به نام خدا

**توضیح الگوریتم های مرتب سازی:**

**الگوریتم Quick Sort:**

الگوریتم مرتب‌ سازی سریع از نوع الگوریتم ‌های مقایسه‌ای است و تکنیک تقسیم و غلبه برای مرتب ‌سازی استفاده می‌کند.

**مراحل الگوریتم quick sort:**

1. **انتخاب یک عنصر محوری:**

اولین مرحله انتخاب یک عنصر محوری از آرایه است. این عنصر به عنوان یک نقطه مرجع برای تقسیم آرایه به دو قسمت استفاده می شود.

1. **پارتیشن بندی آرایه:**

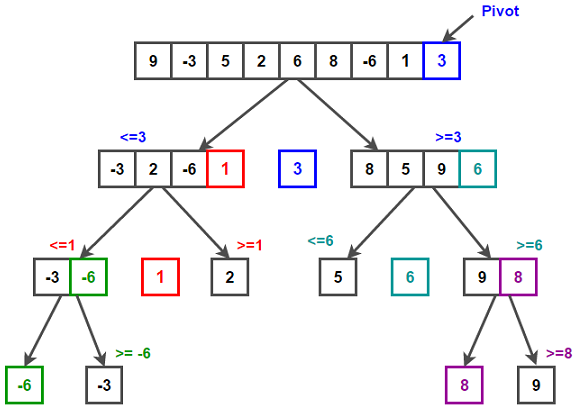
مرحله بعدی این است که آرایه را به دو قسمت تقسیم می شود، یکی شامل عناصر کمتر از عنصر محوری و دیگری حاوی عناصر بزرگتر از عنصر محوری است. برای هر عنصر بررسی می شود که آیا کوچکتر یا بزرگتر از عنصر محوری است. اگر کمتر از پیوت باشد در پارتیشن سمت چپ و اگر بزرگتر از پیوت باشد در پارتیشن سمت راست قرار می گیرد.

1. **مرتب سازی بازگشتی پارتیشن ها:**

بعد از پارتیشن بندی ارایه، الگوریتم به صورت بازگشتی همان فرآیند را برای پارتیشن های چپ و راست اعمال می کند. این کار تا زمانی ادامه می‌یابد که پارتیشن‌ها فقط یک عنصر داشته باشند، در این مرحله الگوریتم، آرایه مرتب شده را برمی‌گرداند.

1. **ترکیب پارتیشن ها:**

پس از مرتب شدن پارتیشن های چپ و راست، الگوریتم آنها را ترکیب می کند تا آرایه مرتب شده نهایی را تولید کند.



الگوریتم quick sort

**مزایا و معایب الگوریتم QuickSort:**

**مزایا:**

* **سرعت**: quicksort دارای میانگین پیچیدگی زمانی O(n log n) است که آن را به یکی از سریع ترین الگوریتم های مرتب سازی به طور متوسط تبدیل می کند. این بدان معناست که برای مرتب سازی مجموعه داده های بزرگ مناسب است.
* **استفاده کارآمد از حافظه**: یک الگوریتم درجا است، به این معنی که داده ها را بدون نیاز به حافظه اضافی برای ذخیره سازی موقت، درون همان آرایه مرتب می کند. این باعث می شود در سناریوهایی با حافظه محدود مفید باشد.
* **پیاده سازی ساده:** مفهوم کلی تقسیم بندی آرایه حول یک عنصر محوری نسبتا ساده است و درک و پیاده سازی آن را در مقایسه با دیگر از الگوریتم های مرتب سازی آسان تر می کند.
* **قابل انطباق**: در حالی که به طور ذاتی تطبیقی نیست، بهینه‌سازی‌هایی مانند انتخاب تصادفی محور می‌توانند عملکرد آن را در شرایط مختلف به طور قابل توجهی بهبود بخشد.

