جلسه سوم

در این تدریس من میخواهم الگوریتم merge sort را با مثال تدریس کنم و تحلیل آن را به شیوه درختی و به شیوه unfolding بیان کنم. البته ایده‌ی divide and conquer را هم بیان خواهیم کرد.

تا کنون با یک ایده آشنا شدیم و آن تکنیک رشد بود که مساله به دو بخش solution و problem تقسیم میشد و هر بار سعی می‌شد یکی از مجموعه مسائل کم و یکی به مجموعه راهکارها بیافزاییم.

ایده‌ی تقسیم و حل :

ایده آن است که مساله را آنقدر بشکنی تا بخش‌هایی که حاصل از شکاندن‌های توست آنقدر کوچک و قابل مدیریت باشد که تو با ابزارها، توان، انرژی و یا هر چیز دیگری که داری بتوانی آن‌ها را حل کنی و در نهایت با حل بخش‌ها مساله‌ی اصلی را حل کرده باشی.

در واقع این ایده در مواجه با مساله سه گام را دنبال می‌کند:

1. تقسیم : مساله به تعدادی زیر مساله که نمونه‌های کوچکتری از همان مساله هست شکسته می‌شود.
2. حل : زیر مسائل به شکل بازگشتی یا غیر بازگشتی حل می‌شود.
3. ترکیب : جواب زیر مسئله‌ها یا تولید جواب مسئله‌ی اصلی.

ایده‌ی تقسیم و حل در زمینه‌های مختلف فعالیت بشری بسیار استفاده می‌شود. در ادامه چند مثال جالب از تکنیک "تقسیم و تسخیر آورده شده است.

چند مثال: - اهمیت

1. الگوریتم‌های جستجو:
   a. الگوریتم‌های جستجویی مانند جستجوی دودویی و جستجوی ترتیبی از تکنیک "تقسیم و تسخیر" برای یافتن یک عنصر خاص در یک لیست یا آرایه استفاده می‌کنند.
   b. این الگوریتم‌ها لیست یا آرایه را به طور مکرر به دو بخش تقسیم می‌کنند تا زمانی که عنصر مورد نظر را پیدا کنند.
2. الگوریتم‌های مرتب‌سازی:
   a. الگوریتم‌های مرتب‌سازی مانند مرتب‌سازی سریع و مرتب‌سازی ادغام از تکنیک "تقسیم و تسخیر" برای مرتب کردن یک لیست یا آرایه استفاده می‌کنند.
   b. این الگوریتم‌ها لیست یا آرایه را به طور مکرر به دو بخش تقسیم می‌کنند و سپس بخش‌های مرتب‌شده را ادغام می‌کنند.
3. فشرده‌سازی داده‌ها:
   a. الگوریتم‌های فشرده‌سازی داده‌ها مانند الگوریتم لِمپل-زیو-استر (Lempel-Ziv-Storer) از تکنیک "تقسیم و تسخیر" برای فشرده‌سازی فایل‌ها استفاده می‌کنند.
   b. این الگوریتم‌ها فایل را به طور مکرر به بخش‌های کوچکتر تقسیم می‌کنند و سپس الگوهای تکراری را در این بخش‌ها پیدا می‌کنند.
4. بازی‌های استراتژیک:
   a. بازی‌های استراتژیکی مانند شطرنج و Go از تکنیک "تقسیم و تسخیر" برای شکست دادن حریف استفاده می‌کنند.
   b. بازیکنان در این بازی‌ها با تقسیم صفحه به بخش‌های کوچکتر و کنترل هر بخش به طور جداگانه، سعی می‌کنند برتری را به دست آورند.
5. برنامه‌ریزی کامپیوتری:
   a. برنامه‌نویسان از تکنیک "تقسیم و تسخیر" برای حل مسائل پیچیده برنامه‌نویسی استفاده می‌کنند.
   b. آنها با تقسیم مسئله به بخش‌های کوچکتر و قابل مدیریت‌تر و سپس حل هر بخش به طور جداگانه، می‌توانند به راه‌حل نهایی برسند.
6. مدیریت پروژه:
   a. مدیران پروژه از تکنیک "تقسیم و تسخیر" برای مدیریت پروژه‌های بزرگ و پیچیده استفاده می‌کنند.
   b. آنها با تقسیم پروژه به بخش‌های کوچکتر و وظایف قابل‌انجام، می‌توانند پروژه را به طور موثرتر مدیریت کنند.
7. یادگیری:

