

مقایسه الگوریتم‌های مرتب‌سازی

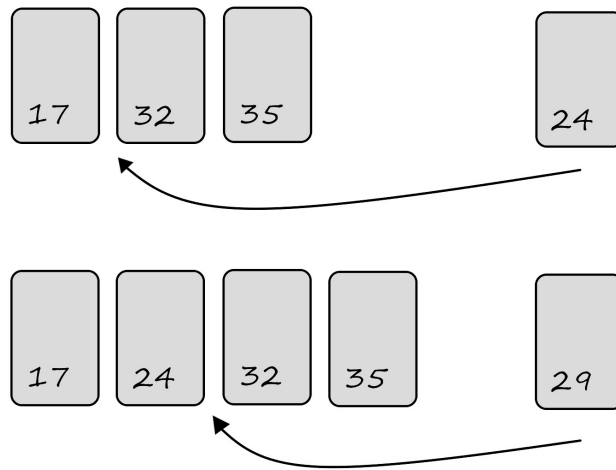
محمد ترابی - علی جعفرآبادی - رضا تاج‌گذاری

تیرماه ۱۴۰۲

۱ مرتب‌سازی درجی^۱

اگر یک دسته کارت به شما داده شود که اعداد ۱ تا ۵۰ روی آن نوشته شده است، چگونه آن را مرتب^۲ می‌کنید؟ احتمالاً اول تعداد کمی کارت برمی‌دارید و آن‌ها را مرتب می‌کنید؛ سپس بقیه کارت‌ها را یکی پس از دیگری نگاه می‌کنید و در جای مناسب میان کارت‌های مرتب شده قرار می‌دهید. شکل ۱ نمایی کلی از این روش مرتب‌سازی نشان می‌دهد.

وقتی کارت‌ها را با این روند مرتب می‌کنیم، همواره تعدادی از کارت‌ها مرتب شده است و کارت‌هایی که هنوز مرتب نشده، یکی پس از دیگری در دسته کارت‌های مرتب شده درج^۳ می‌شوند. اگر با این روش کارت‌ها را مرتب کنیم، درواقع از مرتب‌سازی درجی استفاده کرده‌ایم.



شکل ۱: مراحل از مرتب کردن کارت‌ها به کمک مرتب‌سازی درجی

^۱Insertion sort

^۲sort

^۳insert

الگوریتم مرتب‌سازی درجی

آرایه‌ای به طول یک همواره مرتب است؛ بنابراین از عنصر دوم شروع می‌کنیم و جلو می‌رویم. فرض کنیم که به عنصر i رسیده‌ایم، عناصر قبل از i مرتب‌اند. لذا از آخرین عنصر قبل از i شروع می‌کنیم و به عقب می‌رویم. هر یک از عناصر بزرگتر از i را یک واحد به سمت راست انتقال^۴ می‌دهیم. وقتی به عنصری کوچک‌تر از i یا به ابتدای آرایه رسیدیم متوقف می‌شویم و i را همانجا درج می‌کنیم. هنگامی که عنصر i را درج می‌کنیم، i عنصر اول آرایه مرتب می‌شوند. بنابراین این کار را ادامه می‌دهیم تا تمام عناصر آرایه مرتب شوند. درستی مرتب‌سازی درجی را می‌توان به کمک ثابت‌های حلقه^۵ اثبات کرد. [۱] شبه‌کد مرتب‌سازی درجی را می‌توانید در الگوریتم ۱ که در ادامه آمده است مشاهده کنید^۶.

