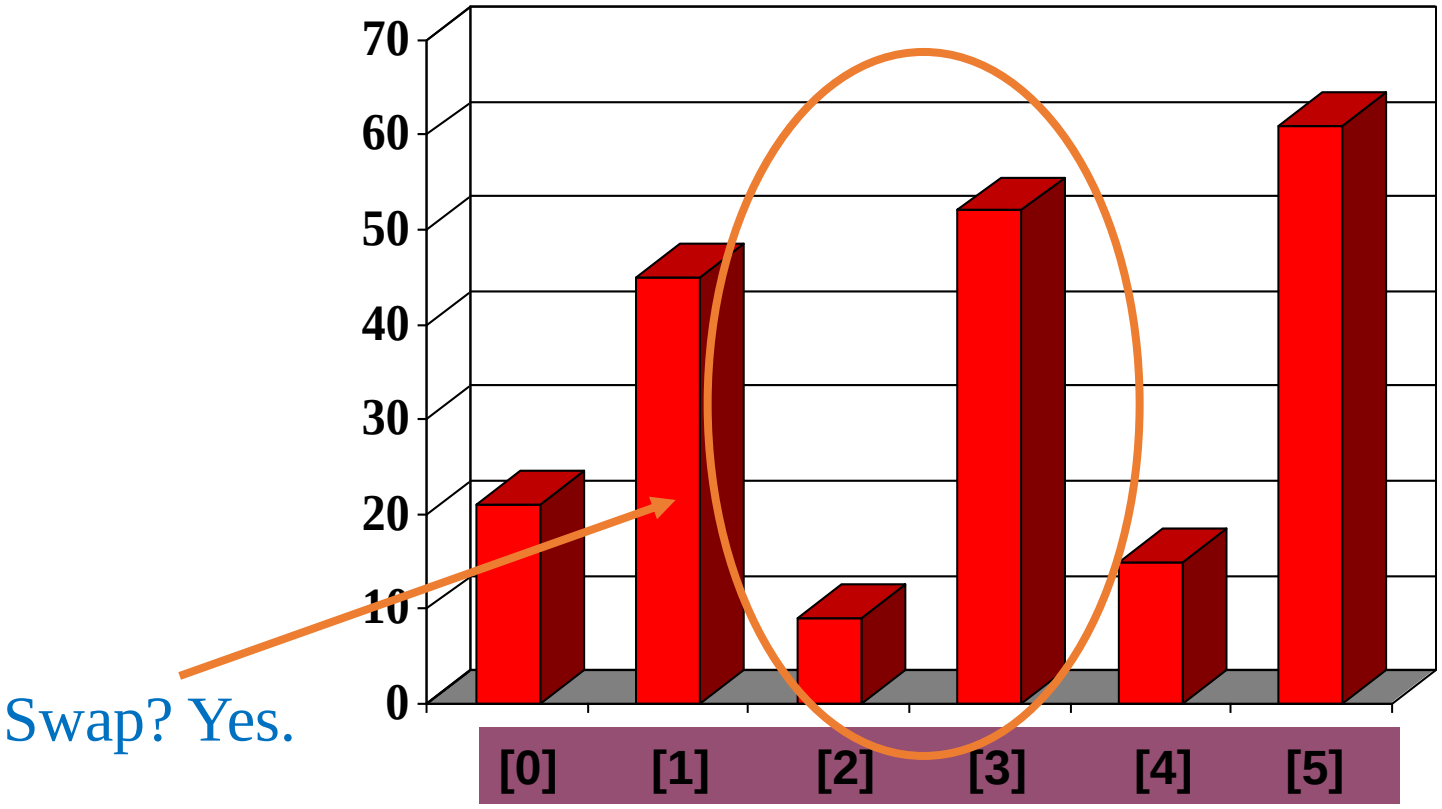
مرتب سازی حبابی

- تکرار کنید



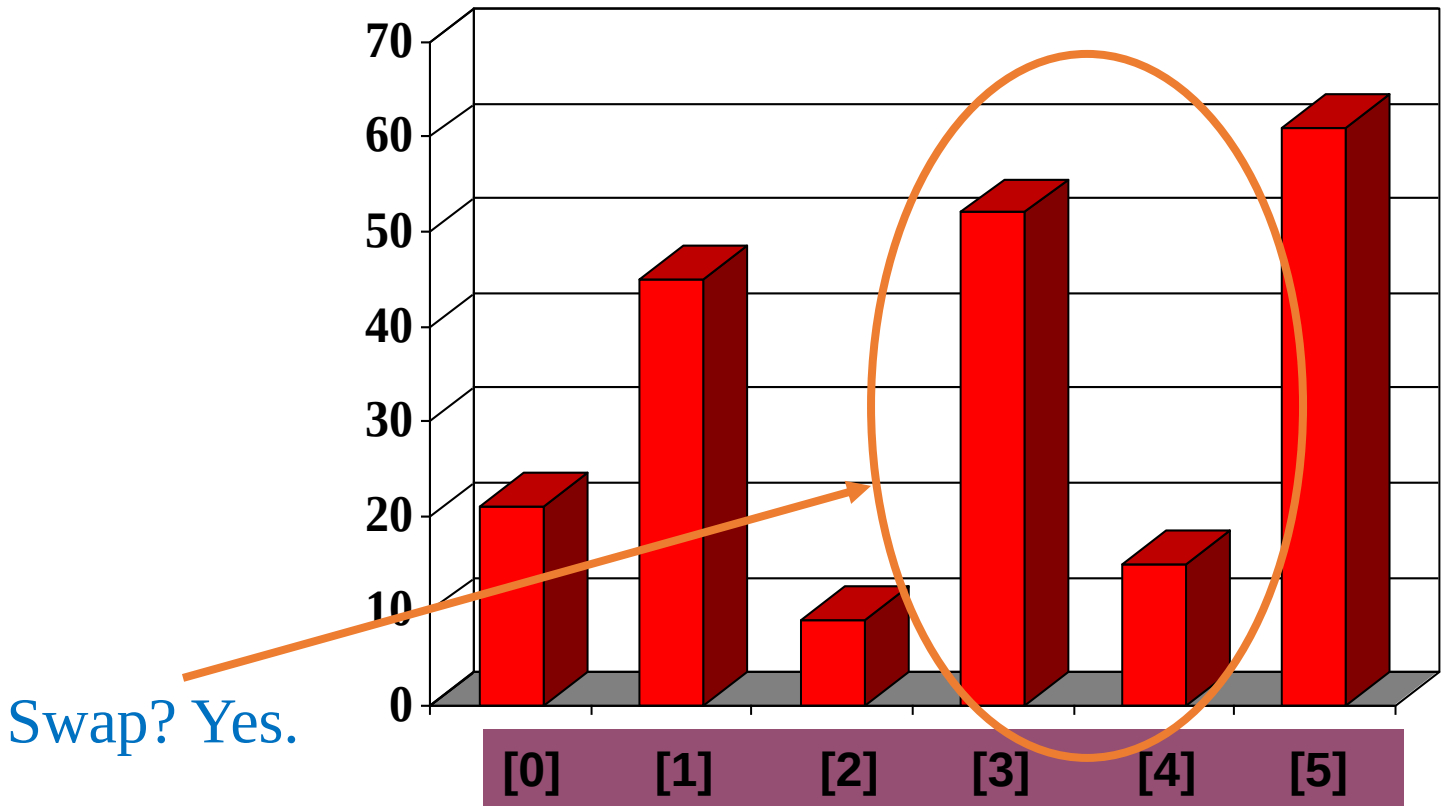
مرتب سازی حبابی

- تکرار کنید



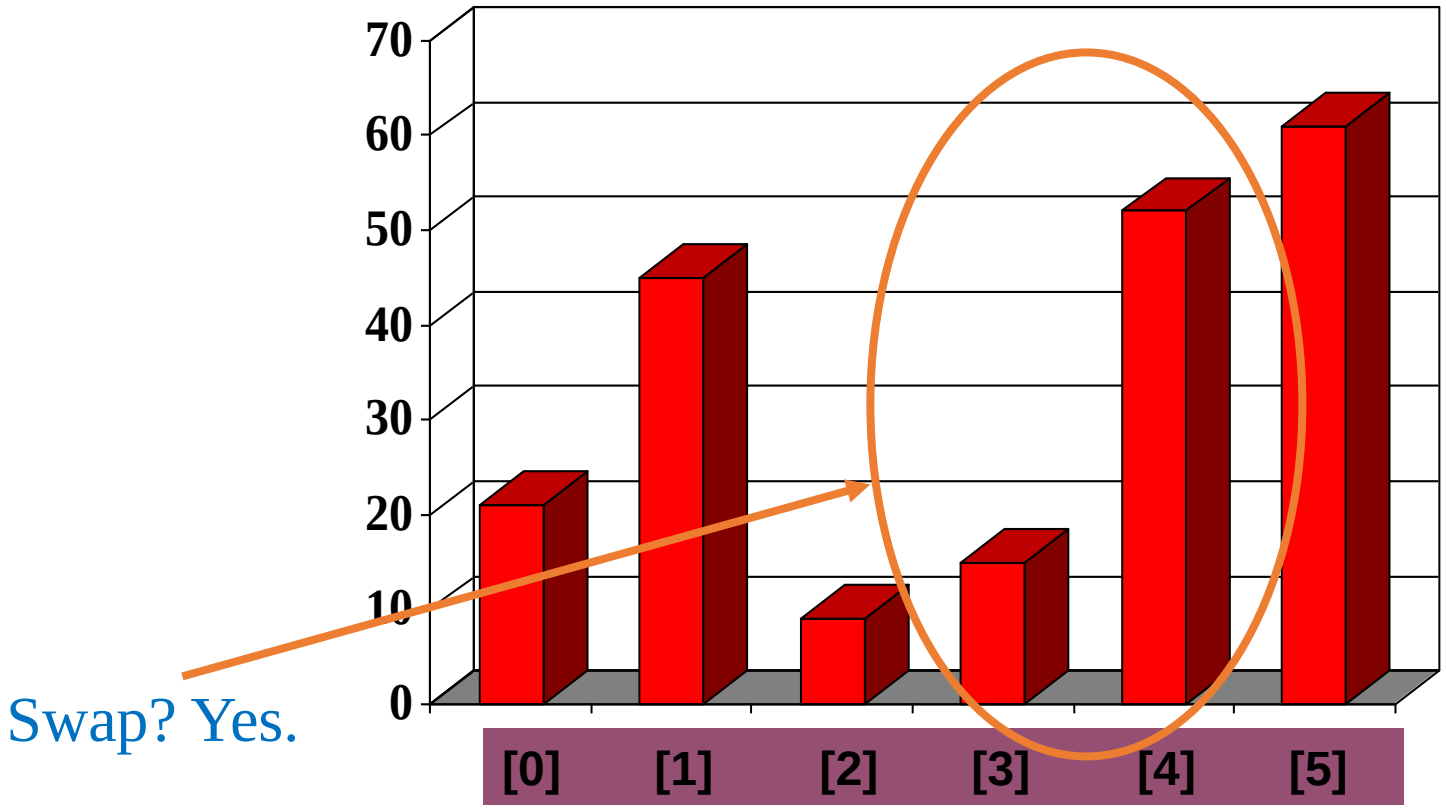
