**بخش ۱: مفاهیم پایه و مقدمه**

1. **مقدمه‌ای بر ساختمان داده و الگوریتم**
   * تعریف ساختمان داده و الگوریتم
   * اهمیت ساختمان داده‌ها و الگوریتم‌ها در مهندسی نرم‌افزار
   * تأثیر انتخاب صحیح ساختمان داده و الگوریتم بر کارایی سیستم
2. **تحلیل پیچیدگی زمانی و فضایی**
   * رابطه‌ی پیچیدگی زمانی و انتخاب ساختمان داده

**بخش ۲: ساختمان داده‌ها و الگوریتم‌های پایه**

۳. **آرایه‌ها و لیست‌های پیوندی**

* تعریف، عملیات و مقایسه
* الگوریتم‌های جستجو در آرایه و لیست پیوندی
* تحلیل کارایی در سیستم‌های مقیاس‌پذیر

1. **پشته و صف**
   * پیاده‌سازی و کاربردها
   * مقایسه روش‌های ذخیره‌سازی (آرایه‌ای vs لیست پیوندی)
   * استفاده در طراحی سیستم‌های توزیع‌شده

**بخش ۳: الگوریتم‌های جستجو و مرتب‌سازی**  
5. **جستجو**

* جستجوی خطی، جستجوی دودویی، جستجوی پرش (Jump Search)
* تأثیر انتخاب ساختمان داده در سرعت جستجو

1. **مرتب‌سازی**
   * Bubble Sort, Selection Sort, Merge Sort, Quick Sort
   * مقایسه‌ی روش‌ها در پردازش داده‌های حجیم

**بخش ۴: ساختمان داده‌های پیشرفته و کاربردها**  
7. **درخت‌ها**

* درخت‌های دودویی، درخت‌های AVL، درخت‌های B و B+
* استفاده در پایگاه داده‌ها و سیستم‌های کش

1. **گراف‌ها**
   * نمایش گراف‌ها (ماتریس مجاورت، لیست مجاورت)
   * الگوریتم‌های پیمایش (DFS, BFS)
   * کاربرد در شبکه‌های کامپیوتری و گراف‌های اجتماعی
2. **جداول هش**
   * مفهوم هش، توابع هش، برخوردها
   * کاربرد در سیستم‌های امنیتی و ذخیره‌سازی کلید-مقدار

**بخش ۵: الگوریتم‌های بهینه‌سازی و برنامه‌نویسی پویا**  
10. **برنامه‌نویسی پویا (Dynamic Programming)**  
- تکنیک‌های Memoization و Tabulation  
- مثال‌هایی از دنیای واقعی

1. **الگوریتم‌های حریصانه (Greedy Algorithms)**
   * الگوریتم‌های بهینه‌سازی مثل Huffman Coding
   * تأثیر این الگوریتم‌ها در طراحی سیستم

**بخش ۶: الگوریتم‌های گراف و سیستم‌های مقیاس‌پذیر**  
12. **الگوریتم‌های کوتاه‌ترین مسیر**  
- Dijkstra, Bellman-Ford, Floyd-Warshall  
- استفاده در مسیریابی شبکه و سیستم‌های توصیه‌گر

1. **الگوریتم‌های پوشا و جریان شبکه**
   * Kruskal, Prim
   * الگوریتم‌های مکس-فلو در طراحی سیستم‌های بزرگ

**بخش ۷: الگوریتم‌ها و ساختمان داده‌ها در طراحی سیستم**  
14. **ساختمان داده‌ها و معماری سیستم**  
- انتخاب ساختمان داده مناسب برای معماری‌های مختلف  
- طراحی بهینه کش (Cache Design)

1. **ساختمان داده‌ها و امنیت نرم‌افزار**
   * توابع هش در امنیت
   * استفاده از درخت‌ها و گراف‌ها در تحلیل داده‌های امنیتی

**بخش ۸: پروژه‌های عملی و جمع‌بندی**  
16. **تمرین‌ها و پروژه‌های واقعی**  
- پیاده‌سازی الگوریتم‌های مطرح  
- چالش‌های بهینه‌سازی

**بخش ۱: مفاهیم پایه و مقدمه**

**مقدمه‌ای بر ساختمان داده و الگوریتم**

تعریف ساختمان داده و الگوریتم:

**ساختمان داده** به روش‌های مختلفی اطلاق می‌شود که برای ذخیره‌سازی، سازمان‌دهی و مدیریت داده‌ها در حافظه رایانه‌ای استفاده می‌شوند. این ساختارها به‌طور مستقیم بر کارایی برنامه‌ها و سیستم‌ها تأثیر می‌گذارند. به عبارت دیگر، انتخاب ساختمان داده مناسب می‌تواند عملکرد یک برنامه را به طور چشمگیری بهبود بخشد یا برعکس، استفاده از ساختمان داده نادرست می‌تواند موجب کاهش سرعت و کارایی سیستم شود.