الگوریتم ۱ مرتب‌سازی درجی

```
1: procedure INSERTIONSORT(arr, n)
2:   for  $i \leftarrow 1$  to  $n - 1$  do
3:      $key \leftarrow arr[i]$ 
4:      $j \leftarrow i - 1$ 
5:     while  $j \geq 0$  and  $arr[j] > key$  do
6:        $arr[j + 1] \leftarrow arr[j]$ 
7:        $j \leftarrow j - 1$ 
8:      $arr[j + 1] \leftarrow key$ 
```

مثال ۱ اگر الگوریتم بالا را روی آرایه^۷ $[۷, ۶, ۶, ۴, ۹]$ اجرا کنیم، آرایه بدین صورت مرتب می‌شود:

$[۷, ۶, ۶, ۴, ۹] \rightarrow [۷, ۷, ۶, ۴, ۹] \rightarrow [۶, ۷, ۶, ۴, ۹] \rightarrow [۶, ۷, ۷, ۴, ۹] \rightarrow$
 $[۶, ۶, ۷, ۴, ۹] \rightarrow [۶, ۶, ۷, ۷, ۹] \rightarrow [۶, ۶, ۶, ۷, ۹] \rightarrow [۶, ۶, ۶, ۷, ۹] \rightarrow$
 $[۴, ۶, ۶, ۷, ۹] \rightarrow [۴, ۶, ۶, ۷, ۹]$

^۴shift

^۵loop invariants

^۶ توجه داشته باشید که الگوریتم بر پایه صفر است؛ یعنی عنصر اول آرایه $arr[۰]$ می‌باشد.

توجه داشته باشید که پیچیدگی زمانی مرتب‌سازی درجی در بهترین حالت خطی است. همچنین بهترین حالت وقتی اتفاق می‌افتد که آرایه مرتب باشد. می‌توانیم نتیجه بگیریم که در مواقعی که آرایه مرتب و یا تقریباً مرتب است مرتب‌سازی درجی بسیار سریع عمل می‌کند. بنابراین اگر از قبل مطلع هستیم که معمولاً داده‌های ما تقریباً مرتب است، استفاده از مرتب‌سازی درجی می‌تواند گزینه مناسبی باشد. به عنوان مثال فرض کنید که یک لیست هزارتایی از اسامی دانشجویان یک موسسه در اختیار دارید که به ترتیب حروف الفبا مرتب شده اند. در سال جدید پنجاه دانشجو در موسسه نام نویسی می‌کنند و اسامی آن‌ها به آخر آرایه اضافه می‌شود. اگرچه روش‌های زیادی برای مرتب کردن لیست جدید دانشجویان وجود دارد، مرتب‌سازی درجی گزینه مناسبی محسوب می‌شود و از دیگر روش‌های مرتب‌سازی که در این مقاله توضیح داده شده، بهتر عمل می‌کند.

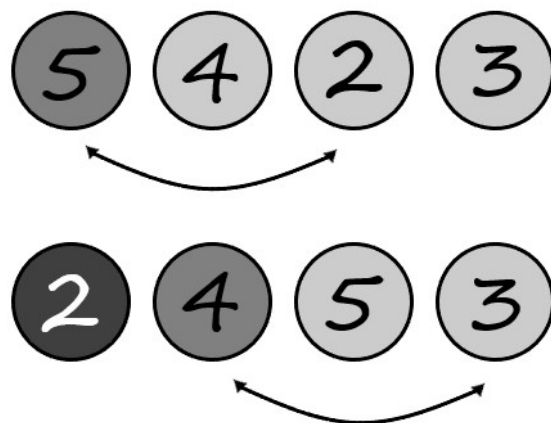
خوب است بدانید می‌توان پیچیدگی زمانی مرتب‌سازی درجی در بدترین حالت را با تغییراتی در الگوریتم آن بهبود بخشید. به عنوان مثال مرتب‌سازی شل^۷ یک روش مرتب‌سازی دیگر است که از مرتب‌سازی درجی استفاده می‌کند و در حالت کلی دارای پیچیدگی زمانی بهتری است. [۲] [۳] جزئیات الگوریتم و پیچیدگی زمانی مرتب‌سازی شل را در این مقاله بررسی نمی‌کنیم.

۲ مرتب‌سازی انتخابی^۸

مرتب‌سازی انتخابی الگوریتم ساده‌ای دارد و احتمالاً یکی از اولین الگوریتم‌های مرتب‌سازی باشد که به ذهنمان می‌رسد. مرتب‌سازی انتخابی اینگونه عمل می‌کند که کوچک‌ترین عنصر آرایه را انتخاب می‌کند، آن را در سمت چپ آرایه قرار می‌دهد و سپس به سراغ عناصر باقی‌مانده می‌رود. [۱] برای درک بهتر می‌توانید به شکل ۲ که در ادامه آمده است نگاه کنید.