مرتب سازی حبابی

- تکرار کنید



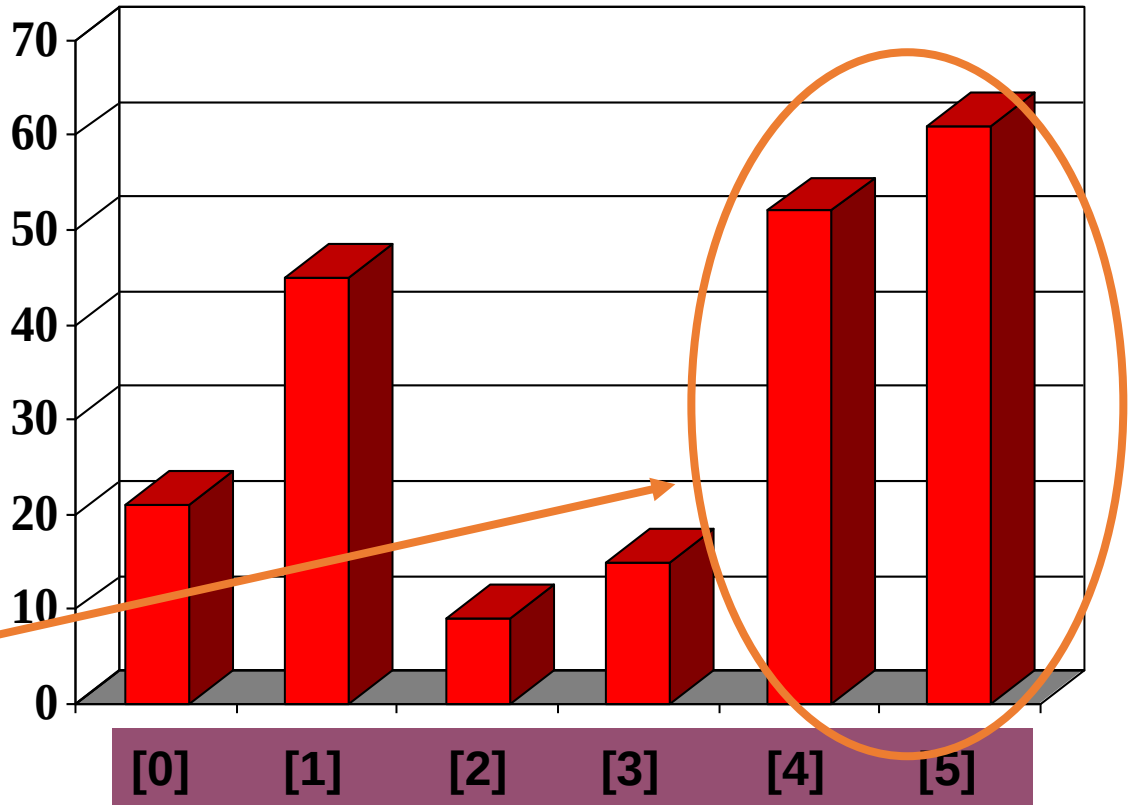
مرتب سازی حبابی

- تکرار کنید



مرتب سازی حبابی

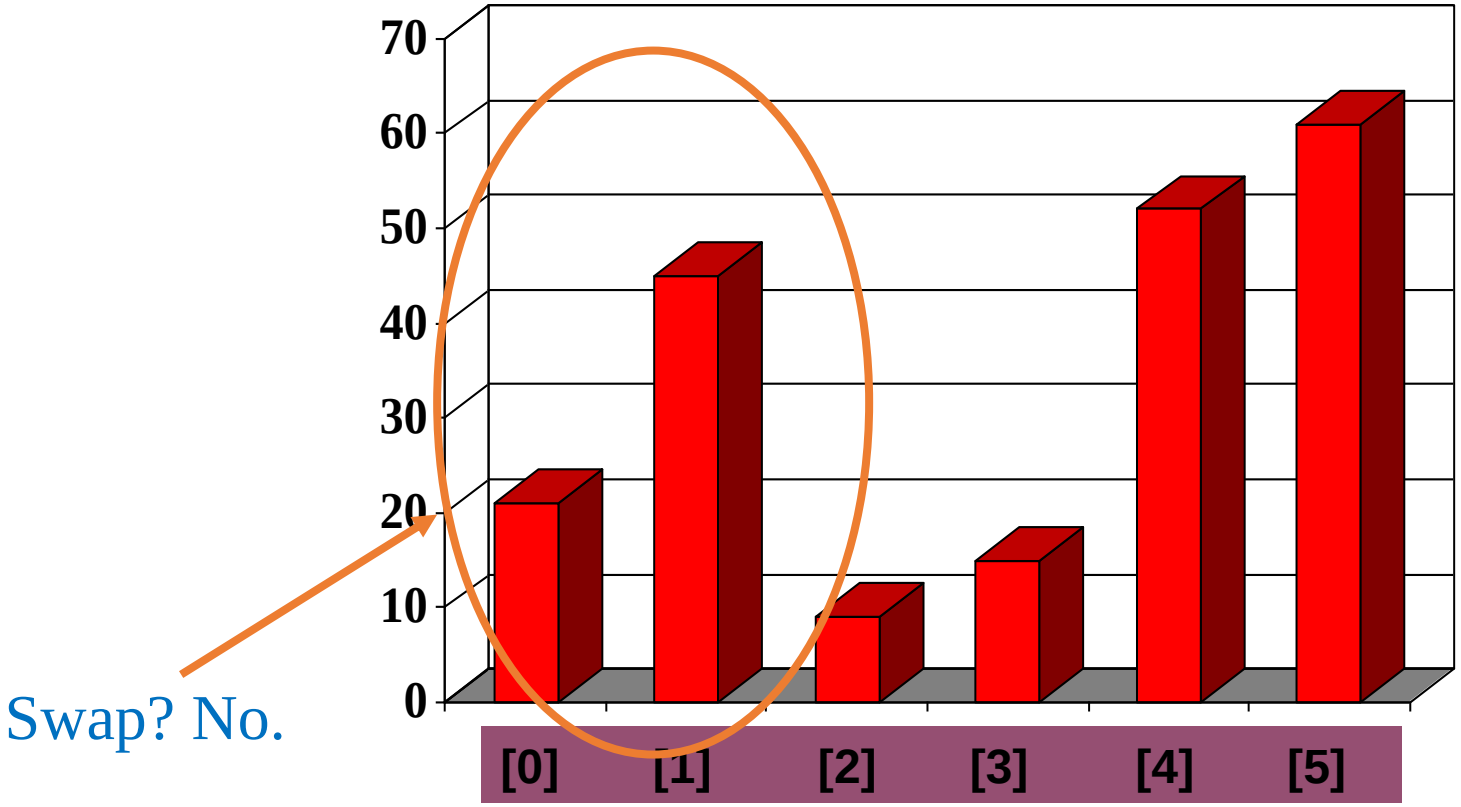
- تکرار کنید



Swap? No.