ساختمان داده‌ها می‌توانند ساده یا پیچیده باشند و می‌توانند شامل انواع مختلفی همچون آرایه‌ها، لیست‌های پیوندی، پشته‌ها، صف‌ها، درخت‌ها، گراف‌ها و جداول هش باشند. این ساختارها اساساً به دو دسته تقسیم می‌شوند: **ساختمان داده‌های خطی** مانند آرایه‌ها و لیست‌ها که داده‌ها به‌صورت متوالی ذخیره می‌شوند، و **ساختمان داده‌های غیرخطی** مانند درخت‌ها و گراف‌ها که پیچیدگی‌های بیشتری دارند و برای مدل‌سازی روابط پیچیده‌تر بین داده‌ها استفاده می‌شوند.

**الگوریتم** به مجموعه‌ای از دستورات و مراحل برای حل یک مسئله خاص گفته می‌شود. الگوریتم‌ها معمولاً به‌صورت گام به گام، واضح و دقیق نوشته می‌شوند تا بتوانند یک ورودی خاص را به خروجی مطلوب تبدیل کنند. در عمل، الگوریتم‌ها از ساختمان داده‌ها برای پردازش داده‌ها و رسیدن به نتیجه مطلوب استفاده می‌کنند.

الگوریتم‌ها و ساختمان داده‌ها در کنار هم عمل می‌کنند تا عملکرد بهینه‌ای را در برنامه‌ها و سیستم‌ها ایجاد کنند. برای مثال، یک الگوریتم جستجو نیاز به انتخاب مناسب‌ترین ساختمان داده برای ذخیره‌سازی داده‌ها دارد تا بتواند جستجوی سریع و مؤثری انجام دهد. از این رو، درک عمیق این مفاهیم برای طراحی سیستم‌های نرم‌افزاری کارآمد ضروری است.

اهمیت ساختمان داده‌ها و الگوریتم‌ها در مهندسی نرم‌افزار:

ساختمان داده‌ها و الگوریتم‌ها پایه‌های اساسی مهندسی نرم‌افزار هستند و تأثیر مستقیمی بر کارایی، مقیاس‌پذیری و امنیت سیستم‌ها دارند. در این بخش به دلایل اهمیت این مفاهیم پرداخته می‌شود.

**۱. بهینه‌سازی عملکرد سیستم‌ها**

یکی از چالش‌های اصلی در توسعه نرم‌افزار، افزایش سرعت پردازش و کاهش مصرف منابع است. انتخاب صحیح ساختمان داده می‌تواند باعث بهبود چشمگیر عملکرد شود. به عنوان مثال:

جستجو در لیست‌های ساده زمان بیشتری نیاز دارد، در حالی که استفاده از جداول هش سرعت دسترسی را افزایش می‌دهد.

برخی از روش‌های مرتب‌سازی در مجموعه‌های کوچک کارآمد هستند، اما برای داده‌های حجیم بهتر است از روش‌های سریع‌تر استفاده شود.

اگر یک سیستم دارای تعداد زیادی کاربر باشد، انتخاب نادرست ساختار داده می‌تواند منجر به کاهش سرعت پاسخگویی و تجربه نامطلوب برای کاربران شود.

**۲. مدیریت بهینه حافظه و پردازنده**

هر ساختمان داده میزان مشخصی از حافظه و پردازنده را مصرف می‌کند. تصمیم‌گیری آگاهانه در مورد این موضوع می‌تواند مانع از مصرف بیهوده منابع شود. برای مثال:

برخی ساختمان داده‌ها به فضای پیوسته در حافظه نیاز دارند، اما سرعت دسترسی بالاتری دارند.

برخی دیگر پراکندگی بیشتری در حافظه دارند، اما امکان درج و حذف داده‌ها را ساده‌تر می‌کنند.

در سیستم‌های توزیع‌شده، استفاده بهینه از حافظه و پردازشگر اهمیت بیشتری پیدا می‌کند و انتخاب یک ساختمان داده نامناسب می‌تواند فشار زیادی به منابع وارد کند.

