# تحلیل الگوریتم‌های divide-and-conquer

برای الگوریتم‌های divide-and-conquer از recurrence equation برای توصیف Runningtime استفاده می‌کنیم.

$T(n)$ = running time of a problem of size $n$.

- اگر سایز مسئله به قدر کافی کوچک باشد، (مثلاً عدد ثابت $c$ آ داشته باشیم که $n \leq c$) به این حالت base case می‌گوییم. آن را در زمان ثابت $\Theta(1)$ حل می‌کند.
- در غیر اینصورت، فرض کنید که مسئله را به $a$ زیرمسئله تقسیم کرده‌ایم که هرکدام سایزشان $\frac{1}{b}$ سایز مسئله اصلی است. (مثلاً در merge sort، $a = b = ۲$)
- زمان تقسیم یک مسئله به سایز $n$ را $D(n)$ فرض می‌کنیم.
- در این حالت، $a$ عدد زیرمسئله به سایز $\frac{1}{b}$ وجود دارد که باید آن را حل کنیم، بدیهی است که زیرمسئله‌ها $T(\frac{n}{b})$ زمان احتیاج دارند تا حل شوند ⇐ برای حل تمام زیرمسئله‌ها نیاز به $aT(\frac{n}{b})$ زمان داریم.
- فرض کنید زمان combine جوابها برابر با $C(n)$ باشد.

در این صورت recurrence زیر را داریم:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c, \\ aT(\frac{n}{b}) + D(n) + C(n) & \text{otherwise.} \end{cases}$$

# تحلیل الگوریتم merge sort

برای سادگی فرض کنیم که $n$ توانی از ۲ است، بنابراین در هر گام از تقسیم سایز هر دو زیرمسئله دقیقاً $\frac{n}{2}$ است و نیز base case زمانی اتفاق می‌افتد که $n=1$ باشد.

وقتی $n \geq 2$ باشد، گام‌های merge sort انجام می‌شود.

Divide : $D(n)=\theta(1)$ ← محاسبه q به عنوان میانگین p و r

Conquer : $2T(\frac{n}{2})$ ← $\frac{n}{2}$ حل کردن ۲ زیرمسئله هرکدام سایز recursively به صورت

Combine : $C(n)=\theta(n)$ ← $\theta(n)$ انجام MERGE بر روی $n$ تا عنصر زیرآرایه‌ها روی هم

زمان‌های Divide و Combine را با هم محاسبه می‌کنیم که $\theta(1)+\theta(n)=\theta(n)$ است.

بنابراین recurrence برای زمان اجرای merge sort به صورت زیر خواهد بود:

$$T(n)=\begin{cases}\theta(1) & \text{if } n=1\\ 2T(\frac{n}{2})+\theta(n) & \text{if } n>1\end{cases}$$

Solving the merge-sort recurrence :

روش ۱:

با استفاده از (master theorem) که اندکی بعدتر توضیح داده می‌شود:

$T(n)=\theta(n \lg n)$ $\quad (\lg n = \log_2 n)$

در مقایسه با Insertion sort که در حالت worst case است $\theta(n^2)$، merge sort سریعتر است.

(Trading a factor of n for a factor of lg n is a good deal)

- برای ورودی های کوچک insertion sort ممکن است سریعتر باشد و برای ورودی های به قدر کافی بزرگ merge sort همیشه سریعتر است،
- به دلیل آنکه running time نسبت به insertion sort نسبت ملایمتر رشد می کند.