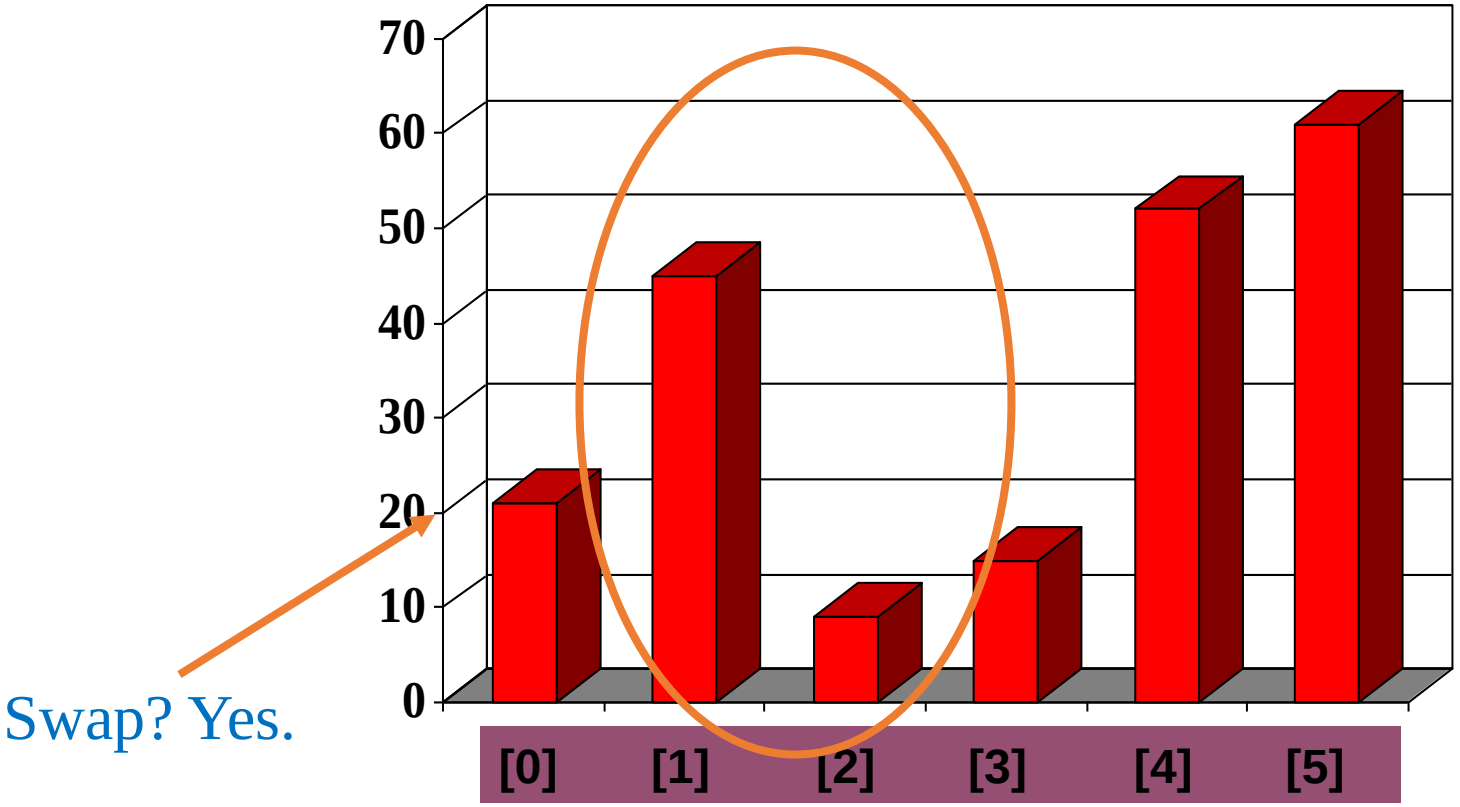
مرتب سازی حبابی

- روی آرایه به تعداد n-1 بار از ابتدا تکرار و در صورت نیاز جابجا نمایید.



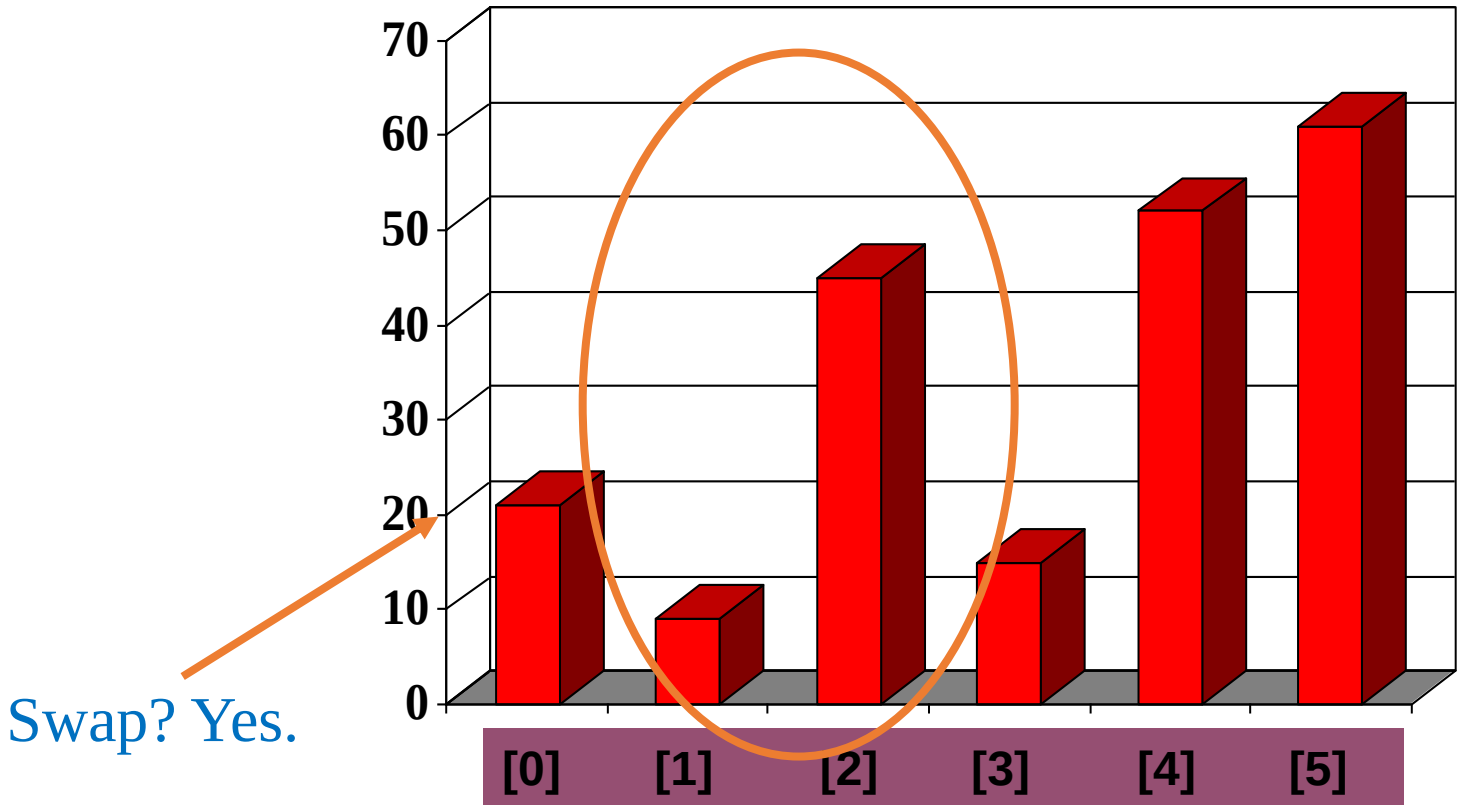
مرتب سازی حبابی

- روی آرایه به تعداد n-1 بار از ابتدا تکرار و در صورت نیاز جابجا نمایید.