a. دانش‌آموزان و دانشجویان می‌توانند از تکنیک "تقسیم و تسخیر" برای یادگیری مطالب پیچیده استفاده کنند.
b. آنها با تقسیم مطالب به بخش‌های کوچکتر و قابل فهم‌تر و سپس مطالعه هر بخش به طور جداگانه، می‌توانند به طور موثرتری یاد بگیرند.

8. حل مسائل روزمره:
   a. افراد می‌توانند از تکنیک "تقسیم و تسخیر" برای حل مسائل روزمره مانند تمیز کردن خانه یا برنامه‌ریزی یک سفر استفاده کنند.
   b. با تقسیم کار به بخش‌های کوچکتر و قابل مدیریت‌تر، می‌توانند کار را به طور موثرتری انجام

تکنیک "تقسیم و تسخیر" یک ابزار قدرتمند است که می‌تواند در بسیاری از Situationen ، از جمله مسائل پیچیده، مفید باشد.

نکته:

- این فقط چند نمونه از استفاده از تکنیک "تقسیم و تسخیر" در دنیای واقعی است.
- بسیاری از موارد دیگر وجود دارد که در آنها از این تکنیک استفاده می‌شود.

الگوریتم Merge Sort

الگوریتم مرتب سازی ادغامی یا merge sort الگوریتمی هست که امروز قصد دارم بر آن تمرکز داشته باشم، آن را بفهمیم ، اجرای آن را روی یک نمونه ببینیم کد آن را درک کنیم. یادگیری این الگوریتم به ما کمک می‌کند تا با روش پیاده سازی بازگشتی آشنا شده ، یک الگوریتم توانمند مرتب‌سازی را یاد گرفته و با تحلیل‌های unfolding و درختی در پی آن بهتر روبرو شویم.

مثالی از Merge Sort

به مثال زیر توجه کنید. سه گام تقسیم، حل و ترکیب به وضوح در تصویر مشخص است.