روش ۲: حل merge-sort recurrence بدون master theorem:

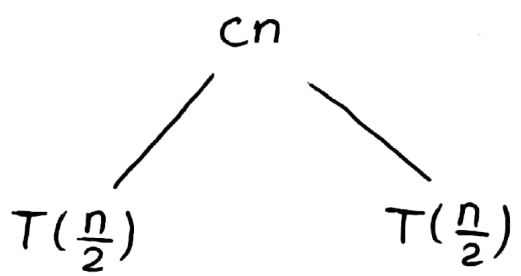
فرض کنید که c یک عدد ثابت برای running time ی حالت base case باشد (این زمان به ازای هر عضو آرایه باشد هم به ازای divide و هم به ازای conquer. (بله، ممکن است که این زمانها برابر نباشند)

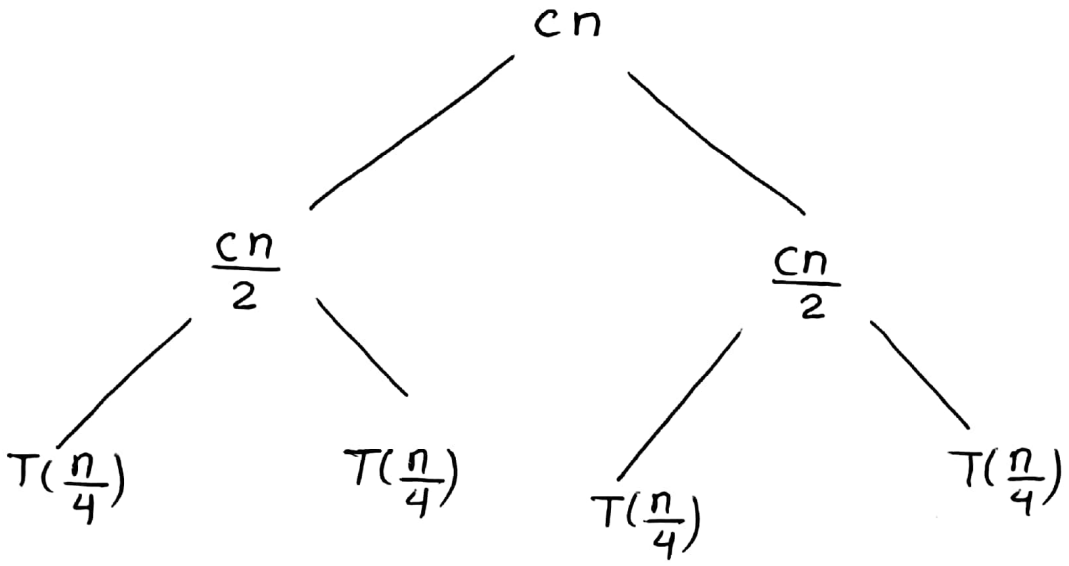
recurrence:

$$T(n) = \begin{cases} c & \text{if } n = 1, \\ 2T(\frac{n}{2}) + cn & \text{if } n > 1. \end{cases}$$