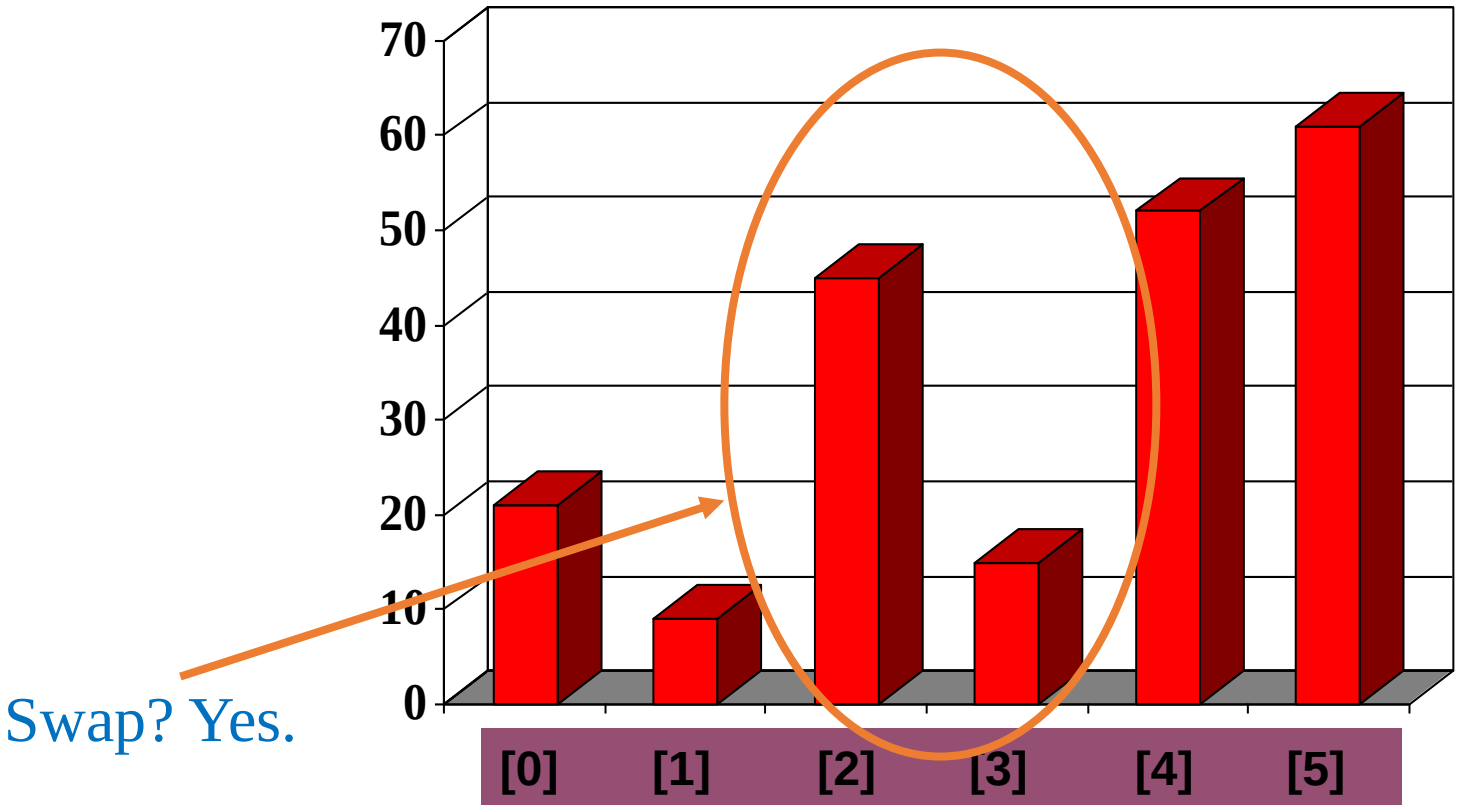
مرتب سازی حبابی

- روی آرایه به تعداد n-1 بار از ابتدا تکرار و در صورت نیاز جابجا نمایید.



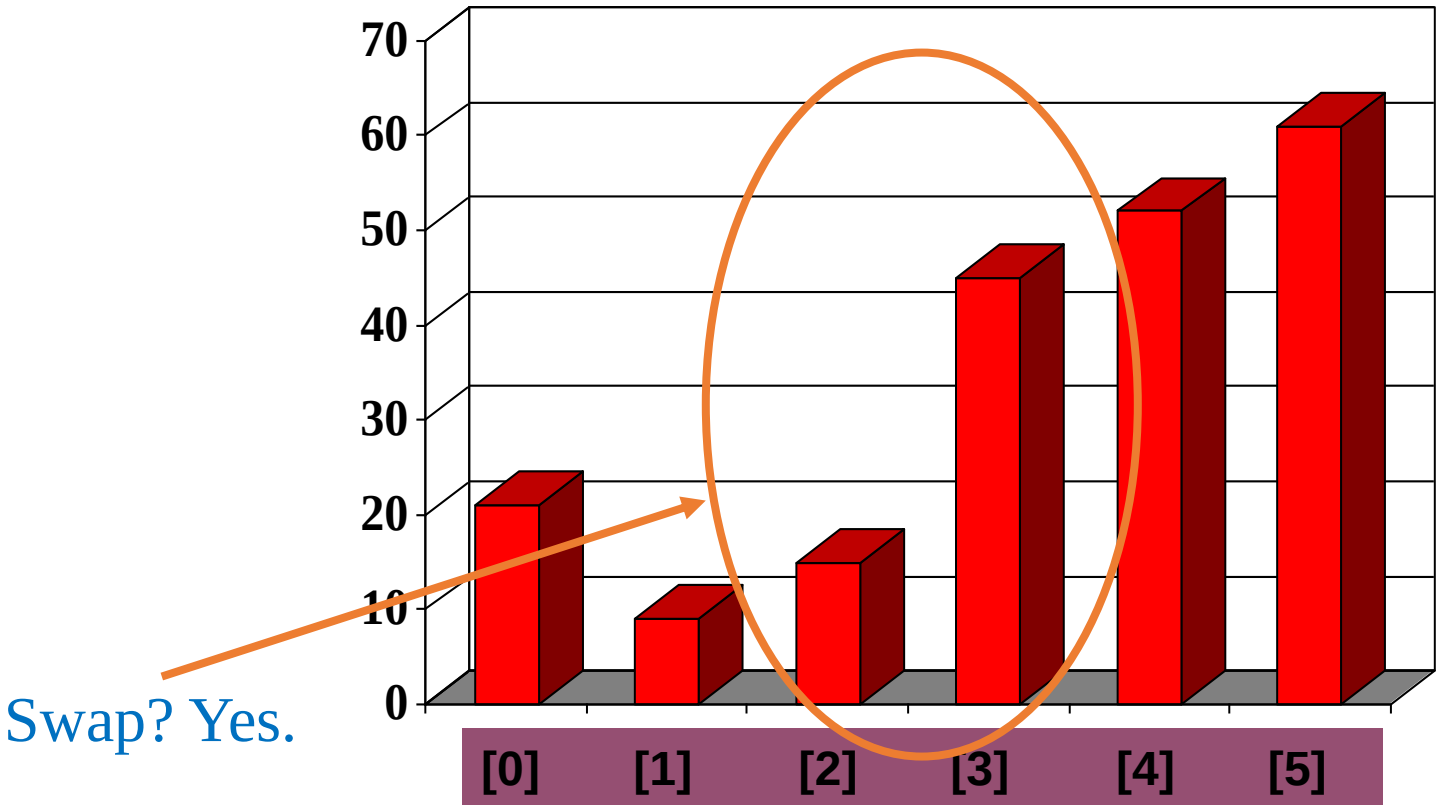
مرتب سازی حبابی

- روی آرایه به تعداد n-1 بار از ابتدا تکرار و در صورت نیاز جابجا نمایید.



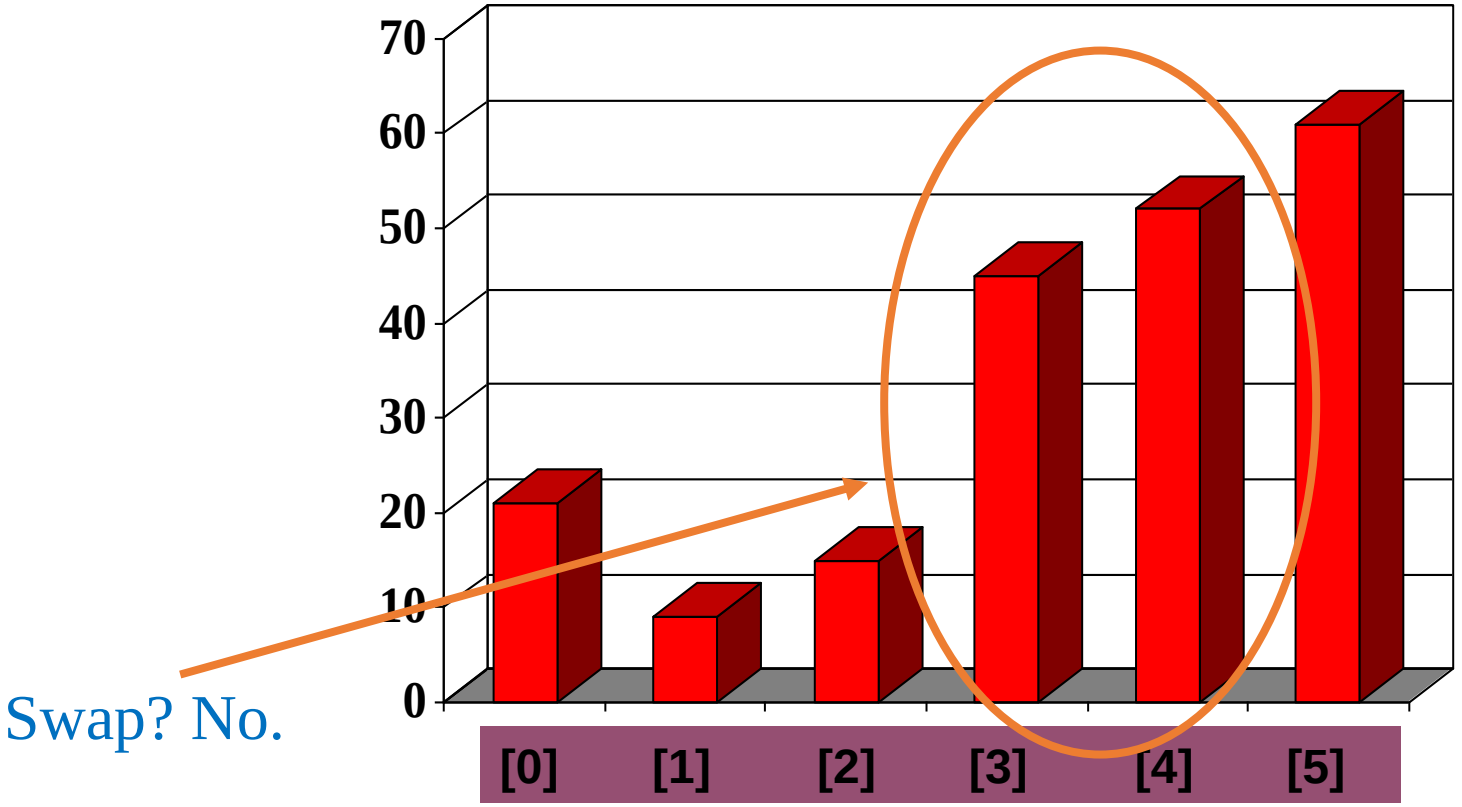
مرتب سازی حبابی

- روی آرایه به تعداد n-1 بار از ابتدا تکرار و در صورت نیاز جابجا نمایید.