**معایب:**

* **بدترین حالت پیچیدگی زمانی(worst case) :** در بدترین حالت، که در آن عنصر محوری به طور مداوم آرایه را به طور ناهموار تقسیم می کند، پیچیدگی زمانی می تواند به O(n^2) کاهش یابد که آن را به طور قابل توجهی کندتر از عملکرد متوسط آن می کند.
* **ناپایداری**: quicksort ترتیب اصلی عناصر برابر را حفظ نمی کند، که ممکن است در برنامه های خاص بحرانی باشد. اگر ترتیب مهم است از یک الگوریتم مرتب سازی پایدار مانند merge sort استفاده می شود.
* **حساسیت نسبت به انتخاب پیوت**: عملکرد تا حد زیادی به عنصر محوری انتخاب شده بستگی دارد. انتخاب ضعیف می تواند منجر به بدترین حالت شود. تصادفی کردن انتخاب پینوت می تواند این مشکل را کاهش دهد اما پیچیدگی را افزایش می دهد.
* **با لیست های پیوندی خوب کار نمی کند**: quicksort در لیست های پیوندی و سایر ساختارهای مبتنی بر اشاره گر ضعیف عمل می کند.

**الگوریتم Heap Sort:**

الگوریتم مرتب‌ سازی هرمی از ساختار داده‌های هرمی برای سازمان‌ دهی عناصر یک آرایه استفاده می‌کند تا بتوان آنها را به طور موثر مرتب کرد. هرم نوع خاصی از ساختار داده مبتنی بر درخت است وگره والد همیشه بیشتر از (یا کمتر از) گره های فرزند خود است.

ایده اصلی مرتب سازی هرمی این است که ابتدا یک هرم از آرایه مرتب نشده ساخته می شود. سپس الگوریتم به طور مکرر بزرگترین (یا کوچکترین) عنصر را از هرم حذف می کند، آن را در انتهای (یا ابتدای) آرایه مرتب شده قرار می دهد و سپس عناصر باقیمانده را مجدداً برای حفظ خاصیت هرمی پر می کند. این روند تا مرتب شدن کل آرایه ادامه می یابد. پیچیدگی زمانی مرتب ‌سازی هرمی O(n log n) است.

**مراحل الگوریتم مرتب سازی هرمی:**

**1. ساخت هرم:**

کار با ساختن یک هرم از آرایه مرتب نشده شروع می شود. این کار با تقسیم مکرر آرایه به زیرآرایه های کوچکتر و سپس انتخاب بزرگترین (یا کوچکترین) عنصر از هر زیرآرایه برای تشکیل ریشه یک پشته جدید انجام می شود. این روند تا زمانی ادامه می یابد که کل آرایه به عناصر جداگانه تقسیم شود، در این مرحله الگوریتم به مرحله بعدی می رود.

**2. انباشته کردن:**

هنگامی که هرم ساخته شد، الگوریتم به طور مکرر بزرگترین (یا کوچکترین) عنصر را از هرم حذف می کند، آن را در انتهای (یا ابتدای) آرایه مرتب شده قرار می دهد و سپس عناصر باقی مانده را مجدداً برای حفظ هرم مجدد به صورت هرمی انباشت می کند. این روند تا مرتب شدن کل آرایه ادامه می یابد.

**3. دوباره انباشته کردن:**

پس از حذف هر عنصر از هرم، الگوریتم برای حفظ خاصیت هرمی، عناصر باقی مانده را دوباره انباشت می کند. این شامل تقسیم مکرر عناصر باقیمانده به زیرآرایه های کوچکتر و سپس انتخاب بزرگترین (یا کوچکترین) عنصر از هر زیرآرایه برای تشکیل ریشه یک هرم جدید است. این روند تا مرتب شدن کل آرایه ادامه می یابد. الگوریتم مراحل 2 و 3 را تکرار می کند تا کل آرایه مرتب شود.

پیچیدگی زمانی مرتب‌سازی پشته O(n log n) است که n تعداد عناصر آرایه است. این به این دلیل است که الگوریتم تکرارهای log n مراحل heapify و reheapify را انجام می دهد و هر تکرار شامل تقسیم آرایه به زیرآرایه های کوچکتر و انتخاب بزرگترین (یا کوچکترین) عنصر از هر زیرآرایه است که زمان O(n) را می گیرد.

مرتب‌سازی پشته یک الگوریتم مرتب‌سازی پایدار است، به این معنی که ترتیب عناصر برابر حفظ می‌شود. همچنین اجرای آن نسبتاً ساده است و از نظر حافظه اضافی مورد نیاز برای انجام مرتب سازی، سربار کمی دارد. با این حال، می‌تواند کندتر از سایر الگوریتم‌های مرتب‌سازی برای آرایه‌های کوچک باشد، و نسبت به الگوریتم‌های دیگر برای مرتب‌سازی آرایه‌هایی با تعداد زیادی عناصر تکراری، کارایی کمتری دارد.