**۳. مقیاس‌پذیری در سیستم‌های بزرگ**

در سیستم‌های مدرن، حجم داده‌ها به‌سرعت افزایش می‌یابد. اگر از ساختارهای مناسبی برای مدیریت داده‌ها استفاده نشود، عملکرد سیستم دچار مشکل می‌شود. به عنوان مثال:

برخی از ساختارهای داده‌ای در پایگاه داده‌ها برای جستجو و ذخیره اطلاعات بهینه شده‌اند.

در برخی از سرویس‌های ابری، روش‌هایی برای کاهش درخواست‌های اضافی و افزایش سرعت پردازش وجود دارد.

**۴. امنیت و قابلیت اطمینان**

برخی از تهدیدهای امنیتی مستقیماً به نحوه پیاده‌سازی ساختمان داده‌ها و الگوریتم‌ها مرتبط هستند. برای مثال:

در برخی روش‌های ذخیره‌سازی اطلاعات، اگر مدیریت درستی روی نحوه دسترسی به داده‌ها انجام نشود، احتمال دسترسی‌های غیرمجاز افزایش می‌یابد.

در برخی از سیستم‌های رمزگذاری، استفاده از توابع هش مناسب می‌تواند امنیت اطلاعات را افزایش دهد.

یک مهندس نرم‌افزار آگاه با انتخاب صحیح این مفاهیم، می‌تواند احتمال بروز آسیب‌پذیری‌های امنیتی را کاهش دهد.

**۵. خوانایی و نگهداشت‌پذیری کد**

برنامه‌هایی که از ساختمان داده‌ها و الگوریتم‌های مناسب استفاده می‌کنند، خواناتر و نگهداشت‌پذیرتر خواهند بود. به عنوان مثال: استفاده از یک ساختار کلید-مقدار به جای لیست‌های ساده، باعث می‌شود داده‌ها سریع‌تر و واضح‌تر مدیریت شوند. در سیستم‌های تحلیلی، استفاده از گراف‌ها می‌تواند پردازش روابط میان داده‌ها را ساده‌تر کند.

تأثیر انتخاب صحیح ساختمان داده و الگوریتم بر کارایی سیستم:

انتخاب مناسب ساختمان داده و الگوریتم تأثیر مستقیمی بر سرعت، کارایی، مصرف منابع و مقیاس‌پذیری سیستم‌های نرم‌افزاری دارد. در این بخش، به بررسی این تأثیرات در سناریوهای واقعی پرداخته می‌شود.

۱. کاهش زمان پردازش و افزایش سرعت اجرا

یک انتخاب مناسب می‌تواند زمان اجرای عملیات را از چند ثانیه به چند میلی‌ثانیه کاهش دهد. برای مثال:

**جستجو در داده‌ها**

اگر یک لیست ساده داشته باشیم، جستجو در آن نیازمند پیمایش کل عناصر است. اما استفاده از جداول هش باعث می‌شود که دسترسی به داده‌ها در زمان ثابتی انجام شود.

در بانک‌های اطلاعاتی، اگر داده‌ها درخت‌بندی نشده باشند، بازیابی اطلاعات به‌شدت کند خواهد بود. درحالی‌که استفاده از درخت‌های B+ باعث افزایش چشمگیر سرعت جستجو می‌شود.

**مرتب‌سازی داده‌ها**

مرتب‌سازی ساده‌ای مانند Bubble Sort در حجم داده‌های زیاد بسیار کند است، در حالی که Quick Sort یا Merge Sort کارایی بالاتری دارند.

در سیستم‌هایی که نیاز به مرتب‌سازی هم‌زمان داده‌ها دارند (مثلاً موتورهای جستجو)، استفاده از الگوریتم‌های کارآمدتر باعث افزایش بهره‌وری سرورها می‌شود.

**۲. بهینه‌سازی مصرف حافظه و منابع سیستم**

برخی ساختمان داده‌ها فضای زیادی اشغال می‌کنند، در حالی که برخی دیگر بهینه‌تر هستند. انتخاب درست باعث کاهش مصرف حافظه و پردازنده می‌شود.

**مثال: ذخیره‌سازی داده‌های حجیم**

یک آرایه ثابت ممکن است فضای زیادی اشغال کند، حتی اگر بیشتر خانه‌های آن خالی باشد. درحالی‌که استفاده از لیست‌های پیوندی یا داده‌ساختارهای فشرده باعث کاهش مصرف حافظه می‌شود.