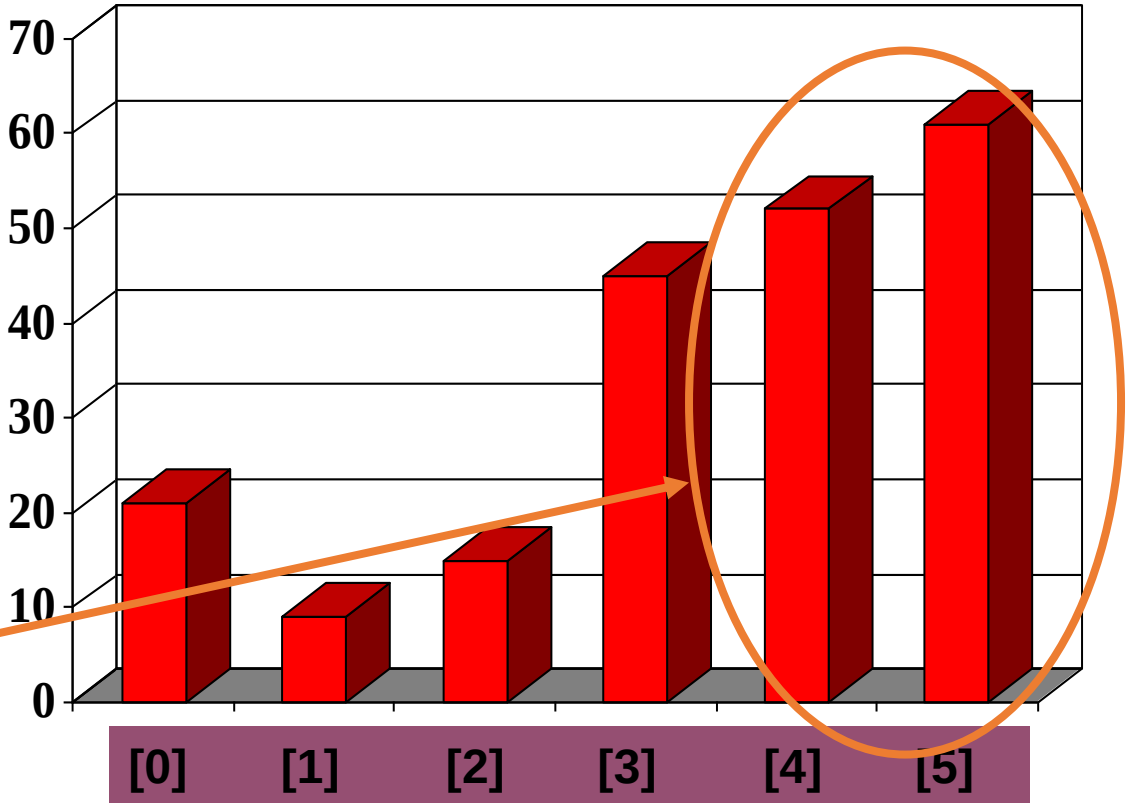
مرتب سازی حبابی

- روی آرایه به تعداد n-1 بار از ابتدا تکرار و در صورت نیاز جابجا نمایید.



مرتب سازی حبابی

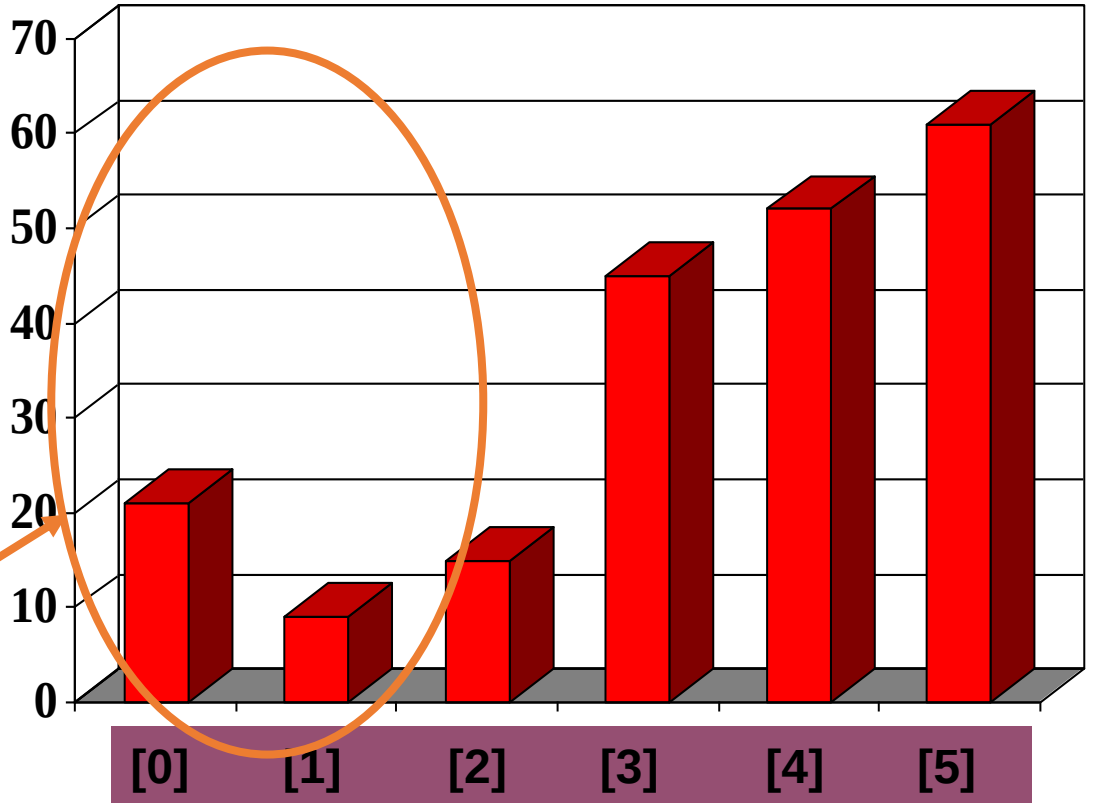
- روی آرایه به تعداد n-1 بار از ابتدا تکرار و در صورت نیاز جابجا نمایید.



Swap? No.

مرتب سازی حبابی

- تکرار کنید تا به اتمام برسد.



Swap? Yes.

الگوریتم مرتب سازی حبابی

```
template <class Item>
void bubble_sort(Item data[ ], size_t n)
{
    size_t i, j;
    Item temp;

    if(n < 2) return;  // nothing to sort!!

    for(i = 0; i < n-1; ++i)
    {
      for(j = 0; j < n-1;++j)
          if(data[j] > data[j+1])      // if out of order, swap!
          {
                  temp = data[j];
                  data[j] = data[j+1];
                  data[j+1] = temp;
           }
     }
}
```

تحلیل زمانی الگوریتم مرتب سازی حبابی

```
template <class Item>
void bubble_sort(Item data[ ], size_t n)
{
    size_t i, j;
    Item temp;

    if(n < 2) return;  // nothing to sort!!

    for(i = 0; i < n-1; ++i)
    {
      for(j = 0; j < n-1;++j)
          if(data[j] > data[j+1])      // if out of order, swap!
          {
                  temp = data[j];
                  data[j] = data[j+1];
                  data[j+1] = temp;
           }
     }
}
```

Outer loop: O(n)

Inner loop: O(n)

Overall : $O(n^2)$

الگوریتم مرتب سازی ادغامی (Merge Sort)

- مرتب‌سازی ادغامی یک الگوریتم «تقسیم و حل» است که در آن ابتدا مسئله به مسائل فرعی تقسیم می‌شود.
- زمانی که راه‌حل‌ها برای مسائل فرعی آماده شد، مجدداً آن‌ها را با هم ترکیب می‌کنیم تا راه‌حل نهایی برای مسئله اصلی به دست آید.

- این یکی از الگوریتم‌هایی است که با استفاده از «بازگشت» به سادگی پیاده‌سازی می‌شود
  - چون به جای مسئله اصلی با مسائل فرعی سر و کار داریم.

