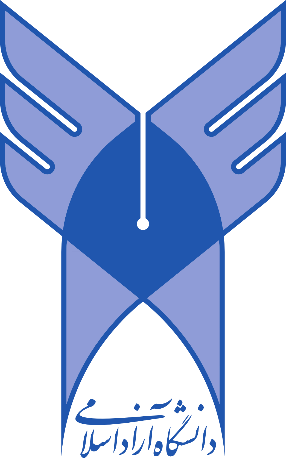
**C:\Program Files (x86)\Hamoon Soft\Chalipa\Chalipa Clipart\Brdr01.wmf**

دانشگاه آزاد اسلامی تهران مرک

مجتمع آیت الله هاشمی رفسنجانی

عنوان :

**تحلیل پیچیدگی الگوریتم ها**

رشته تحصیلی :

**مهندسی کامپیوتر نرم افزار**

استاد راهنما:

**سرکارخانم دکتر نیره زاغری**

تهیه کنندگان:

**حسن پرسته**

**مجتبی سیانکی**

**زمستان 99**

**C:\Program Files (x86)\Hamoon Soft\Chalipa\Chalipa Clipart\Brdr01.wmf**

**الگوریتم های مرتب سازی**

منظور از مرتب سازی داده، چیدمان داده ‌ها در قالبی خاص است. الگوریتم مرتب سازی روشی برای چیدمان داده ‌ها با ترتیبی خاص تعیین می ‌کند. اغلب ترتیب های رایج به صورت ترتیب عددی یا الفبای هستند. اهمیت مرتب‌ سازی در این نکته است که جستجوی داده ‌ها در صورت مرتب بودن می ‌تواند تا سطح بالایی بهینه سازی شود. مرتب‌ سازی همچنین می ‌تواند برای نمایش داده ‌ها در قالب ‌های خواناتر کمک کند.

**اصطلاح‌ های مهم**

برخی اصلاح‌ ها به طور کلی در مباحث مرتبط با تکنیک‌ های مرتب ‌سازی بیشتر به چشم می ‌آیند که در ادامه برخی از آن‌ ها را ارائه کرده‌ ایم:

**ترتیب صعودی:** یک توالی از مقادیر مختلف، زمانی دارای ترتیب صعودی نامیده می ‌شود که همه عناصر پشت سر هم بزرگ ‌تر از عنصر قبلی ‌شان باشند. برای نمونه فهرست 1، 3، 4، 8، 9 دارای ترتیب صعودی است چون هر عنصر از عنصر قبلی خود بزرگ ‌تر است.

**ترتیب نزولی:** یک توالی از مقادیر، زمانی دارای ترتیب نزولی خوانده می ‌شود که همه عناصر پشت سر هم از عنصر قبلی خود کوچک ‌تر باشند. برای نمونه، 9، 8، 6، 4، 3، 1 دارای ترتیب نزولی است و هر عنصر از عنصر قبلی خود کوچک ‌تر است.

**ترتیب غیر افزایشی:** یک توالی از مقادیر دارای ترتیب غیر افزایشی نامیده می ‌شود، وقتی همه عناصر پشت سر هم کمتر یا مساوی عنصر قبلی‌ شان باشند. این ترتیب زمانی رخ می‌ دهد که در توالی مقادیر مورد نظر، عناصری تکراری وجود داشته باشند. برای نمونه، 9، 8، 6، 3، 3، 1 دارای ترتیب غیر افزایشی است، چون هر عنصر کوچک ‌تر یا مساوی (در مورد 3) از عنصر قبلی است، اما هیچ عنصری از عنصر قبلی خود بزرگ‌ تر نیست.

**ترتیب غیر کاهشی:** یک توالی از مقادیر، دارای ترتیب غیر کاهشی خوانده می ‌شود، اگر عناصر پشت سر هم همواره بزرگ ‌تر یا مساوی عنصر قبلی ‌شان باشند. این ترتیب زمانی رخ می ‌دهد که در یک توالی عناصر تکراری وجود داشته باشد. برای نمونه لیست 1، 3، 3، 6، 8، 9 یک فهرست با ترتیب غیر کاهشی است، چون هر عنصر کوچک ‌تر یا مساوی (در مورد 3) از عنصر قبلی است؛ اما بزرگ ‌تر نیست.

