مرتب سازی

یک الگوریتم مرتب سازی الگوریتمی است که عناصر یک لیست را در ترتیب معینی قرار می دهد. کارائی مرتب سازی برای بهینه سازی کاربردهای الگوریتم های دیگر مانند جستجو و ادغام اهمیت دارد. که لیست های مرتبی را برای کار به طرز صحیح نیاز دارند. مرتب سازی همچنین برای تهیه یک خروجی خوانا تر برای انسان نیز مفید است.

مسئله مرتب سازی میزان زیادی از تحقیقات را به خود جذب کرده است. شاید به دلیل این است که علی رغم سادگی پیچیدگی حل آن به صورت کارآمد است

مرتب سازی حبابی
مرتب سازی انتخابی
مقایسه الگوریتم های مرتب سازی

الگوریتم های مرتب سازی اغلب بر اساس زیر دسته بندی می شوند:

- پیچیدگی زمانی مقایسه عناصر برحسب اندازه لیست (n) . معمولا برای یک الگوریتم مرتب سازی عادی O(n log n) بهترین حالت و $O(n^2)$ بدترین حالت است. زمان ایده آل O(n) است.
- پیچیدگی زمانی تعداد جابه جائی ها برای الگوریتم های in place.
- مصرف حافظه (و استفاده از منابع دیگر سیستم). خصوصا برخی از الگوریتم های مرتب سازی in place هستند. که تنها به O(n) یا O(log n) حافظه علاوه بر داده های در حال مرتب شدن نیاز دارند. درحالیکه بقیه احتیاج به محل کمکی برای نگهداری داده های موقت دارند.
- بعضی از الگوریتم ها بازگشتی یا غیر بازگشتی یا هردو هستند.
- تعادل. الگوریتم های مرتب سازی متعادل ترتیب نسبی رکوردها با کلیدهای مساوی را برقرار می کنند.
- آیا مرتب سازی مقایسه ای هستند یا خیر. یک مرتب سازی مقایسه ای (comparison sort) داده را تنها با مقایسه دو عنصر یا یک عملگر مقایسه بررسی می کند.
- متد کلی. درج،تعویض، انتخاب، ادغام و غیره. مرتب سازی حبابی و سریع مرتب سازی تعویض و heapsort مرتب سازی انتخابی هستند.
- تعادل. الگوریتم های مرتب سازی متعادل ترتیب نسبی رکوردها با کلیدهای مساوی را برقرار می کنند. یعنی اگر دو رکورد R و S با یک کلید وجود داشته باشد و R قبل از S در لیست اصلی آمده باشد، در لیست مرتب شده هم R قبل از S می آید.

مرتب سازی حبابی

مرتب سازی حبابی (Bubble sort) ساده ترین روش مرتب کردن داده ها می باشد.

قدم هائی را تکرار می کند تا داده های لیست مرتب شوند. در هر قدم دو عنصر با هم مقایسه می شوند و اگر ترتیب آنها درست نباشد با هم جابه جا می شوند. قدم ها تا زمانی که کل لیست مرتب شود و جا به جائی موردنیاز نباشد تکرار می شود.

الگوریتم از ابتدای لیست شروع می کند. دو عنصر اول را مقایسه می کند، اگر اولی از دومی بزرگتر بود جای آنها عوض می شود. به همین ترتیب ادامه می دهد تا به انتهای لیست برسد. الگوریتم مجددا همین عمل را از ابتدای لیست تکرار می کند تا زمانی هیچ جا به جائی در آخرین گام صورت نگیرد.

عناصر کوچکتر به سمت بالای لیست حرکت می کنند به همین دلیل "حبابی" نامیده شده است.

با وجود سادگی این الگوریتم بسیار ناکارآمد است و به جز آموزش به ندرت در جاهای دیگر استفاده می شود.

پیاده سازی

شبه کد الگوریتم مرتب سازی حبابی به صورت زیر است. آرایه A لیستی با n عنصر که اندیس آن از یک شروع می شود.

```
for (i:= 1 to n-1 )
  for (j:= n downto i + 1)
    if A[j-1] > A[j] then
      swap(A[j-1],  A[j])
    end if
  end for
end for
```

یک راه برای بهتر کردن الگوریتم فوق توجه به این مطلب است که در هر مرحله بزرگترین عنصر به انتهای لیست منتقل می شود. بنابراین برای یک لیست با nعنصر در مرحله بعد نیاز به بررسی n-1 عنصر دیگر می باشد. یعنی هر بار یک عنصر در محل خود قرار می گیرد که در گام بعدی می توان آن را در نظر نگرفت.

با توجه به اینکه ممکن است در مرحله ای کلیه عناصر در محل خود قرار بگیرند اگر جا به جائی در یک گام وجود نداشته باشد به معنی مرتب بودن لیست و خاتمه الگوریتم است.

شبه کد زیر الگوریتم بهینه شده مرتب سازی حبابی است.

```
do
  swapped := false
  n := n - 1
  for (i:= 1 to n)
    if A[i] > A[i + 1] then
      swap( A[i], A[i+1] )
      swapped := true
    end if
  end for
while swapped
```

مثال. آرایه ای با عناصر "8 2 4 1 5" را در نظر بگیرید. گام های لازم برای مرتب سازی لیست به صورت صعودی به صورت زیر است. در هر مرحله عماصری که مقایسه می شوند پررنگ تر نشان داده شده اند.