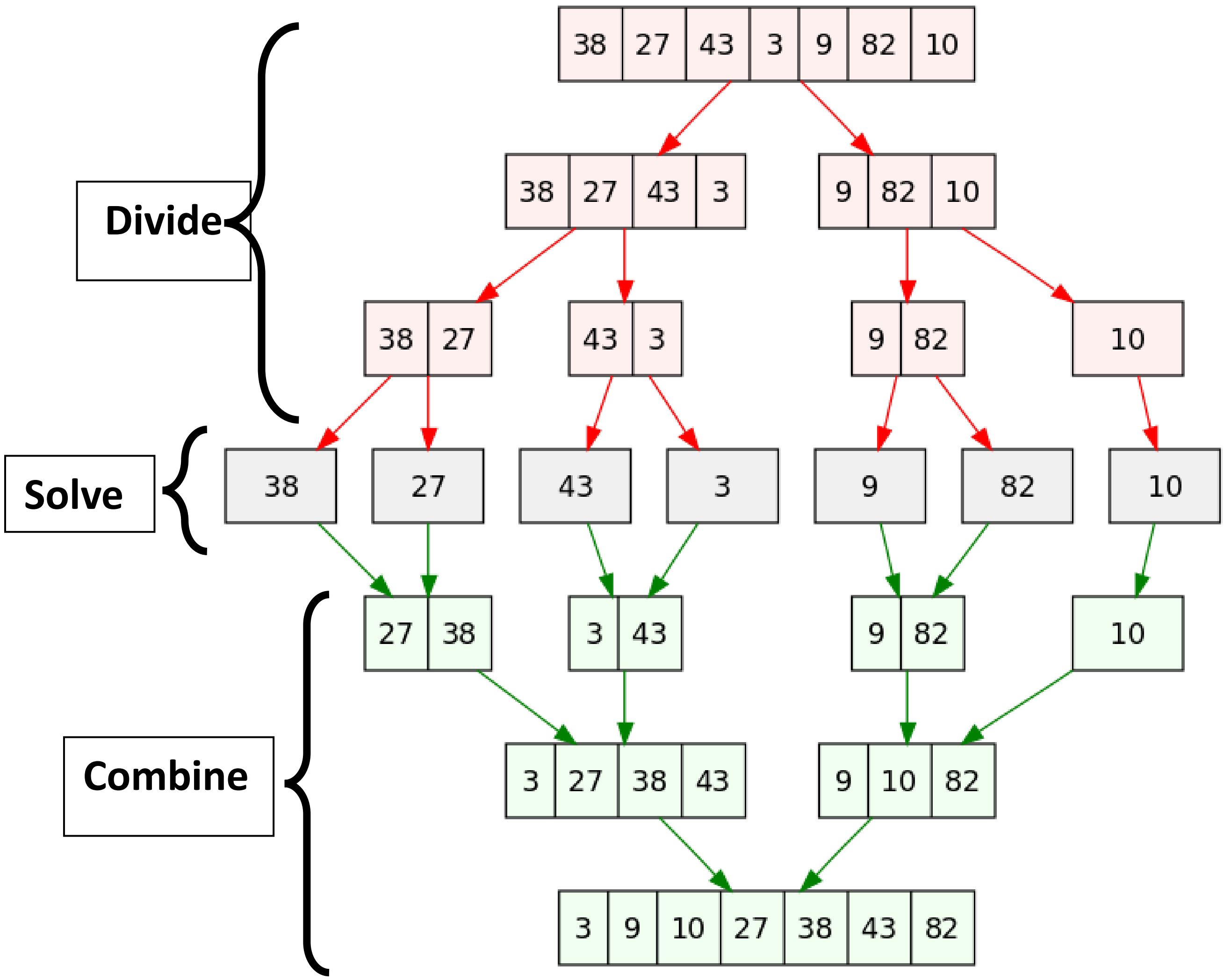
شرح کلی الگوریتم:

سه بخش به وضوح قابل رویت است بخش تقسیم، بخش حل و بخش ترکیب.

بخش تقسیم که میتوان آن را به شیوه‌ی بازگشتی پیاده‌سازی کرد آن است که هر بار مساله‌ی خود را به دو زیر مساله تقسیم کنیم و این تقسیم کردن را بارها و بارها انجام دهیم تا به ابعادی برسیم که می‌توانیم با آن ابعاد مواجه شویم، حلشان کنیم و یا خودشان حل شده باشد. گاهی اوقات آرایه‌ها اصلا در ram جا نمی‌شوند گاهی که آن آرایه را میتوانیم به ابعادی در بیاوریم که به حافظه اصلی رایانه راه یابد و بعد sort را روی آن انجام دهیم و یا مثل merge sort آنقدر مساله را به زیر مساله‌هایی تقسیم کنیم که عملا زیر مساله‌ها وارد فضای حل شوند. بعبارتی آنقدر آرایه‌ها را تقسیم کنیم که هر زیر مساله شامل یک عنصر شود و ما میدانیم یک آرایه به طول یک همواره مرتب خواهد بود.

بخش حل کد و بیان خاصی ندارد. در واقع یک شرط پایان شکستن در تقسیم هست در الگوریتم merge sort. وقتی که آرایه‌هایی به طول یک حاصل شد در واقع ما زیر مساله‌هامون پس از شکستن حل کردیم و به نوعی با یک تیر دو نشان زدیم.

به طور کلی، بخش **تقسیم و حل را برای الگوریتم merge sort میتوان به سادگی با یک تابع بازگشتی نوشت. به طوری که آرایه‌ای که داریم را دائم در بازگشت به دستمان می‌رسد را تقسیم بر دو کنیم تا حدی که به آرایه‌ای به طول یک برسیم.**



1. MERGE-SORT (A, p, r)
   a. If p<r
   b. q=$\lfloor (p+r) / 2 \rfloor$
   c. MERGE-SORT (A, p, q)
   d. MERGE-SORT (A, q+1, r)
   e. MERGE(A,p,r)

