الگوریتم های Sort در زبان برنامه نویسی C#

در زبان برنامه نویسی C# توابع مرتب‌سازی مثل هر زبان برنامه‌نویسی دیگر از الگوریتم‌های متنوعی استفاده می‌کنند. بعضی از معروف‌ترین الگوریتم‌های مرتب‌سازی در C# عبارتند از:

۱. Bubble Sort

۲. Selection Sort

۳. Insertion Sort

۴. Merge Sort

۵. Quick Sort

۶. Shell Sort

۷. Counting Sort

۸. Radix Sort

۹. Bucket Sort

۱۰. Heap Sort

دلیل استفاده از این الگوریتم‌ها در جهت بهینه‌سازی سرعت عملیات و کاهش منابع مصرفی می‌باشد. هر الگوریتم ممکن است در شرایط مختلفی کارایی بهتری داشته باشد. در ادامه به توضیح نحوه عملکرد هر یک از این الگوریتم‌ها، مزایا و معایب، و همچنین پیچیدگی زمانی هر یک می‌پردازیم.

۱. Bubble Sort:

دلیل : دلیل استفاده از الگوریتم Bubble Sort در زبان C# می‌تواند به دلیل سادگی و قابلیت درک آن باشد. همچنین این الگوریتم می‌تواند برای آرایه‌های کوچک و با تعداد کمی از آیتم‌ها به خوبی عمل کند.

نحوه : در این الگوریتم، از ابتدای لیست شروع کرده و هر دو المان مجاور را با یکدیگر مقایسه می‌کنیم. در صورتی که دو مقدار به طور معکوس مرتب باشند ، موقعیت آنها را جابجا می‌کنیم. این عملیات را به تعداد المان‌ها تکرار می‌کنیم. سپس این روند را تا جایی که هیچ جابه‌جایی دیگری انجام نشود. عیب این الگوریتم این است که برای مرتب‌سازی آرایه‌های بزرگ زمان بسیار زیادی می‌برد.

پیچیدگی زمانی: بهترین حالت ← $O(n)$ و حالت میانگین ← $O(n^2)$ و بدترین حالت ← $O(n^2)$

۲. Selection Sort:

دلیل : دلیل استفاده از الگوریتم Selection Sort در زبان C# می‌تواند به دلیل سادگی و قابلیت درک آن باشد. همچنین این الگوریتم، برای آرایه‌های کوچک با تعداد کمی از آیتم‌ها کارآمد است و به عنوان یک الگوریتم پایه‌ای مرتب‌سازی در زبان C# استفاده می‌شود اما Selection Sort به دلیل محدودیت‌هایش قابل استفاده در زمینه‌های مختلفی مانند ترتیب‌بندی لیست‌ها، ایجاد لیست‌های جستجو و بهبودیافته کردن دیگر الگوریتم‌هاست.

نحوه : در این الگوریتم، در هر مرحله به دنبال کوچک‌ترین/بزرگ‌ترین مقدار در باقی مانده لیست هستیم و آن را با اولین مقدار در لیست جابه‌جا می‌کنیم و سپس به سراغ المان بعدی می‌رویم.

پیچیدگی زمانی: بهترین حالت ← $O(n^2)$ و حالت میانگین ← $O(n^2)$ و بدترین حالت ← $O(n^2)$

۳. Insertion Sort:

دلیل : به دلیل سادگی ، به دلیل تعداد کم مقایسات و جابجایی‌ها ، پایداری(ترتیب دو عنصر با مقادیر برابر در آرایه حفظ می‌شود) و همچنین به صورت تجربی اثبات شده که اگر تعداد داده ها کمتر از ۲۰ تا باشد، از همه سریع تر است.

نحوه : الگوریتم Insertion Sort یکی از الگوریتم‌های مرتب‌سازی آرایه‌هاست که براساس رویکردی مبتنی بر قرار دادن یک عنصر در محل مناسب در آرایه، آرایه را به صورت مرتب می‌کند. در این الگوریتم، با اضافه کردن هر عنصر به آرایه، آن عنصر را با عناصر قبلی آرایه مقایسه می‌کنیم تا محل مناسب برای قرار دادن آن در آرایه پیدا کنیم.

پیچیدگی زمانی: بهترین حالت ← $O(n)$ و حالت میانگین ← $O(n^2)$ و بدترین حالت ← $O(n^2)$

۴. Merge Sort :

دلیل : merge sort به دلیل کارایی بالا، پایدار بودن و قابل استفاده در حافظه کم، یکی از الگوریتم‌های مرتب‌سازی پرکاربرد در زبان سی شارپ است.

