بسمه تعالی

مرتب سازی سریع(quick sort)

خلاصه

یکی از سریع ترین الگوریتم های مرتب سازی، الگوریتم quick sort می باشد که نه تنها در بحث آموزش بلکه در محیط ها و برنامه های کاربردی روزمره نیز مورد استفاده قرار می گیرد. پیچیدگی این الگوریتم در حالت میانگین ($O(n \log n$ و در بدترین حالت ($O(n^2$ می باشد. این الگوریتم مرتب سازی توسط فردی به نام هور «۱۹۶۲» ابداع گردید. این الگوریتم مشابه الگوریتم مرتب سازی ادغامی، از روش تقسیم و حل یا divide-and-conquer strategy حل می گردد.

روش حل الگوریتم

در این روش ابتدا آرایه را به دو قسمت تقسیم نموده و بعد یکی از عناصر آرایه را به عنوان محور اصلی در نظر می گیریم. «فرقی نمی کند که عنصر اولی باشد یا دومی یا وسطی و یا هر عنصر دیگر». سپس تمام عناصر کوچکتر از عنصر محور را در سمت چپ و عناصر بزرگتر را در سمت راست عنصر محور قرار می دهیم. حال نتیجه کار به صورت دو آرایه + عنصر محوری تبدیل شده است. اگر دقت کنید عناصر موجود در هر آرایه یا از عنصر محوری بزرگتر و یا کوچکتر می باشند ولی هیچ یک از دو آرایه مرتب نمی باشند.

پس باید دوباره به ازای هر یک از آرایه های حاصل و بصورت بازگشتی، الگوریتم فوق را اجرا نموده تا در نهایت آرایه اولیه مرتب گردد.

ویژگی‌های مرتب‌سازی سریع

۱. پیچیده‌گی زمانی اجرای الگوریتم در بهترین حالت ($\Theta(n \log n$ و در بدترین حالت ($\Theta(n^2$ است. با استفاده محاسبات ریاضی می‌توان نشان داد در حالت متوسط نیز مرتبه‌ی اجرا ($\Theta(n \log n$ است.

۲. این الگوریتم یک مرتب‌سازی درجا است. یعنی میزان حافظه‌ی مصرفی الگوریتم مستقل از طول آرایه است.

۳. زمانی که تعداد عناصر آرایه کم باشد، سرعت اجرای مرتب‌سازی درجی بهتر از مرتب‌سازی سریع است. به همین دلیل طی مراحل بازگشتی مرتب‌سازی سریع، اگر طول بازه عدد کوچکی باشد، معمولا بازه با مرتب‌سازی درجی مرتب می‌شود.

۴. الگوریتم مرتب‌سازی سریع با پیاده‌سازی فوق یک روش ناپایدار است. چنین الگوریتمی لزوما ترتیب عناصر با مقدار یکسان را پس از مرتب‌سازی حفظ نمی‌کند.

۵. انتخاب عنصر محوری بحث مفصلی دارد. اما در کل یک انتخاب آزاد است. می‌توان عنصر اول، عنصر آخر، یا هر عنصر دیگری را به عنوان عنصر محوری انتخاب کرد. حتی لازم نیست از ابتدا تا انتها از یک روش انتخاب استفاده کرد. یکی از روش‌های رایج، انتخاب یک عنصر تصادفی به عنوان عنصر محوری است. اگرچه انتخاب عنصر محوری مناسب باعث بالا رفتن کارایی الگوریتم می‌شود، اما عموما هزینه‌ی لازم برای یافتن چنین محوری بالا بوده و مقرون به صرفه نیست.

الگوریتم مرتب سازی سری

```
1.  procedureQUICKSORT (A,q,r)
2.  begin
3.  if q< r then
4.  begin
5.  x:= A[q];
6.  s:= q;
7.  for i:= q+ 1 to  r do
8.  if A[i] x then
9.  begin
10. s:= s+ 1 ;
11. swap(A[s],  A[i]);
12. end if
13. swap(A[q],  A[s]);
14. QUICKSORT (A,q,s)
15. QUICKSORT (A,s+ 1,r)
16. end if
17. endQUICKSORT
```

✓ **فرمول موازی برای یک CRCW PRAM**

RAM را از نوع CWCR فرض کنید. همه پروسه ها در ابتدای کار ایجاد شوند و همگی سعی می کنند مقدار عنصرشان را در متغیر root بنویسند سپس پروسه ها غیر از پروسه ای که موفق شده مقدار خود را در root بنویسد سعی می کنند root را بخوانند، اگر عنصرشان از این مقدار کوچکتر بود خود را LeftChild منتسب می کنند وگرنه RightChild می شوند.