تابع ۵ خطی مرتب سازی ادغامی به ما می‌گوید مساله‌ی خود را اگر قابل شکستن است (a) وسط آن را پیدا کن (b) و اینبار آرایه را به دو نیم بشکن، بعبارتی دوباره تابع mergesort را برای همان آرایه اما اینبار از p تا q و یکبار دیگر هم از p+1 تا r فراخوانی

کن. در واقع به بیان دیگر می‌گوید mergesort برای کل آرایه برابر است با فراخوانی mergesort برای دو تا نصف همون آرایه یکی از ابتدا تا میانه و یکی از میانه تا انتها. اما چه زمان خط merge اجرا می‌شود؟ زمانیکه آرایه به اندازه‌ی یک رسیده باشد بعبارتی مسائل آنقدر کوچک شده باشند که دیگر قابل کوچکتر کردن نباشد و همچنین مساله از سویی حل شده است. در آن زمان Merge اجرا خواهد شد.

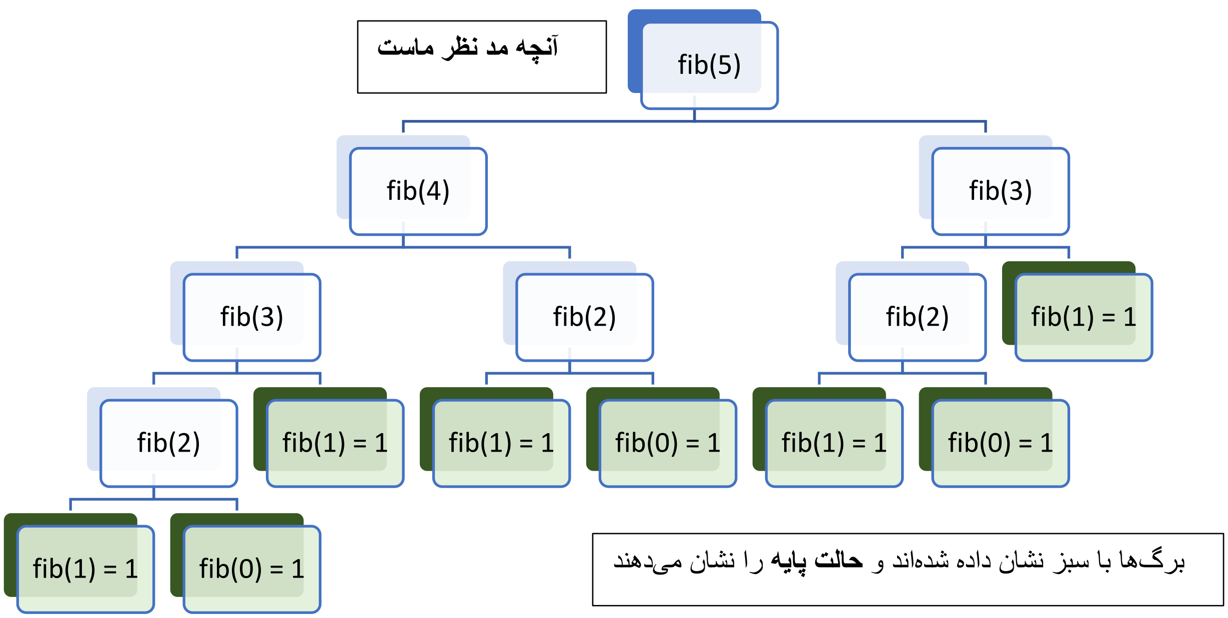
عبارت فوق را میتوان با یک رابطه‌ی ریاضی نشان داد:

$$mergesortFunction(\boldsymbol{array_n}) = 2\, mergesortFunction\left(\boldsymbol{array_{n/2}}\right) + mergeFunction\left(2\boldsymbol{array_{n/2}}\right)$$