در سیستم‌های پردازش داده‌های کلان (Big Data)، اگر داده‌ها به‌درستی ایندکس نشده باشند، درخواست‌های پردازشی فشار زیادی به منابع وارد می‌کنند.

**مثال: انتخاب نوع ذخیره‌سازی**

یک سیستم گزارش‌گیری لحظه‌ای (Real-Time) نیازمند دسترسی سریع به داده‌ها است، بنابراین باید از حافظه‌ی نهان (Cache) یا ساختمان داده‌هایی مانند Trie برای کاهش زمان جستجو استفاده کند.

در یک سرویس پیام‌رسان، ذخیره پیام‌ها در صف‌های بهینه‌شده مانند **Redis List** باعث پردازش سریع‌تر و کاهش تأخیر می‌شود.

**۳. مقیاس‌پذیری سیستم‌های نرم‌افزاری**

ساختمان داده‌های نامناسب باعث می‌شود که یک سیستم در حجم کم به‌خوبی کار کند، اما در حجم زیاد دچار افت عملکرد شود.

**مثال: پردازش درخواست‌های کاربران در سرورهای مقیاس‌پذیر**

اگر یک وب‌سایت از ساختمان داده نامناسبی برای مدیریت کاربران فعال استفاده کند، ممکن است با افزایش تعداد کاربران، زمان پاسخگویی سرور افزایش یابد.

در یک فروشگاه اینترنتی، انتخاب یک سیستم ذخیره‌سازی کارآمد برای مدیریت محصولات و سفارشات می‌تواند از مشکلات کارایی در روزهای پرترافیک جلوگیری کند.

**مثال: شبکه‌های اجتماعی و مدیریت ارتباطات کاربران**

در یک شبکه اجتماعی که کاربران دنبال‌کننده‌های زیادی دارند، استفاده از **گراف‌ها** برای نمایش ارتباطات باعث پردازش سریع‌تر درخواست‌های کاربران می‌شود.

اگر قرار باشد هر بار از پایگاه داده برای جستجوی دوستان مشترک استفاده شود، سرعت سیستم به‌شدت کاهش می‌یابد. اما استفاده از **هش‌مپ‌ها** یا **ساختمان داده‌های خاص گرافی** باعث بهینه‌سازی فرآیند می‌شود.

**۴. افزایش امنیت و کاهش آسیب‌پذیری‌ها**

انتخاب نادرست ساختمان داده و الگوریتم ممکن است به مشکلات امنیتی منجر شود.

**مثال: ضعف در ذخیره‌سازی اطلاعات حساس**

اگر یک سیستم احراز هویت، گذرواژه‌ها را به‌صورت متنی ذخیره کند، امنیت آن بسیار پایین خواهد بود. اما استفاده از **توابع هش** مانند SHA-256 باعث افزایش امنیت داده‌ها می‌شود.

**مثال: حملات مرتبط با کارایی و محاسبات**

در حملات **Denial of Service (DoS)**، مهاجمان تلاش می‌کنند سیستم را با درخواست‌های زیاد از کار بیندازند. اگر ساختمان داده‌ها و الگوریتم‌ها بهینه نباشند، این حملات راحت‌تر سیستم را مختل می‌کنند.

**۵. خوانایی، نگهداشت‌پذیری و توسعه‌پذیری کد**

استفاده از ساختمان داده‌های مناسب باعث می‌شود که کدها خواناتر و قابل نگهداشت باشند.

**مثال: مدیریت کاربران در یک سیستم بزرگ**

اگر اطلاعات کاربران در یک لیست ساده ذخیره شود، افزایش تعداد کاربران باعث کاهش سرعت جستجو و حذف کاربران می‌شود. اما استفاده از **ساختار درختی یا هش‌مپ** باعث بهینه شدن مدیریت کاربران می‌شود.

برنامه‌هایی که از ساختمان داده‌های مناسب استفاده می‌کنند، برای توسعه‌های آینده آماده‌تر هستند و تغییرات در آن‌ها آسان‌تر اعمال می‌شود.