**در ادامه الگوریتم مرتب سازی حبابی را مشاهده می کنید!**

**الگوریتم مرتب ‌سازی حبابی (Bubble Sort)**

مرتب ‌سازی حبابی یک الگوریتم مرتب ‌سازی ساده است. این الگوریتم مرتب‌ سازی یک الگوریتم مبتنی بر مقایسه است که در آن هر جفت از عناصر مجاور با هم مقایسه می ‌شوند و در صورتی که در ترتیب مطلوب نباشند با هم تعویض می ‌شوند. این الگوریتم برای مجموعه داده‌ های بزرگ مطلوب نیست، زیرا پیچیدگی حالت میانگین و بدترین حالت آن برابر با است که در آن تعداد آیتم ‌هایی است که باید مرتب شوند.

* **ویژگی‌های مرتب‌سازی حبابی**

1. مرتب‌سازی حبابی یک روش مرتب‌سازی درجا است. یعنی نیاز به فضای کمکی نداشته و با جابجا کردن عناصر در داخل خود لیست، آنها را مرتب می‌کند.
2. مرتب‌سازی حبابی یک روش مرتب‌سازی پایدار است. یعنی در حین مرتب‌سازی ترتیب عناصری که مقدار یکسانی دارند تغییر نمی‌کند.
3. پیچیدگی زمانی آن به صورت است.

**طرز کار مرتب سازی حبابی:**

به عنوان مثال یک آرایه نامرتب را در نظر می‌ گیریم. الگوریتم مرتب ‌سازی حبابی باید مرتبه روی این آرایه کار کند تا آن را مرتب و منظم کند.

۱. ابتدا، دو عنصر اول آرایه با یکدیگر مقایسه می‌شوند و با توجه به آنکه ۵ از ۱ بزرگتر است (۱<۵)، این دو عنصر با یکدیگر جا به جا می‌شوند.

( 5 1 4 2 8 ) –> ( 1 5 4 2 8 )

۲. در اینجا، عناصر دوم و سوم آرایه مقایسه می‌شوند و با توجه به اینکه ۵ از ۴ بزرگ‌تر است (۴<۵)، این دو عنصر با یکدیگر جا به جا می‌شوند.

( 1 5 4 2 8 ) –> ( 1 4 5 2 8 )

۳. اکنون، عنصر سوم و چهارم آرایه مقایسه می‌شوند و با توجه به اینکه ۲ از ۵ کوچک‌تر است (۲<۵)، این دو عنصر با یکدیگر جا به جا می‌شوند.

( 1 4 5 2 8 ) –> ( 1 4 2 5 8 )

۴. در اینجا، عنصر چهارم و پنجم آرایه مقایسه می‌شود و چون ۵ از ۸ کوچک‌تر است (۵<۸) دو عنصر در جای خود بدون هر گونه جا به جایی باقی می‌مانند؛ چون در واقع، ترتیب (صعودی) در آن‌ها رعایت شده است.

( 1 4 2 5 8 ) –> ( 1 4 2 5 8 )

اکنون یک دور کامل در آرایه زده شد. دومین دور نیز به شیوه بیان شده در بالا انجام می‌شود.

۱. جا به جایی اتفاق نمی‌افتد.

( 1 4 2 5 8 ) –> ( 1 4 2 5 8 )

۲. با توجه به بزرگ‌تر بودن ۴ از ۲ (۲<۴)، این دو عنصر با یکدیگر جا به جا می‌شوند.

( 1 4 2 5 8 ) –> ( 1 2 4 5 8 )

۳. جا به جایی اتفاق نمی‌افتد.

( 1 2 4 5 8 ) –> ( 1 2 4 5 8 )

۴. جا به جایی اتفاق نمی‌افتد.

( 1 2 4 5 8 ) –> ( 1 2 4 5 8 )

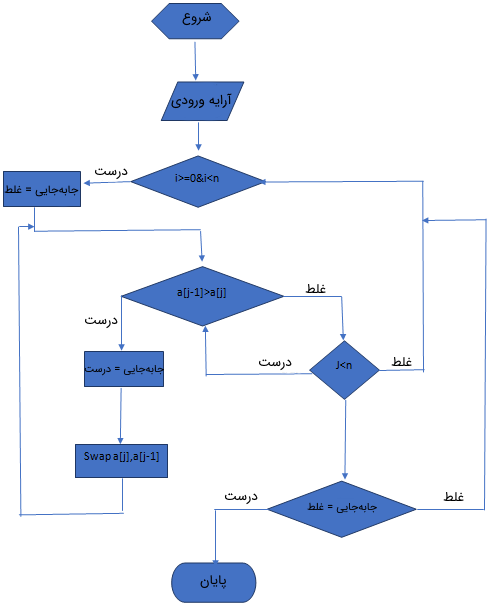
در حال حاضر، آرایه مرتب شده است، اما الگوریتم نمی‌داند که آیا کار به پایان رسیده یا خیر؛ بنابراین، به یک دور کامل دیگر بدون انجام هرگونه جا به جایی نیاز دارد تا بفهمد که مرتب‌سازی با موفقیت به پایان رسیده است.