به شکلی ساده‌تر روابط ریاضی که وابستگی تابع به خود را نشان دهد **رابطه بازگشتی** می‌گویند. برای مثال **دنباله‌ی فیبوناچی دنباله‌ای است که** می‌توان توالی این دنباله را به کمک رابطه‌ی بازگشتی نشان داد. در دنباله‌ی فیبوناچی هر جمله‌ی دنباله از مجموعه دو جمله‌ی قبلی همین دنباله کسب شده است. برای همین **رابطه‌ی بازگشتی فیبوناچی را میتوان به شکل زیر نوشت:**

$$\boldsymbol{Fibonacci(a_n) = Fibonacci(a_{n-1}) + Fibonacci(a_{n-2})}$$

از طرف جایگذاری پیاپی عبارات رابطه‌ی بازگشتی در یک دیگر ایجاد درخت می‌کند، به برگ‌های این درخت حالت پایه برای رابطه بازگشتی گفته می‌شود بزای مثال بگذارید با تابع فیبوناچی ادامه دهیم و میخواهیم جمله پنجم این عبارت را بدست آوریم.



پس چند نکته‌ی مهم را در زیر لیست می‌کنیم.

- **رابطه‌ی بازگشتی** در ریاضیات معادل توابع بازگشتی در زبان برنامه نویسی دارند
- هر رابطه‌ی بازگشتی یک **حالت پایه** دارد که همانند گام پایه استقرا برای ما مشخص است و به کمک آن محاسبات صورت می‌گیرد.
- جایگذاری پی در پی رابطه‌های **بازگشتی ایجاد شکل درخت** می‌کند. از طرفی به این جایگذاری **unfolding** می‌گوییم.

برگردیم به mergesort و نوشتن کد آن؛

خب، بعبارتی فرمول مساله اصلی شکسته شده به دو زیر مساله. اگر زیر مسائل حل شده باشد کد پایتون به خط MERGE خواهد رسید و پس از اجرای تقسیم و غلبه حال نوبت به ترکیب خواهد رسید.

اما کد merge به چه صورت است؟

ایده ترکیب بسیار ساده است. دو آرایه‌ی مرتب داریم و می‌خواهیم این دو آرایه‌ی مرتب را با هم ترکیب کنیم. بعبارتی دو تا صف دانشجو مرتب (صف قدیم) به طول n/2 داریم و میخواهیم یک صف مرتب (صف جدید)به طول n ایجاد کنیم. کافیست؛

1. **به ابتدای هر دو صف نگاه کنیم در این قسمت از صف دو تا از کوچکترین افراد هر صف قرار گرفته‌اند پس minimum این دو minimum کل عناصر خواهد بود.**
2. **عنصر کمینه را از صف قدیم خارج کنید و حال آن را وارد صف جدیدی کنید. حال بدنبال دومین کمینه هستیم که باز هم یکی از عناصر ابتدایی صف قدیم است. اولین عنصر که در صف جدید قرار گرفت و از صف قبلی خود حذف شد. حال اگر به دو صف قدیمی نگاهی بیاندازیم عناصر ابتدایی صف باز هم قدشان از بقیه عناصر در صف خودشان کوتاه تر است و در بین این دو نماینده مینیم جایگاه دوم را مشخص می‌کند.**
3. **این روال انتخاب کمینه را برای هر جایگاه در صف جدید تکرار می‌کنیم و آنقدر ادامه می‌دهیم تا اینکه همه‌ی عناصر دو صف قدیم به صف جدید انتقال یابند و یا یکی از دو صف قدیم به کلی به صف جدید انتقال یابد.**
4. **در صورتیکه یکی از صف های قدیم تهی شد، یعنی یک صف قدیم باقی مانده بود؛ آنگاه تمام عناصر صف قدیم باقیمانده را به انتهای صف جدید انتقال خواهیم داد در این صورت صف جدید شامل تمامی عناصر دو صف به طور مرتب خواهد بود.**

مثال دو آرایه به طول ۵ را با کمک الگوریتم merge از مرتب سازی ادغامی با هم ترکیب کنید.