^۷Shell sort

^۸Selection sort



شکل ۲: مراحل از مرتب کردن اعداد به کمک مرتب‌سازی انتخابی

الگوریتم مرتب‌سازی انتخابی

از اولین عنصر آرایه شروع می‌کنیم. اگر به عنصر i برسیم، i را به عنوان اندیس^۹ کوچک‌ترین عنصر ذخیره می‌کنیم؛ سپس از عنصر بعد از i شروع می‌کنیم و جلو می‌رویم. اگر عنصری از کوچک‌ترین عنصر کوچک‌تر بود اندیس آن را به عنوان اندیس کوچک‌ترین عنصر نگه می‌داریم. وقتی به آخر آرایه رسیدیم، کوچک‌ترین عنصر ذخیره شده را با i جابه‌جا^{۱۰} می‌کنیم و به سراغ عنصر بعدی می‌رویم. مراحل گفته شده را برای i جدید تکرار می‌کنیم تا به انتهای آرایه برسیم. شبه‌کد مرتب‌سازی انتخابی در الگوریتم ۲ که در ادامه آمده است قابل مشاهده است. همچنین شبه‌کد جابه‌جایی را می‌توانید در الگوریتم ۳ مشاهده کنید.

^۹index

^{۱۰}swap

الگوریتم ۲ مرتب‌سازی انتخابی

```
1: procedure SELECTIONSORT(arr, n)
2:   for  $i \leftarrow 0$  to  $n - 1$  do
3:      $minIndex \leftarrow i$ 
4:     for  $j \leftarrow i + 1$  to  $n$  do
5:       if  $arr[j] < arr[minIndex]$  then
6:          $minIndex \leftarrow j$ 
7:     SWAP(arr, i, minIndex)
```

الگوریتم ۳ جابه‌جایی

```
1: procedure SWAP(arr, i, j)
2:    $temp \leftarrow arr[i]$ 
3:    $arr[i] \leftarrow arr[j]$ 
4:    $arr[j] \leftarrow temp$ 
```

مثال ۲ اگر الگوریتم ۲ را روی آرایه $[5, 4, 2, 3, 7]$ اجرا کنیم، آرایه بدین صورت مرتب می‌شود:

$[5, 4, 2, 3, 7] \rightarrow [2, 4, 5, 3, 7] \rightarrow [2, 3, 5, 4, 7] \rightarrow [2, 3, 4, 5, 7] \rightarrow [2, 3, 4, 5, 7]$

مثال ۳ اگر الگوریتم ۲ را روی آرایه $[7, 6, 6, 4, 9]$ اجرا کنیم، آرایه بدین صورت مرتب می‌شود:

$[7, 6, 6, 4, 9] \rightarrow [4, 6, 6, 7, 9] \rightarrow [4, 6, 6, 7, 9] \rightarrow [4, 6, 6, 7, 9] \rightarrow [4, 6, 6, 7, 9]$

نکته ۱ توجه کنید در مثال ۳ با اینکه آرایه بعد از یک جابه‌جایی مرتب می‌شود، الگوریتم به کار خود ادامه می‌دهد و به‌ازای تمام عناصر i تا $n - 1$ اجرا می‌شود. با مقایسه مثال ۲ و مثال ۳ که در بالا آمده‌اند، می‌توان وابسته نبودن مرتب‌سازی انتخابی به عناصر ورودی را بهتر درک کرد.

۳ مرتب‌سازی ادغامی^{۱۱}

مرتب‌سازی ادغامی یکی دیگر از روش‌های مرتب‌سازی است که از روش تقسیم و حل^{۱۲} استفاده می‌کند. این روش در سال ۱۹۴۵ میلادی ابداع شد. [۴] استفاده از این روش نسبت به روش‌های قبلی سرعت مرتب‌شدن اعداد را به طور قابل توجهی بالا می‌برد.