الگوریتم مرتب سازی ادغامی (Merge Sort)

- الگوریتم آن را می‌توان به صورت فرایند ۲ مرحله‌ای زیر توصیف کرد:

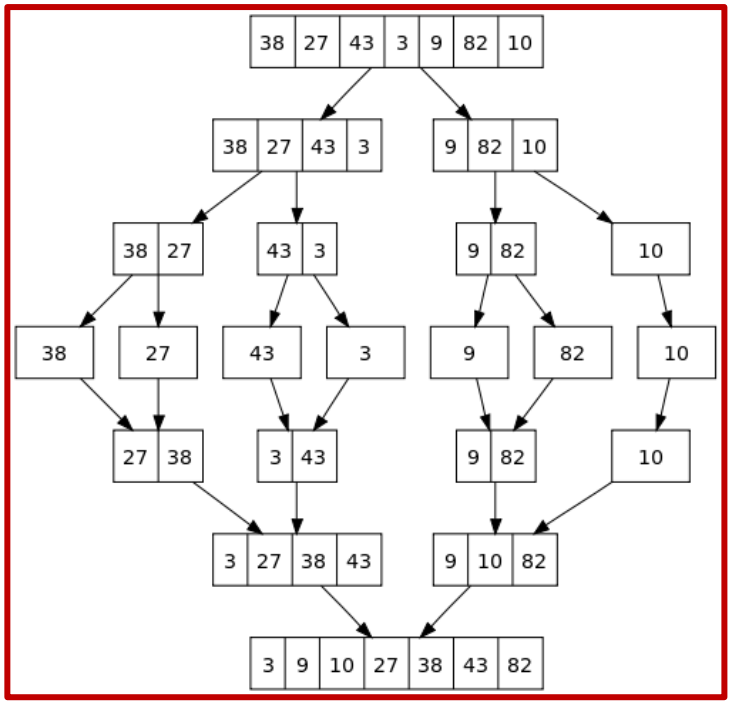
1. تقسیم: در این مرحله آرایه ورودی به دو نیمه تقسیم می‌شود. محور تقسیم نقطه میانی آرایه است. این مرحله به صورت بازگشتی روی همه آرایه‌های نیمه انجام می‌یابد تا این که دیگر نیمه آرایه‌ای برای تقسیم وجود نداشته باشد.
2. حل: در این مرحله باید آرایه‌های تقسیم‌شده را مرتب‌سازی و ادغام کنیم و این کار از بخش زیرین به سمت بالا برای به دست آوردن آرایه مرتب انجام می‌یابد.

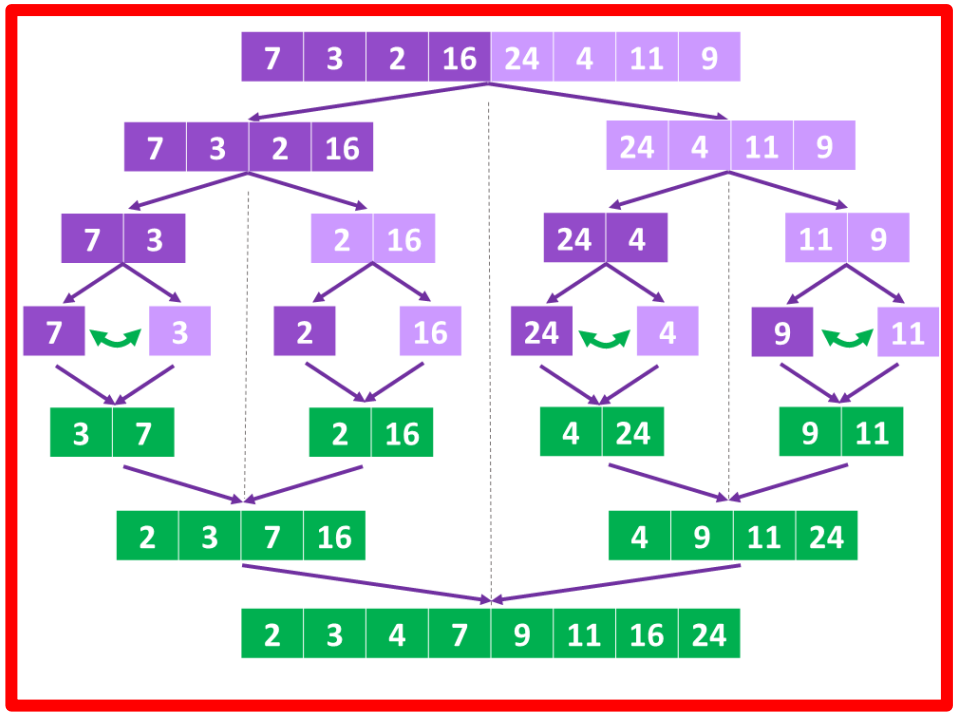
الگوریتم مرتب سازی ادغامی (Merge Sort)

❑ مرتبه زمانی

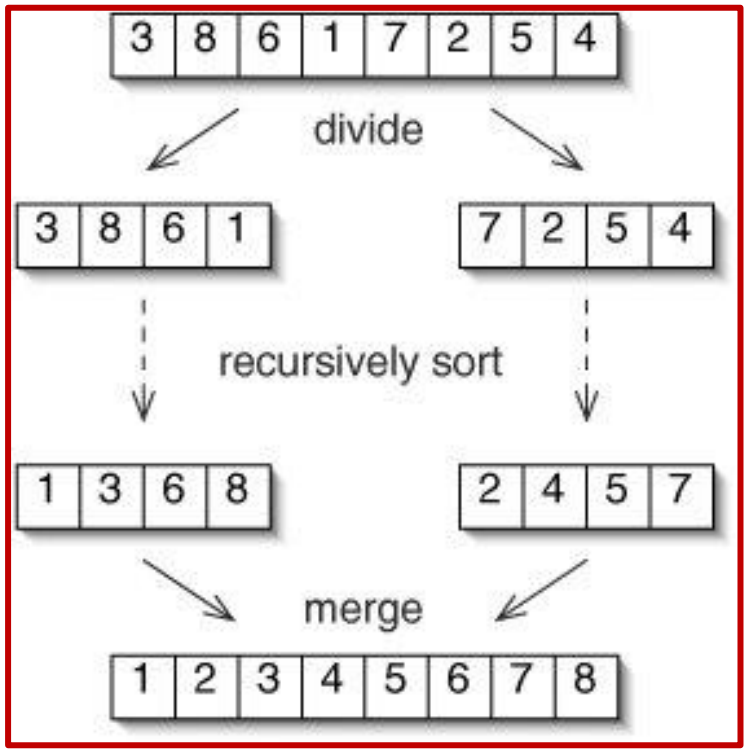
- چون در هر مرحله آرایه به دو نیمه مساوی تقسیم می شود، تعداد مراحل (تقسیم و ترکیب) از مرتبه O(logn) خواهد شد.
- در ترکیب (ادغام دو آرایه مرتب شده) مرتبه زمانی مورد نیاز O(n) می باشد.
- در مجموع مرتبه زمانی این الگوریتم همواره O(nlogn) می باشد.

الگوریتم مرتب سازی ادغامی (Merge Sort)



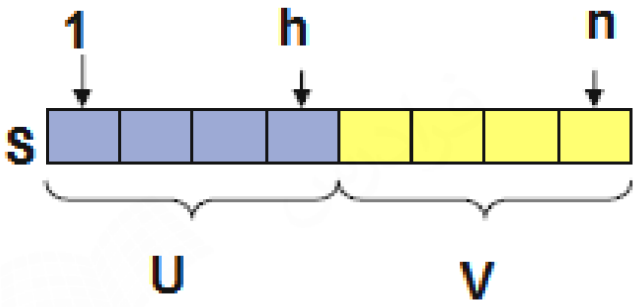


الگوریتم مرتب سازی ادغامی (Merge Sort)



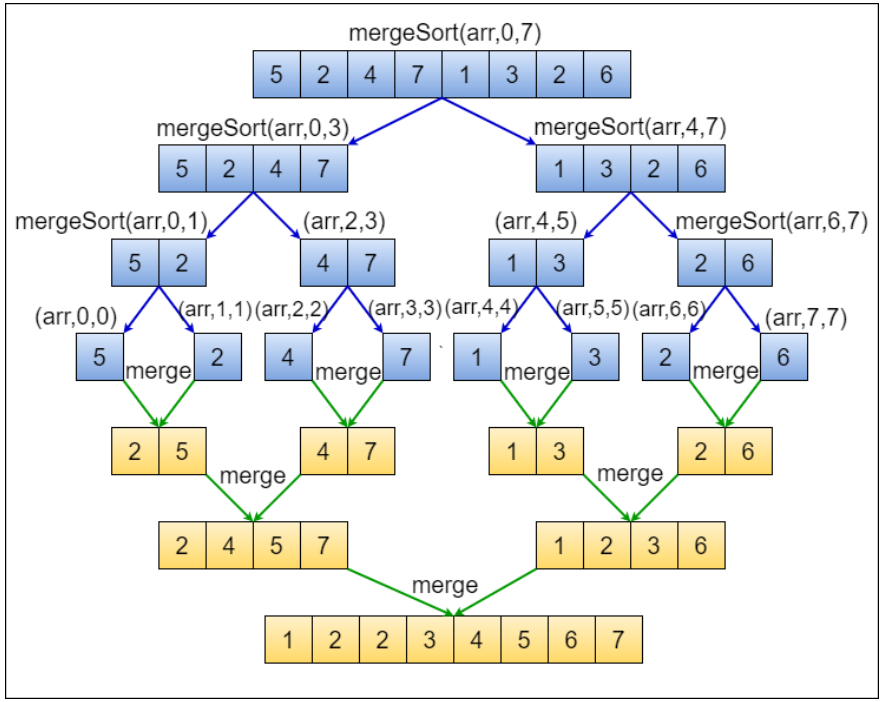
الگوریتم مرتب سازی ادغامی (Merge Sort)