					1	2	4	9	10
					3	5	6	7	8
									جواب نهایی
1	2	3	4	5	6	7	8	9	10

DRAW

شبه کد تابع merge به شکل زیر خواهد بود.

```
MERGE(A,p,q,r)
N1 = q-p+1
N2 = r-q
Create arrays L[1 .. n1+1] and R[1..n2+1]
For i=1 to n1:
   L[i] = A[p+i-1]
For j=1 to n2:
   R[j] = A[q+j]
L[n1+1] = ∞
R[n2+1] = ∞
i = 1
j = 1
For k = p to r :
   If L[i] <=R[j]
      A[k] = L[i]
      i = i + 1
   Else
      A[k] = R[j]
      j = j+1
```

1. تابع merge فاقد هر دستور بازگشتی است.
2. بعنوان ورودی یک آرایه گرفته و همان آرایه را با ابعادی که آن ابعاد را هم از ورودی می‌گیرد مرتب (ترکیب دو آرایه مرتب) می‌کند.
3. برای اینکار از auxiliary space یا فضای کمکی L و R استفاده می‌کند که این فضا وابسته به ورودی ما هست پس این الگوریتم not inplace هست یا غیر درجاست (بر خلاف روش insertion sort)

اما چگونه آن را تحلیل کنیم یا بعبارتی چگونه این الگوریتم را با insertion sort مقایسه کنیم؟

برای اینکار باید دو گام برداریم.

1. **تابع الگوریتم را در دنیای ریاضیات مشخص کنیم**
2. **تابع بدست آمده را متوجه شویم از چه مرتبه‌ای است یا بعبارتی به دنیای مجموعه‌ها ببریم**

و از آنجاییکه در آنجا وضعیت مرتب سازی درجی را میدانیم میتوانیم این الگوریتم را با آن مقایسه کنیم اگر خواستیم.

برای اینکار باید طبق معمول تعداد دستورات را بشماریم.

1. **تعداد دستورات دو تابع یکی merge sort و دیگری merge**
2. **Merge تابعی غیر بازگشتی است**
3. **Merge sort تابعی بازگشتی است.**

اما اینکار لازم نیست چرا که بدلیل سادگی از قبل ما توانستیم رابطه‌ی بازگشتی این الگوریتم را در دنیای ریاضیات بنویسیم اما چالش اصلی بدست آوردن مرتبه‌ی این رابطه بازگشتی است یا بعبارتی اینکه حال رابطه‌ی بازگشتی را چطور در خود دنیای ریاضیات تحلیل کنیم.

برای تحلیل روابط بازگشتی با دو تکنیک مواجه هستیم؛

1. **تکنیک UnFolding**
2. **تکنیک روش درختی**

رابطه‌ی بازگشتی mergesort به شرح زیر بود:

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + mergeEquation & n > 1 \\ 1 & n = 1 \end{cases}$$