این روش اینگونه عمل می‌کند که آرایه را به دو قسمت تقسیم می‌کند و هر قسمت را به صورت بازگشتی مرتب می‌کند. وقتی که دو قسمت آرایه مرتب شدند، در واقع دو آرایه مرتب شده داریم و می‌خواهیم آن دو را باهم ادغام^{۱۳} کنیم. ادغام کردن دو آرایه مرتب شده و تبدیل آن به یک آرایه مرتب شده، در زمان خطی^{۱۴} قابل انجام است. خطی بودن ادغام یکی از دلایل سریع شدن مرتب‌سازی درجی است.

الگوریتم مرتب‌سازی ادغامی

آرایه اولیه را به دو قسمت، که اندازه هر قسمت حدود نصف آرایه اولیه است، تقسیم می‌کنیم. این عمل را به طور بازگشتی ادامه می‌دهیم تا به آرایه‌هایی به یک عضو برسیم. آرایه با یک عضو همواره مرتب است. سپس آرایه‌های کوچک‌تر مرتب شده را با یکدیگر ادغام می‌کنیم تا آرایه‌های بزرگ‌تر مرتب شده داشته باشیم. این عمل را روی کل آرایه انجام می‌دهیم و کل آرایه مرتب می‌شود. شبه‌کد مرتب‌سازی ادغامی و شبه‌کد ادغام^{۱۵}، که در مرتب‌سازی ادغامی استفاده می‌شود، به ترتیب در الگوریتم ۴ و الگوریتم ۶ در ادامه آمده است.

¹¹Merge sort

¹²divide and conquer

¹³merge

¹⁴linear

¹⁵merge

الگوریتم ۴ مرتب‌سازی ادغامی

```
1: procedure MERGESORT( $arr, left, right$ )  
2:   if  $left < right$  then  
3:      $middle \leftarrow \frac{left+right}{2}$   
4:     MERGESORT( $arr, left, middle$ )  
5:     MERGESORT( $arr, middle + 1, right$ )  
6:     MERGE( $arr, left, middle, right$ )
```

```

1: procedure MERGE(arr, left, middle, right)
2:    $n1 \leftarrow middle - left + 1$ 
3:    $n2 \leftarrow right - middle$ 
4:   leftArr  $\leftarrow$  new Array[n1]
5:   rightArr  $\leftarrow$  new Array[n2]
6:   for  $i \leftarrow 0$  to  $n1$  do
7:     leftArr[ $i$ ]  $\leftarrow arr[left + i]$ 
8:   for  $j \leftarrow 0$  to  $n2$  do
9:     rightArr[ $j$ ]  $\leftarrow arr[middle + 1 + j]$ 
10:   $i \leftarrow 0$ 
11:   $j \leftarrow 0$ 
12:   $k \leftarrow left$ 
13:  while  $i < n1$  and  $j < n2$  do
14:    if leftArr[ $i$ ]  $\leq$  rightArr[ $j$ ] then
15:      arr[ $k$ ]  $\leftarrow leftArr[i]$ 
16:       $i \leftarrow i + 1$ 
17:    else
18:      arr[ $k$ ]  $\leftarrow rightArr[j]$ 
19:       $j \leftarrow j + 1$ 
20:     $k \leftarrow k + 1$ 
21:  while  $i < n1$  do
22:    arr[ $k$ ]  $\leftarrow leftArr[i]$ 
23:     $i \leftarrow i + 1$ 
24:     $k \leftarrow k + 1$ 
25:  while  $j < n2$  do
26:    arr[ $k$ ]  $\leftarrow rightArr[j]$ 
27:     $j \leftarrow j + 1$ 
28:     $k \leftarrow k + 1$ 

```

مثال ۴ اگر الگوریتم ۴ را روی آرایه $[۷, ۶, ۳, ۱]$ اجرا کنیم، آرایه بدین صورت مرتب می‌شود:

$[۷, ۶, ۳, ۱] \rightarrow [۷, ۶], [۳, ۱] \rightarrow [۷], [۶], [۳, ۱] \rightarrow [۶, ۷], [۳, ۱] \rightarrow$
 $[۶, ۷], [۳], [۱] \rightarrow [۶, ۷], [۱, ۳] \rightarrow [۱, ۳, ۶, ۷]$