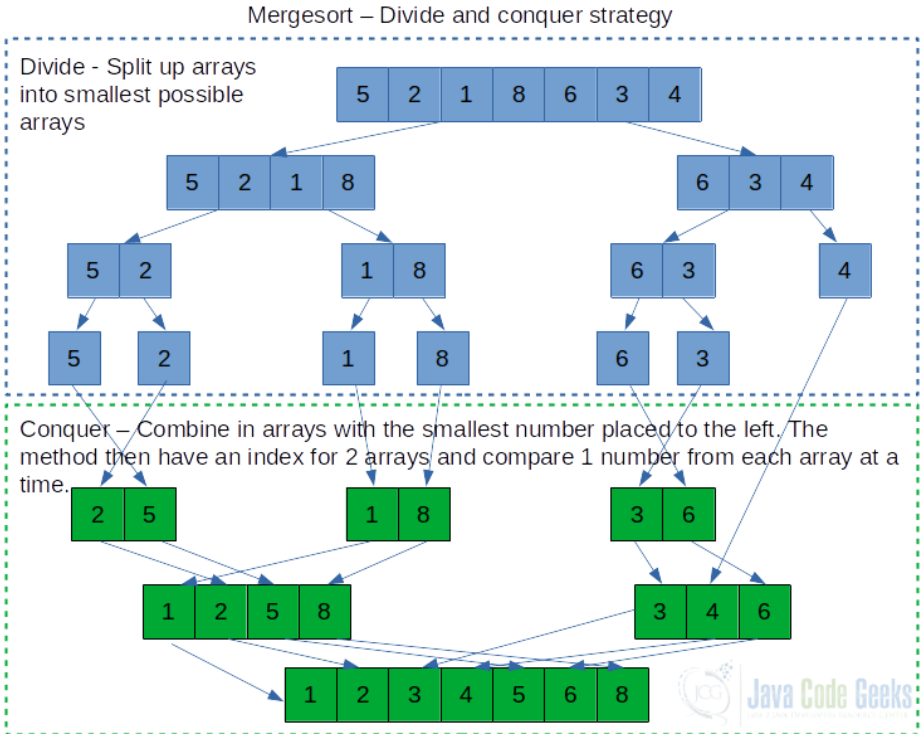
```
mergesort ( S[ ] , n )
{
   h = [n/2];
   m = n – h;
   if (n > 1)
       {
          copy  S[1..h]  to U[1..h];
          copy  S[h +1..n]  to  V[1..m];
          mergesort ( U , h );
          mergesort ( V , m );
           merge (U , h , V , m , S);
       }
}
```



$$T(n) = 2T(\frac{n}{2}) + n - 1$$
$$\Rightarrow T(n) = n\lg n - (n-1)$$
$$\in \theta(n\lg n)$$

الگوریتم مرتب سازی ادغامی (Merge Sort)





الگوریتم مرتب سازی سریع(Quicksort)

- یکی از الگوریتم‌های مرتب‌سازی است که به دلیل مصرف حافظه کم، سرعت اجرای مناسب و پیاده سازی ساده بسیار مورد قبول واقع شده است.
- هر پیاده سازی این الگوریتم به صورت کلی از دو بخش تشکیل شده است.
  - تقسیم بندی آرایه (partition)
  - مرتب کردن

الگوریتم مرتب سازی سریع(Quicksort)

- این الگوریتم طی مراحل بازگشتی زیر یک روش تقسیم و غلبه برای مرتب کردن داده‌ها ارائه می‌نماید:

۱- انتخاب عنصر محوری: یکی از عناصر آرایه به عنوان عنصر محوری (Pivot) به عنوان مثال عنصر اول انتخاب می‌شود.

۲- تقسیم آرایه: چینش عناصر آرایه به قسمی تغییر داده می‌شود که تمامی عناصر کوچکتر یا مساوی محور در سمت چپ آن، و تمامی عناصر بزرگتر در سمت راست آن قرار بگیرند. این دو قسمت زیر آرایه‌های چپ و راست نامیده می‌شوند.

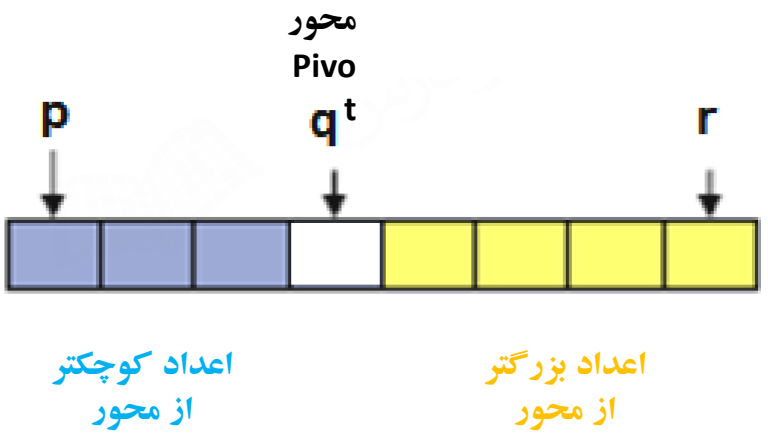
۳- مرتب‌سازی بازگشتی: زیرآرایه‌های چپ و راست به روش مرتب‌سازی سریع مرتب می‌شوند.

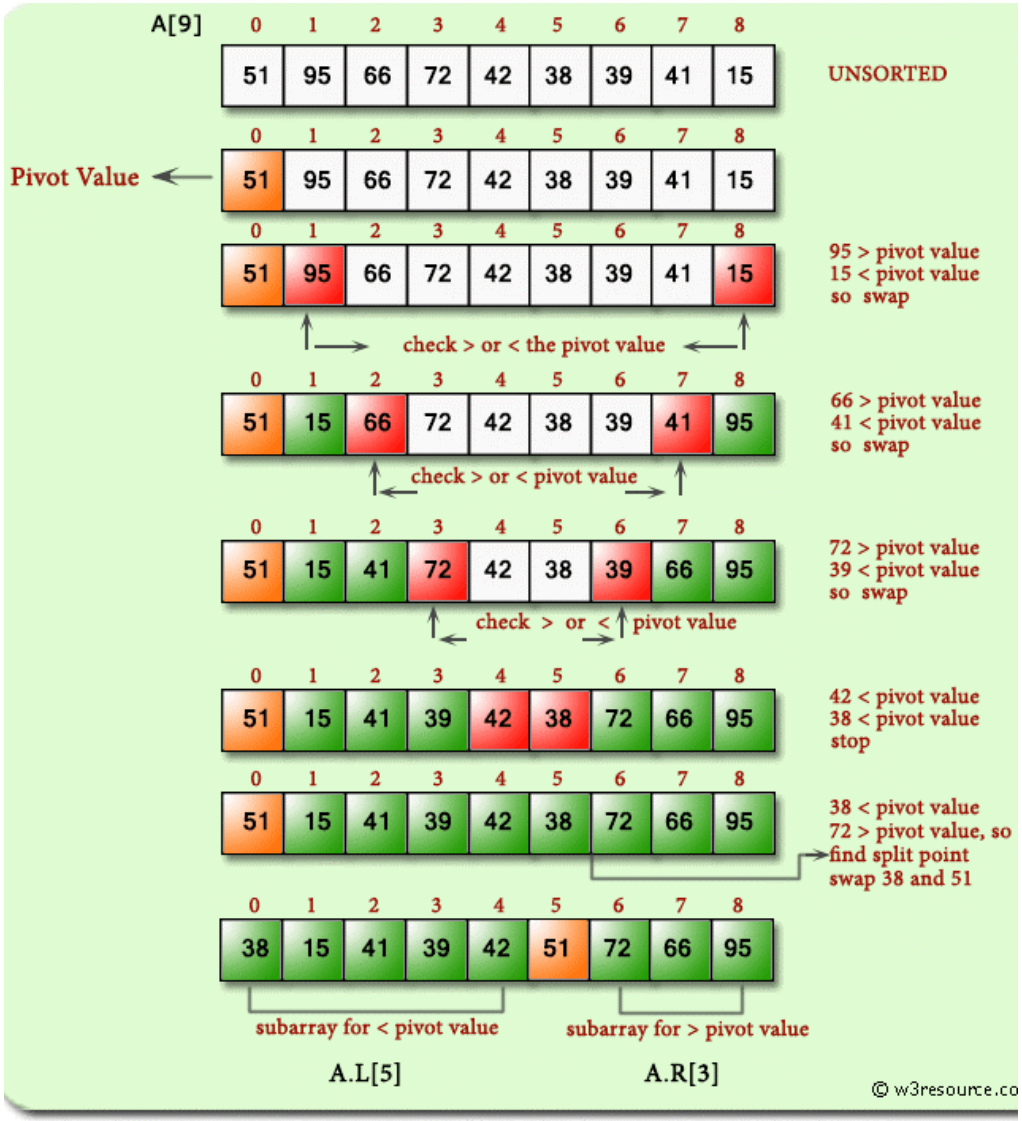
الگوریتم مرتب سازی سریع(Quicksort)

- تابع پارتیشن با دریافت محدوده ای از آرایه، سعی می کند آن محدوده را به دو نیمه تقسیم نماید.