( 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 )

فلوچارت الگوریتم مرتب سازی حبابی:



**کد الگوریتم مرتب سازی حبابی به C++:**

****

**محاسبه ی پیچیدگی زمانی در بهترین ، متوسط و بدترین حالت:**

* **بهترین حالت**
* **حالت متوسط**
* **بدترین حالت**

حال چگونه این ها محاسبه شدند؟

**توضیح اجرای الگوریتم:**

فرض کنیم لیستی با n عنصر با روش مرتب‌سازی حبابی مرتب می‌شوند. عمل اصلی روش‌های مرتب‌سازی مبتنی بر مقایسه، مقایسه‌ها هستند. در مرحله‌ی اول، n - 1 مقایسه صورت می‌گیرد، تا بزرگترین عنصر به انتهای لیست منتقل شود. در مرحله‌ی دوم، n - 2 مقایسه و به همین ترتیب، در مرحله‌ی iام (i < n) تعداد n - i مقایسه صورت می‌گیرد. این تعداد را می‌توان از کد برنامه‌نویسی ذکر شده‌ی فوق هم استخراج کرد. پس تعداد کل مقایسه‌ها برای مرتب کردن n عنصر برابر است با:

چنین عبارتی از مرتبه ی است.

بهینه سازی الگوریتم مرتب سازی حبابی؟

برای بهینه‌تر شدن الگوریتم، کدها را به صورت زیر تغییر می‌دهیم:

void bubble\_sort\_2(int *arr*[], int *n*)

{

    int i, j, t, c, hasChange;

    for (i = *n* - 2; i >= 0; i--)

    {

        hasChange = 0;

        for (j = 0; j <= i; j++)

            if (*arr*[j] > *arr*[j + 1])

            {

                t = *arr*[j];

*arr*[j] = *arr*[j + 1];

*arr*[j + 1] = t;

                hasChange = 1;

            }

        if (hasChange == 0)

            break;

    }

}

مقدار متغیر hasChange قبل از شروع هر مرحله صفر شده و در صورت جابجا شدن محتوای دو خانه از آرایه مقدار آن به یک تغییر پیدا می‌کند. بنابراین، اگر مقدار آن در پایان مرحله صفر باشد، یعنی هیچگونه جابجایی در لیست صورت نگرفته و می‌توان نتیجه گرفت لیست مرتب است (چرا؟).

چنین کدی مرتبه‌ی زمانی اجرای الگوریتم را در بهترین حالت به تقلیل می‌دهد. چرا که اگر لیست از همان ابتدا مرتب باشد، با تمام شدن اولین مرحله (با n-1 مقایسه) و بررسی متغیر C ، مرتب بودن لیست مشخص شده و ادامه روند مرتب‌سازی متوقف می‌شود.

**ویژگی های مرتب سازی حبابی**

1. **همانگونه که بحث شد، پیچیدگی زمانی اجرای این الگوریتم در بدترین حالت و حالت متوسط از مرتبه‌ی است. پیچیدگی زمانی بهترین حالت، با تابع مرتب سازی اول (بهینه نشده) از مرتبه ی و با تابع مرتب سازی دوم (نسخه ی بهینه شده) از مرتبه ی است.**
2. **مرتب‌سازی حبابی یک روش مرتب‌سازی درجا است. یعنی نیاز به فضای کمکی نداشته و با جابجا کردن عناصر در داخل خود لیست، آنها را مرتب می‌کند.**
3. **مرتب‌سازی حبابی - با پیاده‌سازی به یکی از روش‌های فوق - یک روش مرتب‌سازی پایدار است. یعنی در حین مرتب‌سازی ترتیب عناصری که مقدار یکسانی دارند تغییر نمی‌کند. اگر در قطعه کدهای فوق، در مقایسه‌ی عناصر آرایه به جای < از =< استفاده می‌کردیم، مرتب‌سازی ناپایدار می‌شد. چرا که عناصری با مقادیر یکسان را نیز جابجا کرده و ترتیب آنها را به هم می‌زد.**