این عمل به صورت بازگشتی دنبال می شود و در نهایت یک درخت با عمق متوسط log n ایجاد می شود.

در خط ۱۱ کد شرط تساوی برای حفظ تعادل الگوریتم چک می شود.

(a)

1	2	3	4	5	6	7	8
33	21	13	54	82	33	40	72

(b) root = 4

(c)

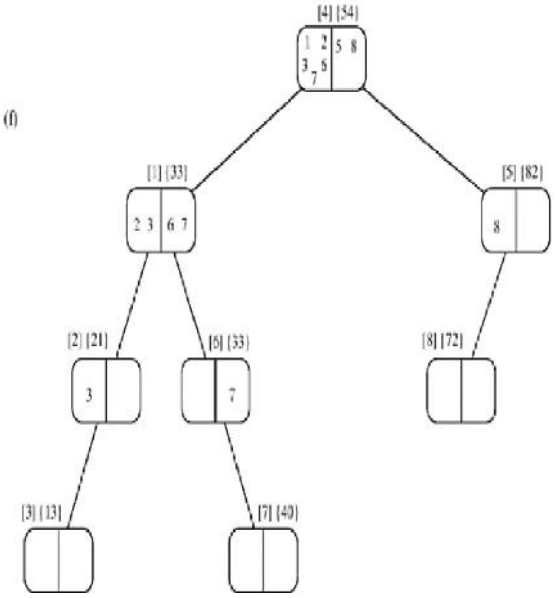
	1	2	3	4	5	6	7	8
leftchild				1				
rightchild				5				

(d)

	1	2	3	4	5	6	7	8
leftchild	2			1	8			
rightchild	6			5				

(e)

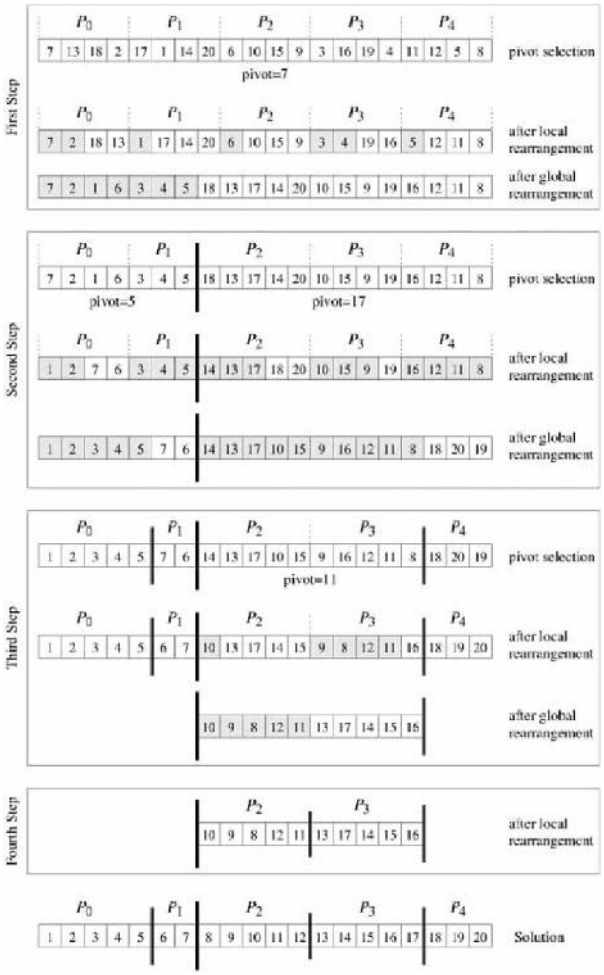
	1	2	3	4	5	6	7	8
leftchild	2	3		1	8			
rightchild	6			5		7		