نکته ۲ مرتب‌سازی ادغامی نسبت به مرتب‌سازی‌های دیگری که در این مقاله بحث شده به حافظه بیشتری نیاز دارد. البته قابل توجه است که اقداماتی برای تغییر این الگوریتم با این هدف که پیچیدگی حافظه آن کم شود انجام شده. چندین روش برای مرتب‌سازی ادغامی با حافظه ثابت^{۱۶} یا به عبارتی مرتب‌سازی ادغامی درجا^{۱۷} پیشنهاد شده است. [۵]

۴ مرتب‌سازی سریع^{۱۸}

مرتب‌سازی سریع یکی از روش‌های پر استفاده برای مرتب‌سازی است. این روش نه تنها سرعت بالایی دارد، بلکه از نظر حافظه نیز خوب عمل می‌کند و حافظه نسبتاً کمی استفاده می‌کند. یکی دیگر از مزایای مرتب‌سازی سریع این است که الگوریتم نسبتاً ساده‌ای دارد و پیاده‌سازی آن دشوار نیست. این الگوریتم توسط تونی هور^{۱۹} در سال ۱۹۵۹ میلادی ابداع شد. [۶] این روش به طور کلی، در مرتب کردن داده‌های تصادفی، از مرتب‌سازی ادغامی سریع‌تر عمل می‌کند؛ مخصوصاً در داده‌های بزرگ‌تر. [۷]

مرتب‌سازی سریع نیز همانند مرتب‌سازی ادغامی از روش تقسیم و حل^{۲۰} استفاده می‌کند. ابتدا یک عنصر محوری^{۲۱} انتخاب می‌کند؛ سپس عناصر کوچک‌تر را سمت چپ و عناصر بزرگ‌تر را در سمت راست آن قرار می‌دهد. به همین دلیل این روش، مرتب‌سازی بخش‌کردن-تبادل^{۲۲} نیز نامیده می‌شود. [۸]

¹⁶constant

¹⁷in-place merge sort

¹⁸Quicksort

¹⁹Tony Hoare

²⁰divide and conquer

²¹pivot

²²partition-exchange sort

الگوریتم مرتب‌سازی سریع

الگوریتم ۶ مرتب‌سازی سریع

```
1: procedure QUICKSORT( $arr, low, high$ )
2:   if  $low < high$  then
3:      $pivot \leftarrow$  PARTITION( $arr, low, high$ )
4:     QUICKSORT( $arr, low, pivot - 1$ )
5:     QUICKSORT( $arr, pivot + 1, high$ )
```

الگوریتم ۷ بخش کردن

```
1: procedure PARTITION( $arr, low, high$ )
2:    $pivot \leftarrow arr[high]$ 
3:    $i \leftarrow low - 1$ 
4:   for  $j \leftarrow low$  to  $high - 1$  do
5:     if  $arr[j] \leq pivot$  then
6:       Swap  $arr[i + 1]$  and  $arr[j]$ 
7:        $i \leftarrow i + 1$ 
8:   Swap  $arr[i + 1]$  and  $arr[high]$ 
9:   return  $i + 1$ 
```

مراجع

- [1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 4th ed. , 2022.
- [2] R. B. Frank, R. M.; Lazarus, “A high-speed sorting procedure,” *Communications of the ACM*, p.20–22, 1960.
- [3] R. Sedgewick, “A new upper bound for shellsort,” *Journal of Algorithms*, p.159–173, 1986.
- [4] D. Knuth. *The Art of Computer Programming*. Addison-Wesley, 2nd ed. .
- [5] M. A. Huang, Bing-Chao; Langston, “Practical in-place merging,” *Communications of the ACM*, p.348–352, 1988.

- [6] C. A. R. Hoare, “Algorithm 64: Quicksort,” *Comm. ACM*, p.321, 1961.
- [7] S. S. Skiena, “The algorithm design manual,” *Springer*, p.129, 2008.
- [8] C. Foster, “Algorithms, abstraction and implementation,” p.98, 1992.