**جمع‌بندی**

انتخاب صحیح ساختمان داده و الگوریتم، تأثیر مستقیمی بر عملکرد، مقیاس‌پذیری و امنیت سیستم‌های نرم‌افزاری دارد. استفاده از یک ساختار داده نامناسب می‌تواند منجر به کاهش سرعت پردازش، افزایش مصرف حافظه و حتی آسیب‌پذیری‌های امنیتی شود، در حالی که انتخاب بهینه می‌تواند کارایی سیستم را بهبود ببخشد، هزینه‌های پردازشی را کاهش دهد و امکان مدیریت داده‌ها را در حجم‌های بالا فراهم کند.

در سیستم‌های نرم‌افزاری مدرن که حجم داده‌ها به‌طور مداوم در حال افزایش است، انتخاب الگوریتم‌های مناسب برای جستجو، مرتب‌سازی و پردازش داده‌ها، نقشی حیاتی در کاهش زمان پردازش و استفاده بهینه از منابع سخت‌افزاری ایفا می‌کند. علاوه بر این، استفاده از روش‌های مناسب ذخیره‌سازی و پردازش داده‌ها در سیستم‌های توزیع‌شده، پایگاه‌های داده و سرویس‌های بلادرنگ، می‌تواند تجربه کاربری بهتری را تضمین کند.

در نهایت، آگاهی از مزایا و معایب هر ساختمان داده و الگوریتم، به توسعه‌دهندگان کمک می‌کند تا راهکارهای بهینه‌تری را برای طراحی و پیاده‌سازی سیستم‌های نرم‌افزاری اتخاذ کنند. درک عمیق این مفاهیم، نه‌تنها باعث بهبود کارایی سیستم‌ها می‌شود، بلکه امنیت، قابلیت نگهداشت و توسعه‌پذیری آن‌ها را نیز افزایش می‌دهد.

**تحلیل پیچیدگی زمانی و فضایی**

**وقتی یک برنامه یا الگوریتم را اجرا می‌کنیم، دو چیز مهم است: چقدر زمان می‌برد؟ و چقدر حافظه مصرف می‌کند؟ این دو عامل به ترتیب به پیچیدگی زمانی (Time Complexity) و پیچیدگی فضایی (Space Complexity) معروف هستند.**

**پیچیدگی زمانی (Time Complexity)**

**پیچیدگی زمانی یعنی بررسی کنیم که الگوریتم چقدر طول می‌کشد تا اجرا شود، مخصوصاً وقتی ورودی بزرگ‌تر می‌شود. برای مقایسه الگوریتم‌ها از نماد O (Big O) استفاده می‌کنیم که نشان می‌دهد در بدترین حالت، زمان اجرا چگونه رشد می‌کند.**

**مثال‌ها:**

**اگر یک لیست را بگردیم تا یک عدد خاص را پیدا کنیم، در بدترین حالت باید کل لیست را بررسی کنیم → O(n)**

**اگر الگوریتمی داشته باشیم که همیشه در یک زمان ثابت اجرا شود (مثلاً یک مقدار خاص را از یک آرایه با ایندکس مشخص بخوانیم)**

**→ O(1)**

**در الگوریتم‌های مرتب‌سازی مثل Merge Sort که داده‌ها را به دو قسمت تقسیم و مرتب می‌کنند، پیچیدگی زمانی O(n log n) است.**

**پیچیدگی فضایی (Space Complexity)**

**پیچیدگی فضایی یعنی بررسی کنیم که الگوریتم چقدر حافظه مصرف می‌کند. بعضی الگوریتم‌ها علاوه بر داده‌های ورودی، به فضای اضافی هم نیاز دارند.**

**مثال‌ها:**

**اگر فقط یک متغیر اضافی برای ذخیره نتیجه داشته باشیم → O(1) (فضای ثابت)**

**اگر یک آرایه جدید برای ذخیره اطلاعات ساخته شود → O(n)**

**اگر در هر مرحله الگوریتم نیاز به ذخیره چندین نسخه از داده‌ها داشته باشد، ممکن است O(n²) یا بیشتر شود.**

**چرا این‌ها مهم هستند؟**

**اگر الگوریتمی زمان زیادی ببرد، روی داده‌های بزرگ کند خواهد شد.  
اگر حافظه زیادی مصرف کند، ممکن است باعث از کار افتادن برنامه شود.  
همیشه باید تعادل بین زمان اجرا و مصرف حافظه برقرار باشد تا بهترین عملکرد را داشته باشیم.**

**نکته: معمولاً الگوریتم‌های سریع‌تر، حافظه بیشتری مصرف می‌کنند و الگوریتم‌هایی که کمتر از حافظه استفاده می‌کنند، ممکن است کندتر باشند. پس بسته به نیاز باید بهترین گزینه را انتخاب کرد.**

رابطه‌ی پیچیدگی زمانی و انتخاب ساختمان داده

انتخاب ساختمان داده‌ی مناسب تأثیر مستقیمی روی کارایی الگوریتم‌ها دارد. هر ساختمان داده‌ای مزایا و معایب خودش را دارد و بسته به نیاز، انتخاب درست می‌تواند **زمان اجرا** و **مصرف حافظه** را بهینه کند.