First Pass:
(**5 1** 4 2 8) ⟶ (**1 5** 4 2 8)
(1 **5 4** 2 8) ⟶ (1 **4 5** 2 8)
(1 4 **5 2** 8) ⟶ (1 4 **2 5** 8)
(1 4 2 **5 8**) ⟶ (1 4 2 **5 8**)
Second Pass:
(**1 4** 2 5 8) ⟶ (**1 4** 2 5 8)
(1 **4 2** 5 8) ⟶ (1 **2 4** 5 8)
(1 2 **4 5** 8) ⟶ (1 2 **4 5** 8)
(1 2 4 **5 8**) ⟶ (1 2 4 **5 8**)

آرایه مرتب شده است اما الگوریتم به کار خود ادامه می دهد تا به مرحله ای برسد که هیچ جابه جائی صورت نمی گیرد.

Third Pass:
(**1 2** 4 5 8) ⟶ (**1 2** 4 5 8)
(1 **2 4** 5 8) ⟶ (1 **2 4** 5 8)
(1 2 **4 5** 8) ⟶ (1 2 **4 5** 8)
(1 2 4 **5 8**) ⟶ (1 2 4 **5 8**)

آرایه مرتب شده است و الگوریتم پایان می پذیرد.

ارزیابی کارائی

اگر n تعداد عناصر لیستی است که دارد مرتب می شود. در هر مرحله یک عنصر در محل خود قرار می گیرد که در مرحله بعد نیاز به بررسی ندارد بنابراین تعداد کل مقایسه ها برابر با 1+ … +(n-2)+(n-1) می شود. حاصل جمع این عبارت مساوی n(n-1)/2 است که پیچیدگی $O(n^2)$ را دارد. پیچیدگی در بدترین حالت O(n) است و زمانی اتفاق می افتد که داده های لیست از قبل مرتب باشند و حلقه do-while تنها یکبار اجرا می شود.

مرتب سازی انتخابی

مرتب سازی انتخابی (Selection Sort) روش بهبود یافته مرتب سازی حبابی است. الگوریتم ابتدا کوچکترین عنصر را توسط جستجوی خطی پیدا می کند سپس آن را در اولین محل لیست قرار می دهد. سپس دومین عنصر کوچک را پیدا می کند و به همین ترتیب تا آخر.

پیاده سازی

شبه کد الگوریتم مرتب سازی انتخابی به صورت زیر است. آرایه A لیستی با n عنصر که اندیس آن از یک شروع می شود.

```
for (i:=1 to n-1)
   min := i
   for (j:=i + 1 to n-1)
     if A[j] > A[min] then
        min := j
     end if
   end for
   swap( A[i],  A[min] )
end for
```

درواقع لیست به دو قسمت تقسیم می شود: عناصری که قبلا مرتب شده اند که از چپ به راست در ابتدای لیست قرار می گیرند و عناصری که باید مرتب شوند.

مثال. در زیر مراحل مختلف برای مرتب کردن آرایه ای با پنج عنصر مشاهده می شود.

```
Fisrt Pass:
( 64 25 12 22 11 ) → ( 11 25 12 22 64 )
Second Pass:
( 11 25 12 22 64 ) → ( 11 12 25 22 64 )
Third Pass:
( 11 12 25 22 64 ) → ( 11 12 22 25 64 )
Forth Pass:
( 11 12 22 25 64 ) → ( 11 12 22 25 64 )
```

ارزیابی کارایی

در مقایسه با الگوریتم های دیگر مرتب سازی انتخابی به دلیل سادگی ساختار، صرف نظر از ترتیب داد های لیست همیشه یک زمان اجرا را می دهد. (n-1) جابجایی در کل موردنیاز است که نسبت به مرتب سازی حبابی کمتر است و اگر تعداد جابجایی ها مهم باشد روش مناسب تری است. تعداد مقایسه ها مانند مرتب سازی حبابی بهینه در کل $(n-1)+(n-2)+\ldots+1=\Theta(n^2)$ است.

مرتب سازی انتخابی برای ساختارهایی مانند لیست پیوندی که روش حذف و اضافه با زمان کمتری کارایی دارند مناسب است.

مقایسه الگوریتم های مرتب سازی

در جدول زیر الگوریتم های مرتب سازی از نظر زمانی با یکدیگر مقایسه شده اند. الگوریتم های زیر همگی از نوع مقایسه ای هستند.

مرتب سازی	روش	پایدار	مصرف حافظه	بهترین	بدترین	توضیحات
Bubble	تعویضی	بله	O(1)	O(n)	$O(n^2)$	الگوریتم بهینه
Selection	انتخابی	خیر	O(1)	$O(n^2)$	$O(n^2)$	
Insertion	درجی	بله	O(1)	O(n)	$O(n^2)$	بهترین حالت وقتی لیست مرتب است
Shell	درجی	خیر	O(1)	O(n log n)	$O(n \log n^2)$	
Shaker	تعویضی	بله	O(1)	$O(n^2)$	$O(n^2)$	
Quick	تقسیم بندی	خیر	O(n)	O(n log n)	$O(n^2)$	
Merge	ادغام	بله	O(n)	O(n log n)	O(n log n)	می تواند به صورت درجا با حافظه O(1) هم پیاده سازی شود.
Radix	ایندکسی	بله	O(n)	O(n log n)	$O(n^2)$	
Counting	ایندکسی	بله	O(k+n)	O(k+n)	O(k+n)	
Heap	انتخابی	خیر	O(1)	O(n log n)	O(n log n)	
Tree	انتخابی	بله	O(n)	O(n log n)	$O(n^2)$	