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، که به لیست های مرتب نیاز دارند، اهمیت دارد. مرتب سازی برای تهیه خروجی های خوانا برای انسان نیز مفید است.

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 درجا (in place).
- مصرف حافظه (و استفاده از منابع دیگر سیستم). برخی از الگوریتم های مرتب سازی برون از جا (out place) هستند. که به محل کمکی برای نگهداری داده های موقت علاوه بر داده های در حال مرتب شدن نیاز دارند.
- بعضی از الگوریتم ها بازگشتی یا غیر بازگشتی یا هردو هستند.
- پایداری. الگوریتم های مرتب سازی پایدار 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.
- متد کلی. روش مرتب سازی داده ها که می تواند درج، تعویض، انتخاب، ادغام و غیره باشد. برای مثال مرتب سازی حبابی و سریع مرتب سازی تعویضی هستند.

مرتب سازی حبابی

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

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

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

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.

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:

```
for i:= 1 to n do
   for j:=n downto i+1 do
     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 نکته است که در هر مرحله بزرگترین عنصر به انتهای لیست منتقل می شود. یعنی هر بار یک عنصر در محل خود قرار می گیرد که در گام بعدی می توان آن را در نظر نگرفت. بنابراین برای یک لیست با n عنصر در مرحله بعد نیاز به بررسی n-1 عنصر دیگر می باشد.

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:=0 to n  do
    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)

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)

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 این مجموع مساوی $n(n-1)/2$ است که پیچیدگی $O(n^2)$ را دارد. پیچیدگی در بهترین حالت O(n) است و زمانی اتفاق می افتد که داده های لیست از قبل مرتب باشد بنابراین 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 و آنرا 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:

```
for i:=0 to n-2 do
    min:=i
    for j:=(i + 1) to n-1 do
        if A[j] < A[min]then
            min:= j
        end if
    end for
    swap (A[i] , A[min])
end for
```

مثال. در زیر مراحل مختلف برای مرتب کردن ۵ عنصر "11 22 12 25 64" مشاهده می شود.

First 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 کارائی

در مقایسه با الگوریتم های دیگر مرتب سازی انتخابی، به دلیل سادگی ساختار، صرف نظر از ترتیب داده های لیست همیشه یک زمان اجرا را می دهد. (n-1) جابه جائی در کل مورد نیاز است که نسبت به مرتب سازی حبابی کمتر است و اگر تعداد جابه جائی ها مهم باشد مرتب سازی انتخابی روش مناسبی است. تعداد مقایسه ها در کل برابر است با $\Theta(n^2)$ =1+...+(n-2)+(n-1).

مرتب سازی انتخابی برای ساختارهائی مانند لیست پیوندی که روش حذف و اضافه کارائی دارند سودمند است. در این حالت کوچکترین عنصر از لیست حذف شده و سپس به انتهای مقادیری که قبلا مرتب شده اند اضافه می شود.
مثال.

64 25 12 22 11
11 64 25 12 22
11 12 64 25 22
11 12 22 64 25
11 12 22 25 64

مقایسه الگوریتم های مرتب سازی

مرتب سازی	روش	پایدار	حافظه	بهترین	بدترین	توضیحات
Bubble Sort	Exchange	Yes	O(1)	O(n)	$O(n^2)$	زمان ها در الگوریتم بهینه
Selection Sort	Selection	No	O(1)	$O(n^2)$	$O(n^2)$	
Insertion Sort	Insertion	Yes	O(1)	O(n)	$O(n^2)$	بهترین حالت وقتی لیست مرتب است
Shell Sort	Insersion	No	O(1)	O(n log n)	$O(n \log n^2)$	
Shaker Sort	exchange	Yes	O(1)	$O(n^2)$	$O(n^2)$	
Quick Sort	Partionioning	No	O(n)	O(n log n)	$O(n^2)$	
Merge Sort	Merging	Yes	O(n)	O(n log n)	O(n log n)	درالگوریتم غیربازگشتی حافظه O(1)
Radix Sort	Indexing	Yes	O(n)	O(n log n)	$O(n^2)$	
Counting Sort	Indexing	Yes	O(k+n)	O(k+n)	O(k+n)	
Heap Sort	Selection	No	O(1)	O(n log n)	O(n log n)	
Tree Sort	Selection	Yes	O(n)	O(n log n)	$O(n^2)$	