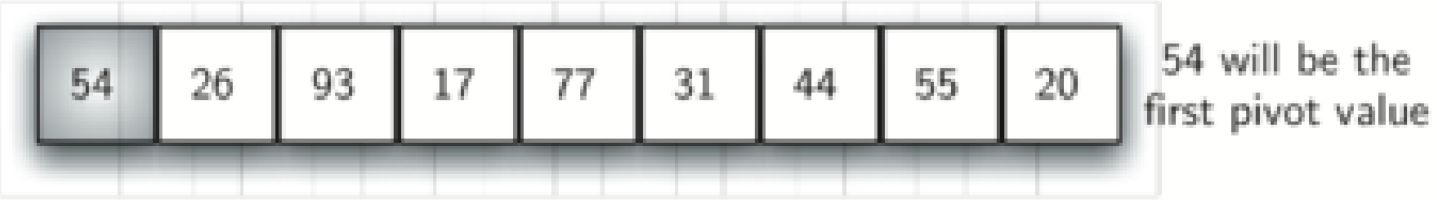
(تمامی اعداد در نیمه سمت چپ کوچکتر از محور و تمامی اعداد سمت راست محور، از آن بزرگتر می باشند.

```
QuickSort (A , p , r)
{
  if ( p < r )
   {
     q = Partition (A , p , r) ;
     QuickSort (A , p , q-1) ;
     QuickSort (A , q+1, r ) ;
   }
}
```



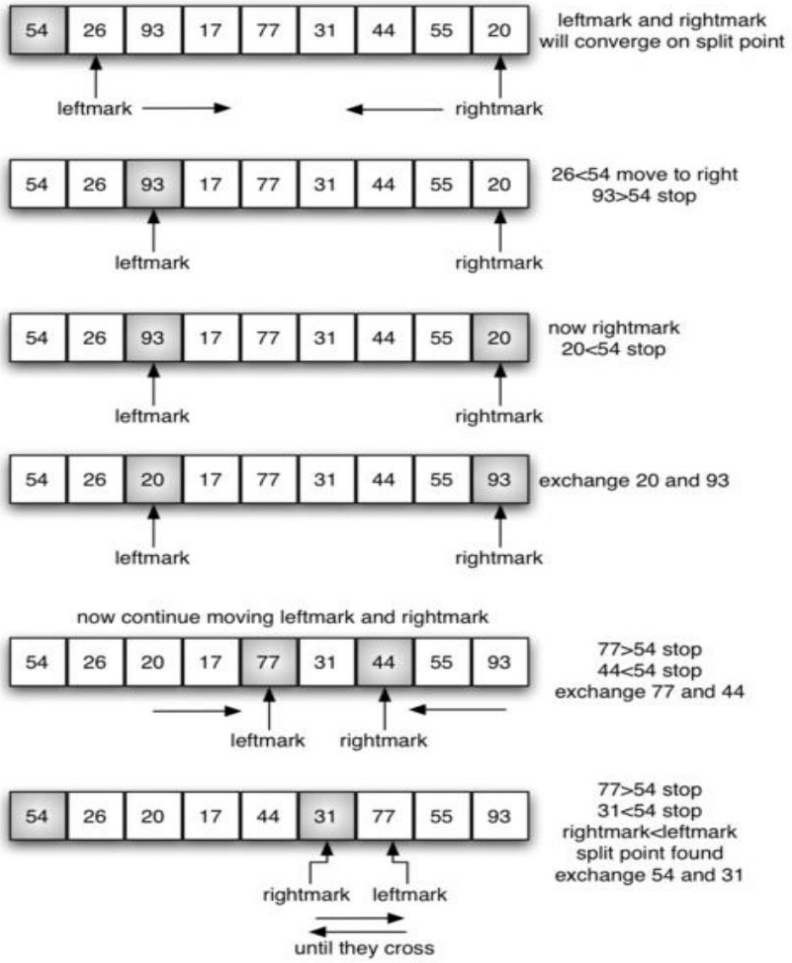


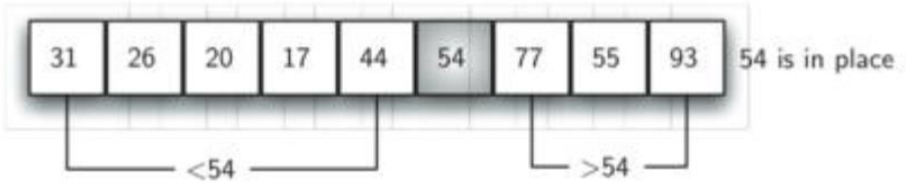
- یکی از داده ها، مثلا عدد اول درون آرایه (در اینجا 51) به عنوان محور انتخاب می شود.
- اشاره گر پایین (از ابتدای آرایه) به سمت بالا (انتهای آرایه) حرکت می کند و اگر عددی را پیدا کند که بزرگتر از مقدار محور است، روی آن توقف می کند. (مثلا در اینجا روی ۹۵)
- اشاره گر بالا (از انتهای آرایه) به سمت پایین (ابتدای آرایه) حرکت می کند و اگر عددی را پیدا کند که کوچکتر از مقدار محور است، روی آن توقف می کند. (روی ۱۵)
- حال مقادیر این دو اشاره را با هم جابجا می نماییم.
- زمانی که این دو اشاره گر از هم رد شدند، آخرین مقدار کوچکتر یا بزرگتر باقیمانده را با خود مقدار محور جابجا می نماییم.

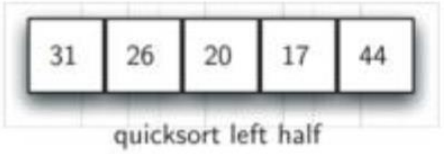


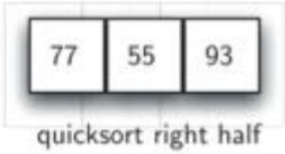
❑ مثالی دیگر از پارتیشن بندی

- محور عدد اول آرایه یعنی ٥٤ انتخاب می شود.
- دو اشاره گر به سمت طرفین آرایه حرکت می کنند و با یافتن دو عدد بزرگتر و کوچکتر از محور، جای آن دو را با هم عوض می کنند





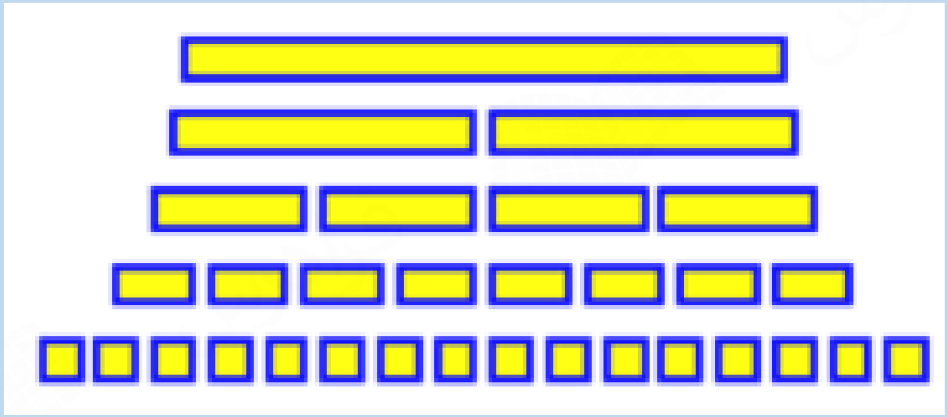




عملکرد مرتب سازی سریع

- در حالتی که در هر مرحله، آرایه به دو نیمه تقریبا مساوی تقسیم شود، تعداد مراحل O(logn) خواهد و چون هر مرحله پارتیشن بندی هم O(n) است در مجموع مرتبه زمانی الگوریتم O(nlogn) خواهد شد.

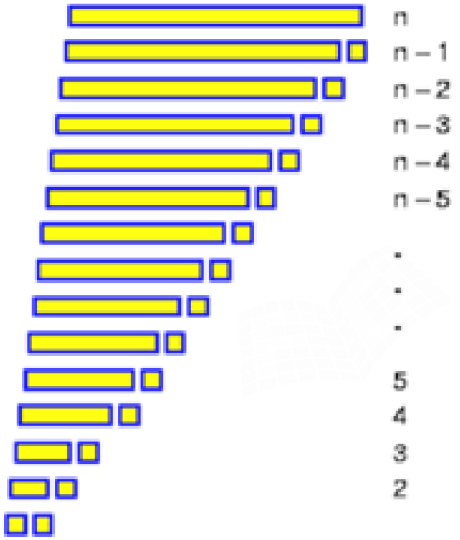
حالت خوب و میانگین :



$$T(n) = T(n/2) + T(n/2)+O(n) \quad \Rightarrow \quad O(nlogn)$$

عملکرد مرتب سازی سریع

در صورتی که اولین عنصر یا آخرین عنصر را به عنوان محور انتخاب کنیم، مرتب سازی سریع برای یک آرایه مرتب، بدترین عملکرد را خواهد داشت.



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

بدترین حالت

در این حالت تعداد مراحل O(n) خواهد شد و در ترکیب با مرتبه زمانی هر مرحله پارتیشن بندی که O(n) است، در مجموع مرتبه زمانی $O(n^2)$ خواهد شد.

الگوریتم پارتیشن بندی

```
Partition(A, s, e, m){
    x ← A[s]; i ← s+1; j ← e;
    do{
        while(A[i] < x) i++;
        while(A[j] > x) j--;
        if(i < j) swap(A[i], A[j]);
    }while(i < j);
    swap(A[s], A[j]);
    return(j);
}
```