نحوه : Merge sort با تقسیم آرایه ورودی به دو نیمه شروع می شود. الگوریتم به صورت بازگشتی خود را برای هر یک از آن نیمه ها فراخوانی می کند تا زمانی که هیچ نیم آرایه ای برای تقسیم در طول فرآیند مرتب سازی وجود نداشته باشد.

پیچیدگی زمانی: بهترین حالت ← O(n.log n) و حالت میانگین ← O(n.log n) و بدترین حالت ← O(n.log n)

۵. Quick sort :

دلیل: الگوریتم Quick sort به دلیل استفاده از روش تقسیم و حل، سرعت بالایی در مرتب‌سازی دارد و به خصوص برای داده‌های بزرگ مناسب است.

قابل پاسخگوئی به داده‌های نامرتب با حجم بالاست.

با توجه به روش تقسیم و حل، به صورت کارآمد و با حداقل تعداد مقایسات، داده‌ها را مرتب می‌کند.

نحوه : در این پیاده‌سازی، تابع Quicksort با دریافت آرایه و محدوده اولیه برای مرتب‌سازی، الگوریتم Quick sort را اجرا می‌کند. تابع Partitionنیز برای جداسازی عنصر Pivot و تقسیم آرایه به دو قسمت استفاده می‌شود.

پیچیدگی زمانی: بهترین حالت ← O(n.log n) و حالت میانگین ← O(n.log n) و بدترین حالت ← $O(n^2)$

۶. Shell Sort :

دلیل: سرعت و کارآیی بسیار بالایی که دارد، در مواقعی که نیاز به مرتب سازی آرایه‌های نسبتاً بزرگ و نامرتب است، مورد استفاده قرار می‌گیرد.

نحوه : در این الگوریتم، ابتدا فاصله‌ی دلخواهی بین عناصر آرایه انتخاب می‌شود. سپس عناصر آرایه در فاصله‌های مختلف با هم مقایسه و مرتب می‌شوند. در مرحله بعد، فاصله‌ی انتخاب شده را نصف می‌کنیم و دوباره عناصر آرایه را در فواصل مختلف مقایسه و مرتب می‌کنیم. این عملیات به همین صورت ادامه می‌یابد تا به فاصله‌ی ۱ برسیم و آرایه نهایی مرتب شده باشد.

پیچیدگی زمانی: بهترین حالت ← O(n.log n) و حالت میانگین ← O(n.log n) و بدترین حالت ← $O(n^2)$

۷. Counting Sort

دلیل: برای مرتب کردن آرایه‌هایی با مقادیر عددی محدود استفاده می‌شود و از مزایای زیادی مانند سرعت بالا، پایداری در برابر داده‌های پرتکرار و کاربردی بودن در حوزه‌هایی مانند محاسبات علمی و پردازش تصویر برخوردار است.

در این الگوریتم، مجموع تعداد عناصر آرایه و تعداد مقادیر ممکن (maxVal) در آرایه، از مرتبه‌ی O(n+maxVal) است که بسیار بهینه است. بنابراین، برای مرتب کردن آرایه‌هایی با محدوده‌ی مقادیر کوچک و متنوع، این الگوریتم به عنوان یکی از بهترین گزینه‌ها در نظر گرفته می‌شود.

نحوه : نیاز به تعریف یک آرایه از اعداد و یک آرایه پشتیبان (auxiliary array) داریم. همچنین برای پیدا کردن بزرگترین عدد در آرایه، از حلقه for استفاده می‌کنیم. سپس آرایه پشتیبان را مقداردهی پیش فرض می‌کنیم. در مرحله‌ی بعد، تعداد تکرارهای عناصر آرایه را در آرایه پشتیبان ثبت می‌کنیم. در ادامه، تعداد کل تکرارات را محاسبه کرده و سپس آرایه را با توجه به تعداد تکرارات عناصر مرتب می‌کنیم.

پیچیدگی زمانی: بهترین حالت ← O(n) و حالت میانگین ← O(n+k) و بدترین حالت ← O(k) k: بزرگترین عنصر آرایه

۸. Radix Sort

دلیل: الگوریتم مرتب سازی Radix Sort نیز مانند Counting Sort برای مرتب کردن آرایه‌هایی با مقادیر عددی محدود استفاده می‌شود اما در مقایسه با Counting Sort، عملکرد بهتری دارد و می‌تواند با محدودیت‌های Counting Sort سازگاری بیشتری داشته باشد. از این رو، Radix Sort در برخی موارد به عنوان جایگزینی مناسب برای Counting Sort درنظر گرفته می‌شود.

