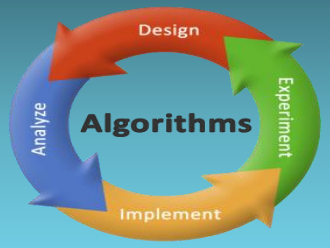
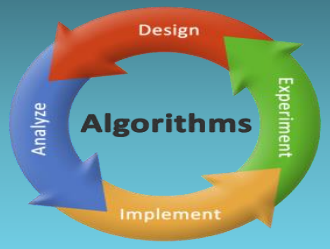
مرتب‌سازی



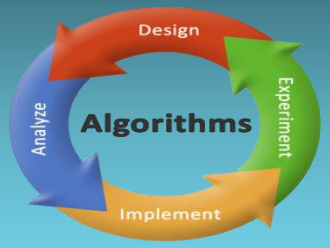
آنچه آموختیم در یک نگاه

Name	Best case	Avg case	Worst case	Stable
Insertion sort	n	n^2	n^2	yes
Merge sort	$n \log n$	$n \log n$	$n \log n$	yes
Heap sort	$n \log n$	$n \log n$	$n \log n$	no
Quick sort	$n \log n$	$n \log n$	n^2	no



مرتب‌سازی مرتبه خطی

- الگوریتم‌های خطی (مرتبه n) :
  - ✓ الگوریتم‌هایی هستند که نیازمند اطلاعاتی در مورد داده ورودی هستند.
  - ✓ بنابراین برای هر نوع داده‌ای قابل استفاده نیستند.
  - ✓ در صورت وجود اطلاعات مذکور، یبا زمان خطی قابل اجرا هستند.
- الگوریتم‌های این دسته عبارتند از:
  - ✓ مرتب سازی شمارشی (Counting Sort)
  - ✓ مرتب‌سازی مبنایی (Radix Sort)
  - ✓ مرتب‌سازی پیمانه‌ای (Bucket Sort)



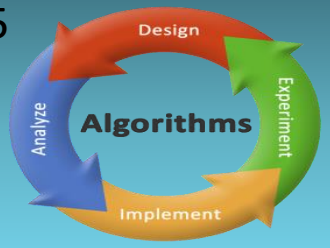
مرتب‌سازی شمارشی (Counting Sort)

این الگوریتم با شمارش تعداد رخداد هر عنصر ممکن و محاسبه‌ی تعداد عناصر کمتر از آن، مکان مناسب عنصر را در خروجی پیدا می کند.

بدیهی است که در این الگوریتم بازه مقادیر عناصر باید متناهی و معلوم باشد؛ برای مثال بین 0 تا k باشد.

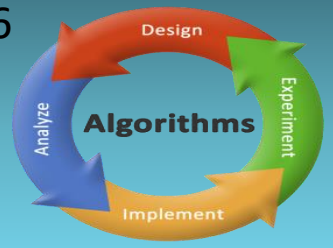
پیچیدگی زمانی این الگوریتم برابر است با:

$$\Theta(k+n) \overset{k=O(n)}{\Longrightarrow} \Theta(n)$$

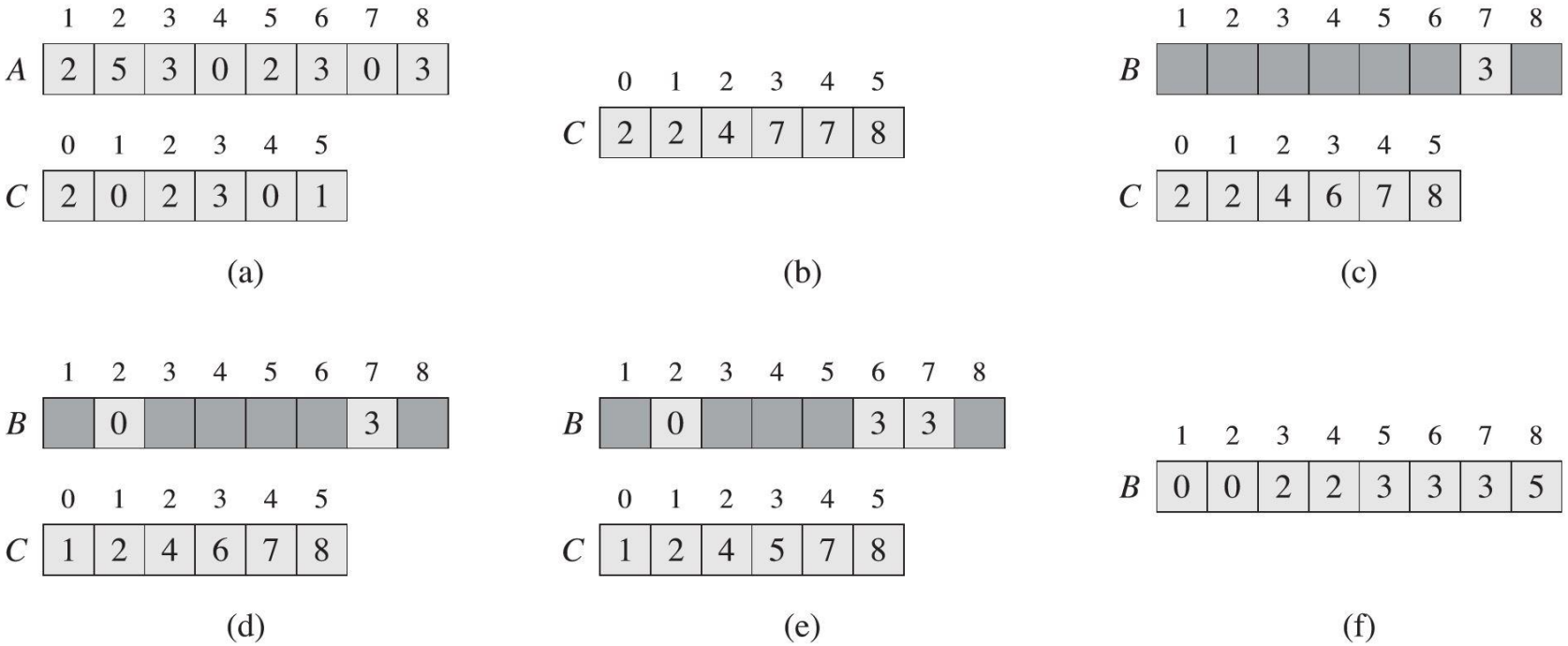


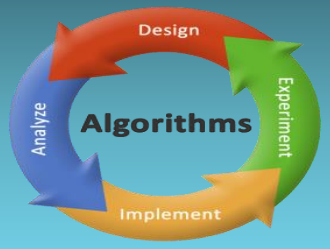
counting sort

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 5      C[A[j]] = C[A[j]] + 1
 6  // C[i] now contains the number of elements equal to i.
 7  for i = 1 to k
 8      C[i] = C[i] + C[i − 1]
 9  // C[i] now contains the number of elements less than or equal to i.
10  for j = A.length downto 1
11      B[C[A[j]]] = A[j]
12      C[A[j]] = C[A[j]] − 1
```



counting sort



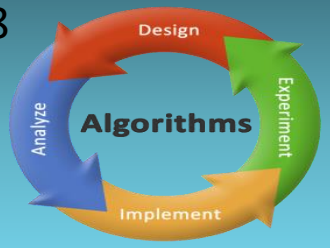


مرتب‌سازی مبنایی (Radix Sort)

این الگوریتم ورودی را به d بخش کوچک می‌شکند و ابتدا بر اساس بخش کم ارزش‌تر و سپس بر اساس بخش پر ارزش‌تر داده‌ها را مرتب می‌کند.

اگر برای مرتب‌سازی هر بخش از الگوریتم مرتب‌سازی شمارشی استفاده کنیم، پیچیدگی زمانی این الگوریتم برابر می‌شود با:

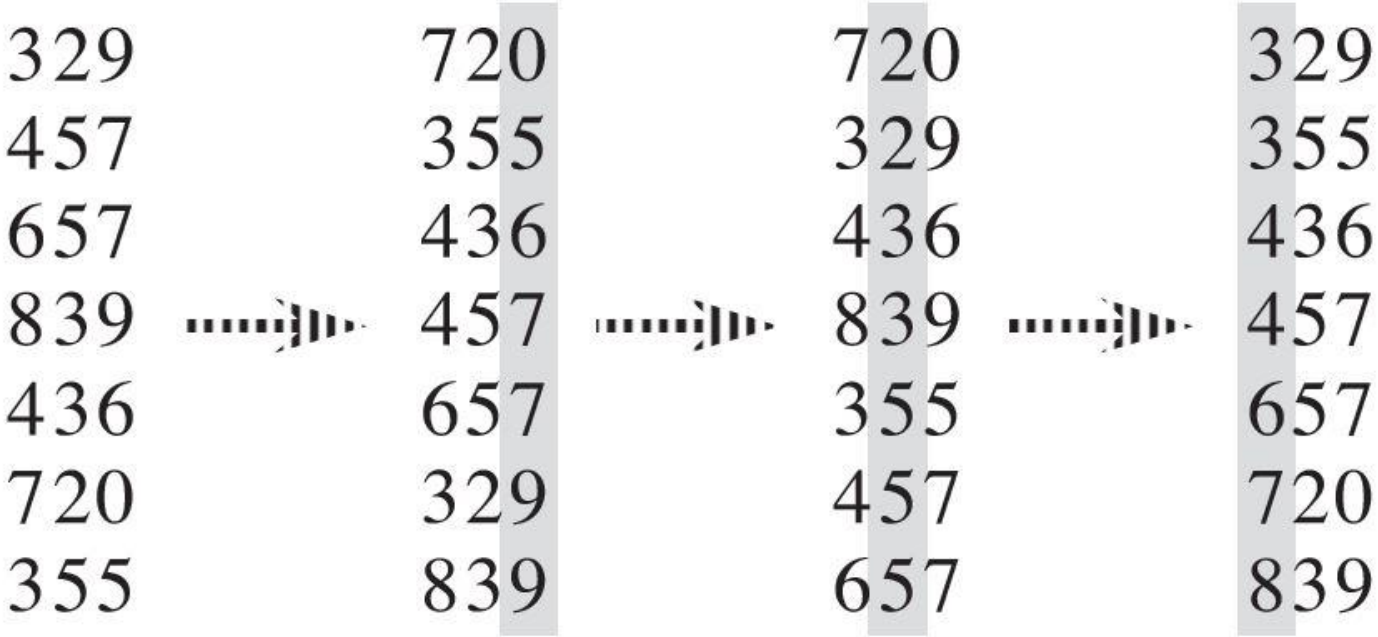
$$\Theta(d(k+n))$$

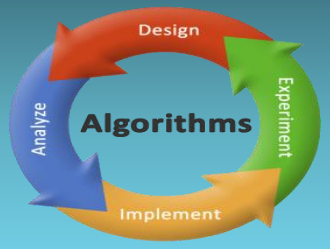


radix sort

RADIX-SORT(A, d)