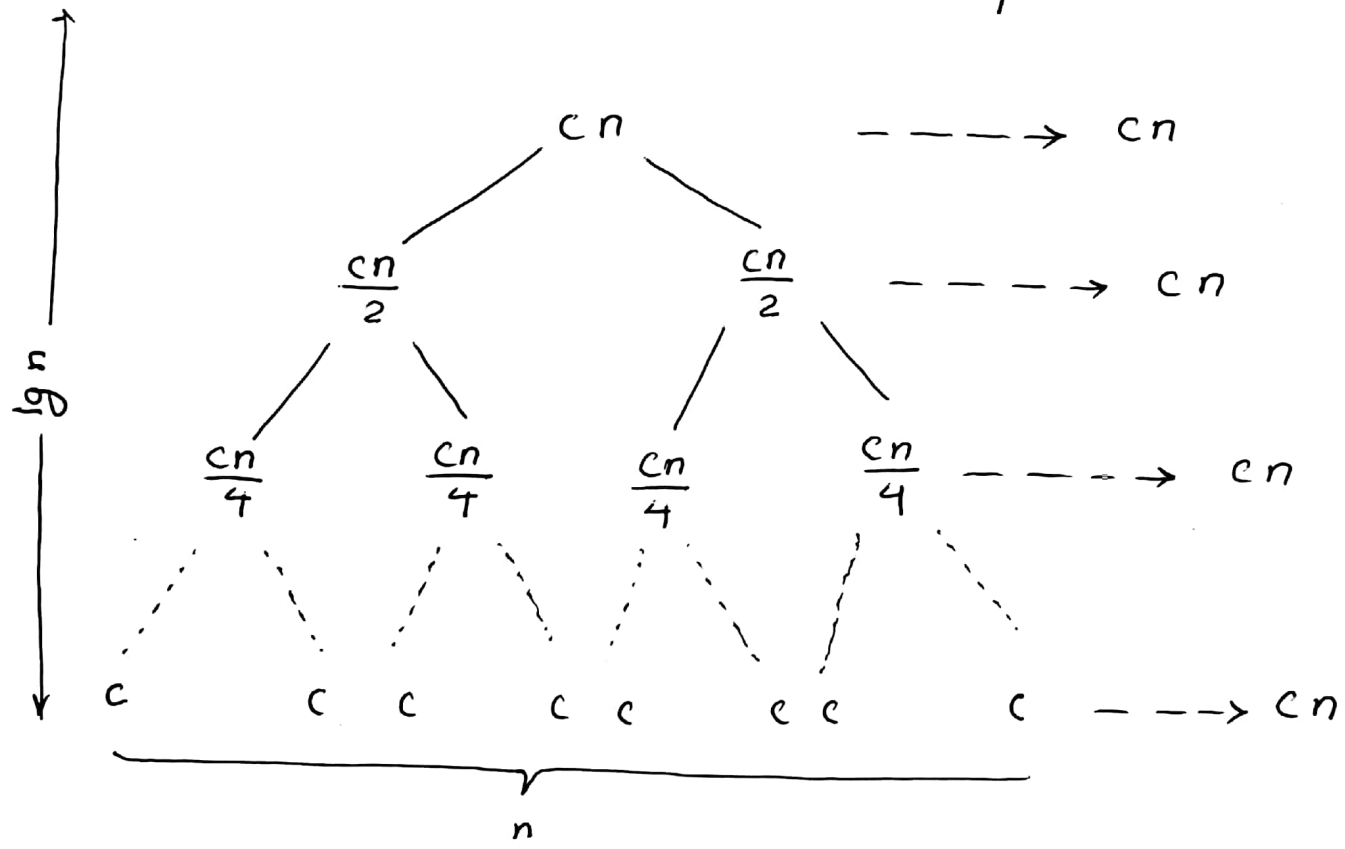
recursion tree:

رسم یک recursion tree نشان می دهد که بسط پی در پی گامهای recurrence چگونه است. برای مسئله اصلی یک هزینه cn و دو هزینه حل زیرمسئله ها وجود دارد که هرکدام $T(n/2)$ است.






این گام‌ها را ادامه می‌دهیم تا سایز مسئله به ۱ برسد.




Total: $cn\lg n + cn$

- هر مرحله هزینه $cn$ دارد.
- مرحله بالایی هزینه $cn$ دارد.
- مرحله بعدی ۲ تا زیر مسئله دارد که هرکدام $\frac{cn}{2}$ است.
- مرحله بعدتر ۴ تا زیر مسئله دارد که هرکدام $\frac{cn}{4}$ است.

به‌ازای تعداد زیر مسئله‌ها ۲ برابر و هزینه هرکدام نصف می‌شود.

- تعداد کل level ها برابر $\lg n + 1$ است.
- از induction استفاده می‌کنیم.
- حالت base case $n = 1 \Leftarrow$ 1 level & $\lg 1 + 1 = 0 + 1 = 1$
- Induction hypothesis : یک درخت برای یک مسئله به سایز $2^i$ تعداد $\lg 2^i + 1$ که همان $i + 1$ است، level دارد.
- چون فرض کردیم که سایز مسئله توانی از ۲ باشد، سایز بعدی بزرگتر از $2^i$ برابر $2^{i+1}$ است.
- یک درخت با $2^{i+1}$ یک level بیشتر دارد نسبت به درخت با سایز $2^i$ $\Leftarrow$ $i+2$ تا level دارد.
- $\lg 2^{i+1} + 1 = i + 2$ پس inductive argument تمام شده است.

- total cost :

$$cn \times \lg n + 1 = cn \lg n + cn$$

$$= \theta(n \lg n)$$