✓ **فرمول موازی Shared-Address-Space**

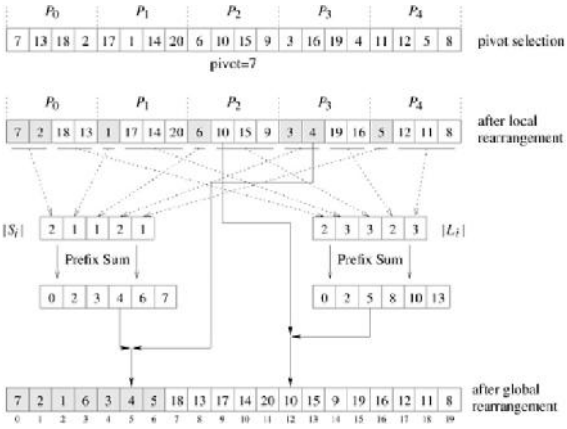
عناصر به صورت موازی بین پردازنده ها تقسیم شود. هر پردازنده حافظه محلی خود را دارد که برای پروسه های دیگر نیز قابل دسترسی است روند زیر اجرا می شود :

۱. Pivot تصادفی شده و مقدار آن به همه اطلاع داده شود(Broad Cast)
۲. کلیه پردازنده ها بر اساس مقدار Pivot عناصرشان را به دو دسته U_i و L_i تقسیم کنند.
۳. U_i ها و L_i ها با هم merge شده و مجموعه های U و L را تشکیل دهند.
۴. پردازنده ها متناسب با اندازه U و L به دو دسته تقسیم شده و هر دسته همین کار را به صورت بازگشتی انجام دهد.
۵. تا جایی ادامه می یابد که هر زیر بلوک ایجاد شده به یک پروسه واگذار شود در این مرحله هر بلوک به صورت سریال مرتب می شود.



الگوریتم مرتب سازی موازی

```
1.  procedureBUILD TREE (A[1... n])
2.  begin
3.  foreach process i do
4.  begin
5.  root := i ;
6.  parenti:= root ;
7.  leftchild[i] := rightchild[i] := n+ 1 ;
8.  end for
9.  repeatfor each process i root do
10. begin
11. if(A[i] < A[parenti]) or
12. (A[i]= A[parenti] and i<parenti) then
13. begin
14. leftchild[parenti] := i ;
15. if i= leftchild[parenti] then exit
16. else parenti:= leftchild[parenti];
17. end for
18. else
19. begin
20. rightchild[parenti] := i ;
21. if i= rightchild[parenti] then exit
22. else parenti:= rightchild[parenti ;[
23. end else
24. end repeat
25. endBUILD_TREE
```



پیچیدگی زمانی در فرمول Shared-Address-Space :

۱. Broad Cast کردن pivot: $\Theta(\log p)$
۲. تقسیم محلی و به دست آوردن L_iوU_i در هر پروسه: $\Theta(n/p)$
۳. prefix-sum جهت تشخیص مکان هر عنصر در لیست کلی: $\Theta(\log p)$
۴. انتقال به لیست کلی: $\Theta(n/p)$

زمان کل $\Theta(n/p) + \Theta(\log p)$ است پس زمان اجرای کل الگوریتم برابر است با:

$$T_P = \overbrace{\Theta\left(\frac{n}{p}\log\frac{n}{p}\right)}^{\text{local sort}} + \overbrace{\Theta\left(\frac{n}{p}\log p\right)}^{\text{array splits}} + \Theta(\log^2 p).$$

زمان مورد نیاز در هر مرحله ضربدر تعداد مراحل$(\log p)$

✓ **فرمول موازی Message-Passing**

۱. پروسه ها به دو دسته High و Low تقسیم می شوند.
۲. کلیه مراحل مانند مرحله قبل، بجای اینکه فقط مرحله Merging بین همه پروسه ها باشدبه صورت جفت جفت بین دو پروسه انجام می شود.(یک پروسه از دسته High و یک پروسه از دسته Low)
۳. پردازنده با اندیس کوچکتر نیمه بزرگتر را برای پردازنده High می فرستد و بالعکس.
۴. بنابراین عناصر کوچکتر از pivot در پردازنده های Low، نیمه بزرگتر به پردازنده High منتقل می شوند.

این عملیات به صورت بازگشتی دنبال می شود تا جایی که به هر پروسه یک دسته عنصر اختصاص یابد.

پیچیدگی زمانی هر مرحله:

۱. Broad Cast کردن pivot: $\Theta(\log p)$
۲. تقسیم محلی در هر پروسه: $\Theta(n/p)$
۳. تبادل عناصر بین جفت پروسه ها: $\Theta(n/p)$

زمان شبیه حالت قبل است فقط در اینجا زمان مرحله ۳ وابسته به پهنای باند است.

مرجع

Introduction to Parallel Computing, Second Edition
By Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar
Publisher: Addison Wesley
Pub Date: January 16, 2003
Pages: 367-375