**۱. آرایه (Array)**

**پیچیدگی دسترسی:** O(1)O(1)O(1) → چون مقدار در یک ایندکس مشخص قابل دسترسی است.

**پیچیدگی جستجو:** O(n)O(n)O(n) → چون در بدترین حالت باید کل آرایه را بررسی کنیم.

**پیچیدگی درج/حذف:** O(n)O(n)O(n) → چون برای درج در وسط یا اول آرایه، نیاز به جابجایی بقیه‌ی عناصر داریم.

**مناسب برای:** وقتی نیاز به دسترسی سریع به عناصر داشته باشیم، ولی درج/حذف زیاد انجام نشود.

**۲. لیست پیوندی (Linked List)**

**پیچیدگی دسترسی:** O(n)O(n)O(n) → چون برای یافتن یک عنصر باید از ابتدا شروع کنیم.

**پیچیدگی جستجو:** O(n)O(n)O(n) → مشابه دسترسی.

**پیچیدگی درج/حذف:** O(1)O(1)O(1) → چون فقط کافی است اشاره‌گرها را تغییر دهیم.

**مناسب برای:** وقتی درج و حذف زیاد انجام می‌شود، ولی نیازی به دسترسی سریع به ایندکس خاصی نداریم.

**۳. پشته (Stack) و صف (Queue)**

**پیچیدگی دسترسی:** O(n)O(n)O(n) → چون باید کل داده‌ها را بگردیم.

**پیچیدگی درج/حذف:** O(1)O(1)O(1) → همیشه درج و حذف از انتها یا ابتدای داده انجام می‌شود.

**مناسب برای** موقعیت‌هایی که عملیات "آخرین ورودی، اولین خروجی" (LIFO) یا "اولین ورودی، اولین خروجی" (FIFO) لازم است، مثل مدیریت فراخوانی توابع یا پردازش درخواست‌ها در سیستم.

**۴. درخت‌ها (Trees)**

**پیچیدگی جستجو، درج و حذف در درخت متوازن:** O(log⁡n)O(\log n)O(logn) → چون در هر مرحله داده‌ها نصف می‌شوند.

**درخت دودویی جستجو (BST):** اگر متوازن باشد، عملکرد خوبی دارد، ولی در بدترین حالت ممکن است به O(n)O(n)O(n) برسد.

**مناسب برای:** ذخیره و جستجوی داده‌ها به‌صورت سریع، مثل دیتابیس‌ها و ساختارهای سلسله‌مراتبی.

**۵. هش‌مپ (Hash Table)**

**پیچیدگی جستجو، درج و حذف:** O(1)O(1)O(1) در بهترین حالت (اگر برخوردی کم باشد).

**در بدترین حالت:** O(n)O(n)O(n) اگر مقدار زیادی برخورد در جدول هش رخ دهد.

**مناسب برای:** جستجوی سریع داده‌ها، مانند ذخیره اطلاعات کاربران بر اساس شناسه.

**جمع‌بندی فصل**

**پیچیدگی زمانی و فضایی** از عوامل کلیدی در ارزیابی الگوریتم‌ها هستند.  
 **Big O Notation**نشان می‌دهد که الگوریتم در بدترین حالت چگونه رفتار می‌کند.  
انتخاب **ساختمان داده‌ی مناسب** تأثیر زیادی بر عملکرد الگوریتم دارد:

**آرایه‌ها** سریع برای دسترسی، ولی کند برای حذف و درج.

**لیست‌های پیوندی** عالی برای درج و حذف، ولی کند برای جستجو.

**درخت‌ها و هش‌مپ‌ها** برای جستجوی سریع داده‌ها مفید هستند.

برای بهینه‌سازی برنامه‌ها باید **الگوریتم‌های سریع‌تر** و **ساختمان داده‌های مناسب‌تر** را انتخاب کنیم تا عملکرد سیستم به حداکثر برسد.