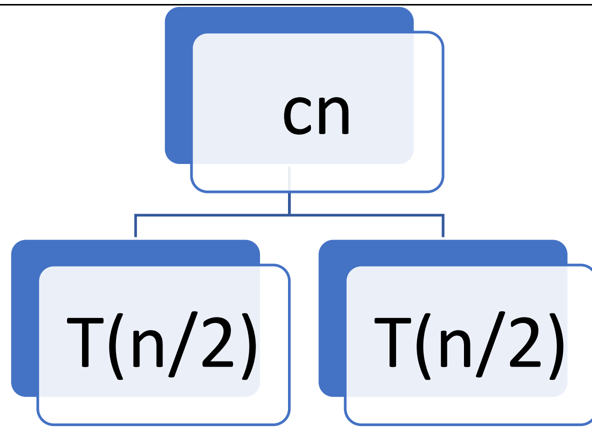
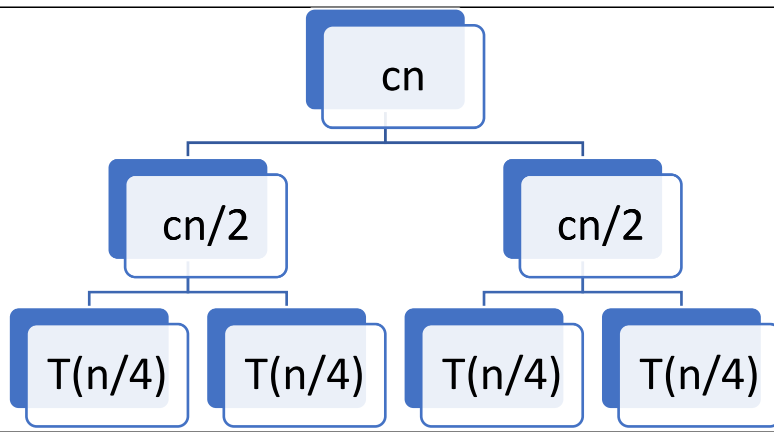
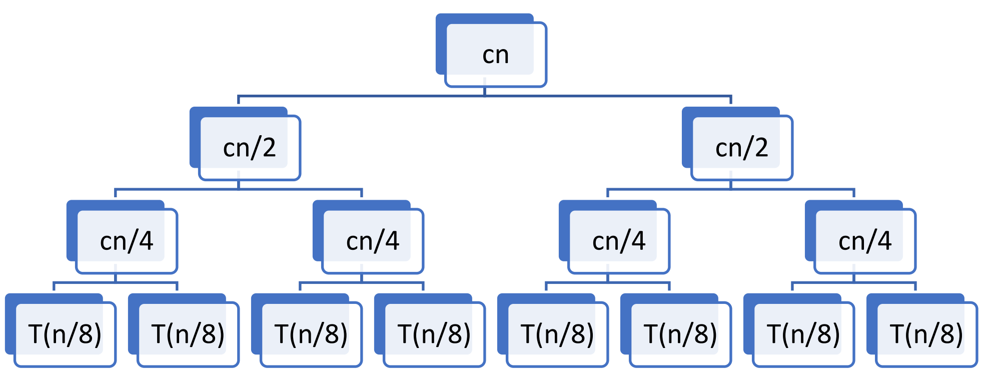
و $mergeEquation$ به سادگی هر چه تمام بدست می‌آید. برای اینکار تعداد دستورات را در بدترین حالت میشماریم.

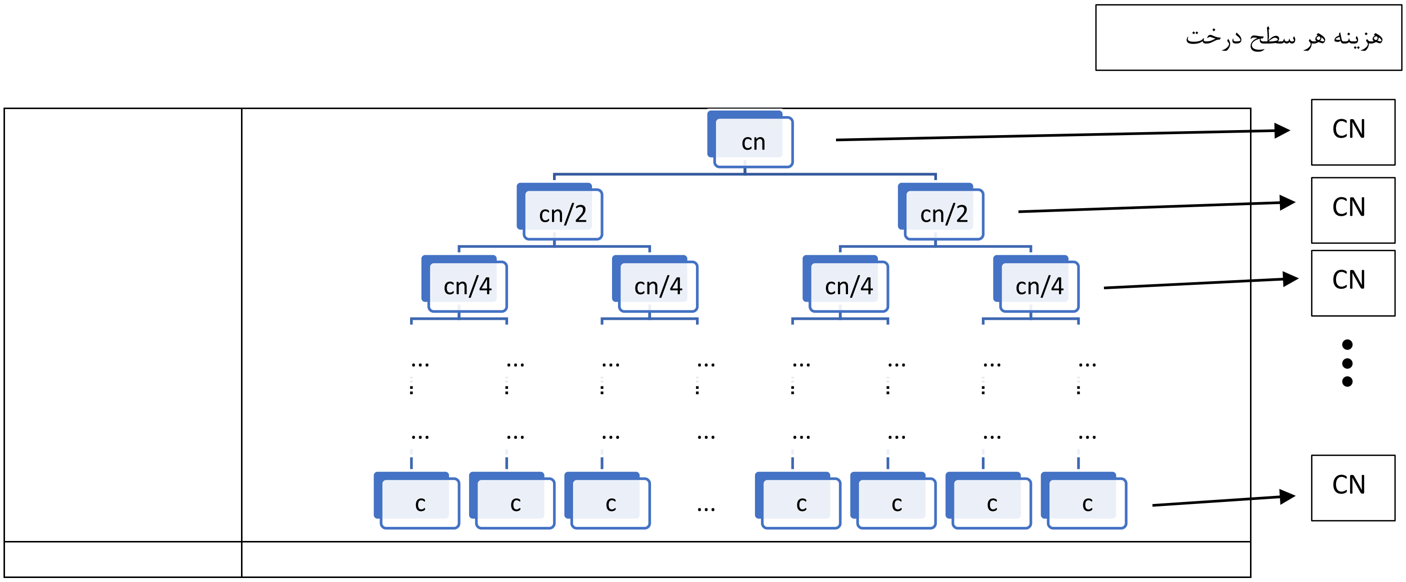
		هزینه‌ها	تعداد دستورات ؟
0	MERGE(A,p,q,r)		
1	N_1 = q-p+1	C1	1
2	N_2 = r-q	C2	1
3	Create arrays L[1 .. n_1+1] and R[1..n_2+1]	C3	1
4	For i=1 to n_1:	C4	n_1
5	L[i] = A[p+i-1]	C5	n_1
6	For j=1 to n_2:	C6	n_2
7	R[j] = A[q+j]	C7	n_2
8	L[n_1+1] = ∞	C8	1
9	R[n_2+1] = ∞	C9	1
10	i = 1	C10	1
11	j = 1	C11	1
12	For k = p to r :	C12	$n = n_1 + n_2$
13	If L[i] <=R[j]	C13	n
14	A[k] = L[i]	C14	n یا کمتر از آن
15	i = i + 1	C15	nیا کمتر از آن
16	Else	C16	n
17	A[k] = R[j]	C17	nیا کمتر از آن
18	j = j+1	C18	nیا کمتر از آن