با توجه به اینکه الگوریتم Radix Sort از حلقه‌ها و شیب ها استفاده نمی‌کند و از الگوریتم Counting Sort به عنوان زیر الگوریتم استفاده می‌کند، عملکرد آن از الگوریتم Counting Sort بهتر در برخی موارد بوده و به دلیل کاربست گسترده‌تر در مسائل بزرگ و پیچیده، به عنوان الگوریتمی مناسب در بسیاری از سامانه‌ها و برنامه‌های کامپیوتری شناخته می‌شود.

نحوه : الگوریتم Radix Sort برای مرتب کردن آرایه‌هایی با اعداد صحیح مثبت استفاده می‌شود. این الگوریتم برای مرتب کردن اعداد، از روش‌هایی مانند LSD (Least Significant Digit) و MSD (Most Significant Digit) استفاده می‌کند. در روش LSD، به ازای هر رقم اعداد، اعداد با توجه به مقدار آن رقم مرتب می‌شوند و سپس به رقم بعدی می‌رویم. در روش MSD، عکس این روند صادق است و از رقم بیشتر شروع به مرتب‌سازی می‌شود.

پیچیدگی زمانی: بهترین حالت ← O(c*n) و حالت میانگین ← O(p(n+d)) و بدترین حالت ← $O(d^2)$

'p' is passes and each digit is going to have up to 'd' different values, 'c' Is the number of digits in the largest number

۹. Bucket Sort

دلیل: الگوریتم مرتب سازی Bucket Sort مانند Counting Sort و Radix Sort برای مرتب کردن آرایه‌هایی با مقادیر عددی محدود استفاده می‌شود.

دلیل استفاده از الگوریتم Bucket Sort در زبان #C همانند دلیل استفاده از الگوریتم‌های مرتب سازی دیگر است. با توجه به اینکه الگوریتم Bucket Sort از حلقه‌ها و شیب ها استفاده نمی‌کند و می‌تواند با محدودیت‌های دیگری نیز سازگاری داشته باشد، عملکرد بهتری نسبت به الگوریتم‌های دیگر در برخی موارد دارد. به علاوه، توانایی این الگوریتم در کاربرد و پیاده سازی آن در بسیاری از مسائل، مهم است. برای مثال، در دنیای وب، الگوریتم Bucket Sort برای مرتب کردن لیست قیمت‌های محصولات در فروشگاه‌های آنلاین استفاده می‌شود.

نحوه : در این الگوریتم، آرایه ورودی ابتدا به بخش‌های کوچکتر تقسیم می‌شود که به آن‌ها Bucket گفته می‌شود، سپس عناصر در هر Bucket با استفاده از یک الگوریتم مرتب سازی انجام می‌شود و سپس عناصر به ترتیب درست با قرار دادن بکت‌ها در کنار هم در آرایه نهایی چیده می‌شوند.

پیچیدگی زمانی: بهترین حالت ← $O(n+k)$ و حالت میانگین ← $O(n+k)$ و بدترین حالت ← $O(n^2)$ k: تعداد باکت ها

۱۰. Heap Sort

دلیل: استفاده از این الگوریتم در زبان #C به دلیل توانایی آن در مرتب سازی داده‌هایی با حجم بالا و همچنین قابلیت استفاده آن در زمان‌هایی که حجم زیادی از حافظه در دسترس نیست، منطقی است. همچنین، الگوریتم Heap Sort از الگوریتم مرتب سازی Quick Sort با ازاین خصوص مزایایی نظیر قابلیت تطبیق با حجم داده‌های بزرگتر، کمتر بودن بهترین، بدترین و متوسط مورد انتظار زمان اجرا و عملکرد ایمن‌تر در حالت‌های خاص (مانند زمانی که عناصر آرایه به ترتیب شده باشند) برخوردار است. در نهایت، استفاده از الگوریتم Heap Sort در بسیاری از موارد به علت سرعت و کارایی آن توصیه می‌شود.

نحوه : در یک max-heap یا min-heap بزرگترین یا کوچکترین مقدار بین داده‌ها همواره در ریشه‌ی درخت قرار دارد . .با حذف این عنصر از درخت، بزرگترین یا کوچکترین عنصر بعدی مجددا در ریشه قرار می‌گیرد .به این ترتیب با حذف متوالی عناصر درخت heap و درج آنها در محل جدید، یک آرایه‌ی مرتب‌شده‌ی نزولی یا صعودی به دست خواهد آمد.

پیچیدگی زمانی: بهترین حالت ← $O(n)$ و حالت میانگین ← $O(n.\log n)$ و بدترین حالت ← $O(n.\log n)$

اعضای گروه: آرمان مقیمی _ علی عدل یار

4002164006 _ 4002164007