1 **for** $i = 1$ **to** d
2 use a stable sort to sort array A on digit i





مرتب‌سازی پیمانه‌ای (Bucket Sort)

این مرتب‌سازی با تقسیم کردن داده ها به بخش‌هایی که داده‌های هر بخش از بخشی دیگر کوچک تر است ، مرتب‌سازی هر بخش و چسباندن آن‌ها به هم داده‌ها را مرتب می‌کند.

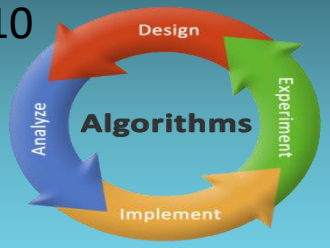
فرضی که در اینجا در مورد داده‌ها داریم این است که:

۱- می‌توان داده‌ها را به بخش‌هایی تقسیم کرد که داده‌های هر بخش از بخشی دیگر کوچک تر باشد.

۲-داده‌ها از پراکندگی مناسبی برخوردار هستند.(یعنی تمام داده‌ها در یک یا دو بخش قرار نگیرند.)

زمان اجرای این الگوریتم برابر است با:

$$O(n * (time\ of\ Insertion\ sort))$$



Bucket sort

```
BUCKET-SORT(A)
1  let B[0..n − 1] be a new array
2  n = A.length
3  for i = 0 to n − 1
4       make B[i] an empty list
5  for i = 1 to n
6       insert A[i] into list B[⌊n A[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i] with insertion sort
9  concatenate the lists B[0], B[1], . . . , B[n − 1] together in order
```