در نهایت $mergeEquation = cn$ میرسیم و رابطه به شکل زیر در خواهد آمد ..

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + cn & n > 1 \\ 1 & n = 1 \end{cases}$$

در زیر روش درخت را خواهیم داشت. یادمان باشد با کمک روش های تحلیل بدنبال پیدا کردن مرتبه تابع هستیم.

T(n)	
	
	



حال کل هزینه مجموعه‌ی همه‌ی *CN* ها خواهد بود که تعداد این *CN* ها ارتفاع درخت است. ارتفاع درخت هم تابعی از *N* هست چرا که هر بار آرایه دارد تقسیم بر دو می‌شود. یعنی اگر *n = 1024* باشد در سطح دوم دو آرایه به طول *n = 512* خواهیم داشت و بعد ۴ آرایه به طول ۲۵۶ و تعداد این سطوح چقدر خواهد بود؟ تعداد این سطوح از **دنباله‌ی *geometric* تبعیت می‌کند.**

$$\mathbf{1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, 1}$$

ارتفاع این درخت از رابطه *logarithm* در مبنای ۲ بدست می‌آید بعبارتی:

$$= CN \times \log_2 n = O(nlogn)$$

مرتب سازی ادغامی از مرتبه‌ی $O(nlogn)$ هست.

تکنیک دیگر استفاده از *unfolding* است.

$$fn = 2f\left(\frac{n}{2}\right) + cn$$

$$f\left(\frac{n}{2}\right) = 2\, f\left(\frac{n}{2^2}\right) + \frac{cn}{2^1}$$

$$fn = 2\left(2\, f\left(\frac{n}{2^2}\right) + \frac{cn}{2^1}\right) + cn = 2^2 f\left(\frac{n}{2^2}\right) + cn + cn$$

$$f\left(\frac{n}{4}\right) = 2\, f\left(\frac{n}{2^3}\right) + \frac{cn}{2^2}$$

$$fn = 2^2 f\left(\frac{n}{2^2}\right) + cn + cn = 2^2\left\{2\, f\left(\frac{n}{2^3}\right) + \frac{cn}{2^2}\right\} + cn + cn = \ 2^3 f\left(\frac{n}{2^3}\right) + cn + cn + cn$$

$$at\ the\ step\ k\ we\ have:$$

$$f(n) = 2^k f\left(\frac{n}{2^k}\right) + cn\, k \rightarrow also\ we\ have\ f(1) = 1\ so\ if\ n$$
$$= \ 2^k\ then\ we\ can\ calculate\ final\ answers$$

$$n = \ 2^k \rightarrow \log_2 n = k$$

$$\boldsymbol{f(n) = 2^{\log_2 n} f(1) + cn\ \log_2 n \rightarrow f(n) = n + cn\ \log_2 n}$$