Divide & Conquer Template

Goal: with Φ some function,

compute $\Phi(x)$ *for given* x *(of size* n*)*

Method of divide & conquer: for n sufficiently big, we

1. decompose x into smaller problems, $x_1 \ldots x_k$
2. recursively compute solutions $y_1 \leftarrow \Phi(x_1) \ldots y_k \leftarrow \Phi(x_k)$
3. recombine $y_1 \ldots y_k$ into solution $y = \Phi(x)$.

We may get the recurrences of the form

$$
\begin{array}{rcll}
T(n) & = & aT(\frac{n}{b}) + f(n) & \text{for } n \geq N \\
T(n) & = & g(n) & \text{for } n < N
\end{array}
$$

- Asymptotically, T does not depend on g or N.
- Thus our focus: how to decompose/recombine
- Experiments may suggest how to handle small input.

Merge Sort

MERGE SORT follows the divide & conquer paradigm:

- decompose just cuts the array in half
- recombine merges two sorted arrays.

Complexity analysis:

- MERGE has running time in $\Theta(n)$
- MERGESORT has running time recurrence:

$$T(n) \in 2T(n/2) + \Theta(n)$$

and thus has running time in $\Theta(n \lg n)$

Quicksort

QUICKSORT also follows divide & conquer:

- decompose amounts to:
  1. choose pivot element p
  2. rearrange array into 3 parts:
     - 2.1 the elements $< p$ (to be recursively sorted)
     - 2.2 the elements $= p$ (need no further processing)
     - 2.3 the elements $> p$ (to be recursively sorted)

     this can be accomplished by Dutch National Flag
     though other algorithms may also be used
- recombine does nothing
  (just "glues" subarrays together)

Space use:

- works by array permutations so appears in-place
- but we need to account for recursion stack

The recursion stack

- could potentially grow linearly
- but only $O(\lg n)$ if we process the shortest part first

Quicksort Running Time

Crucial question: how to choose the pivot?

- Since decomposition is already in $\Theta(n)$, we can spend $O(n)$ on the choosing without affecting asymptotic complexity
- if we are fortunate to always choose the median element then we get the recurrence

  $$T(n) \in 2T(\frac{n}{2}) + \Theta(n)$$

  and thus QuickSort will run in $\Theta(n \lg n)$
- if we are unfortunate to always choose the smallest element then QuickSort will run in $\Theta(n^2)$
- but even if we always pick element such that say only 10 % are smaller, QuickSort will run in $O(n \lg(n))$
- if we pick pivot randomly the expected running time will be in $O(n \lg(n))$, as one can show

Multiply Large Integers

To multiply $\begin{array}{rcl} P & = & w2^n + y \\ Q & = & x2^n + z \end{array}$

we may apply Divide & Conquer and combine the results of multiplying smaller integers:

$$P \cdot Q = (w \cdot x)2^{2n} + (w \cdot z + y \cdot x)2^n + y \cdot z$$

for the recurrence $T(n) = 4T(\frac{n}{2}) + \Theta(n)$ and thus $T(n) \in \Theta(n^2)$. But we also have

$$P \cdot Q = (w \cdot x)2^{2n} + ((w + y) \cdot (x + z) - w \cdot x - y \cdot z)2^n + y \cdot z$$

for the recurrence $T(n) = 3T(\frac{n}{2}) + \Theta(n)$ and thus $T(n) \in \Theta(n^{\lg(3)})$. We can even get, for all $k \geq 2$,

$$T(n) = (2k - 1)T(\frac{n}{k}) + \Theta(n)$$

with solution $T(n) \in \Theta(n^{\log_k(2k-1)})$ and thus for any $q > 1$ we can get $T(n) \in o(n^q)$ by choosing k big.

Matrix Multiplication

Goal: given $n \times n$ matrices a and b (thus input size is $\Theta(n^2)$), compute their product c given by

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Straightforward algorithm:

$$\begin{array}{l}
\textbf{for } i \leftarrow 1 \textbf{ to } n \\
\quad \textbf{for } j \leftarrow 1 \textbf{ to } n \\
\quad\quad c_{ij} \leftarrow 0 \\
\quad\quad \textbf{for } k \leftarrow 1 \textbf{ to } n \\
\quad\quad\quad c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}
\end{array}$$

which runs in time $\Theta(n^3)$.

Divide & Conquer, Naive

Decompose each of a, b into 4 submatrices:

$$a = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \quad b = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

Then $c = a \times b$ can be computed as

$$\begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

and thus we get the recurrence

$$T(n) = 8T(\frac{n}{2}) + \Theta(n^2)$$

with solution $T(n) \in \Theta(n^3)$ which is no improvement.

Divide & Conquer, Clever

We need to compute

$$\begin{aligned} c_{11} &= a_{11}b_{11} + a_{12}b_{21} & c_{12} &= a_{11}b_{12} + a_{12}b_{22} \\ c_{21} &= a_{21}b_{11} + a_{22}b_{21} & c_{22} &= a_{21}b_{12} + a_{22}b_{22} \end{aligned}$$

which will be the case [Strassen 1969] if

$$\begin{aligned} c_{11} &= m_2 + m_3 & c_{12} &= m_1 + m_2 + m_5 + m_6 \\ c_{21} &= m_1 + m_2 + m_4 - m_7 & c_{22} &= m_1 + m_2 + m_4 + m_5 \end{aligned}$$

$$\begin{aligned} \text{where } m_1 &= (a_{21} + a_{22} - a_{11})(b_{22} - b_{12} + b_{11}) \\ m_2 &= a_{11}b_{11} \\ \dots & \quad \dots \\ m_7 &= a_{22}(b_{11} + b_{22} - b_{12} - b_{21}) \end{aligned}$$

and thus we get the recurrence $T(n) = 7T(\frac{n}{2}) + \Theta(n^2)$ with solution $T(n) \in \Theta(n^{\lg(7)}) \subset O(n^{2.81})$

- even cleverer schemes exist; current best: $O(n^{2.373})$
- this only pays off for really big matrices
- if matrices are sparse there are faster methods.

Selection by Divide & Conquer

To find the kth smallest key in an array, we

choose a pivot p

and then partition using DutchNationalFlag into:

- r red elements, with key $< p$
- w white elements, with key $= p$
- b blue elements, with key $> p$.

Then we do a case analysis; if

- $k \leq r$: recursively find kth smallest in red partition
- $r < k \leq r + w$: return p
- $k > r + w$: recursively find $(k - r - w)$th smallest in blue partition

If we are unfortunate to always choose the smallest element as pivot, we get recurrence

$$T(n) = T(n-1) + \Theta(n) \text{ and thus } T(n) \in \Theta(n^2)$$

Choosing the Pivot as the Median

If we always use the median as pivot, and can find it in time $M(n)$, we get recurrence

$$T(n) = 1 \cdot T(\frac{n}{2}) + \Theta(n) + M(n)$$

If we could get $M(n) \in O(n)$ we thus have

$$T(n) = T(\frac{n}{2}) + \Theta(n)$$

yielding $T(n) \in \Theta(n)$.

- But finding the median is special case of selection
- so this looks like a "chicken-and-egg" problem.

Instead, we shall go for a pseudo-median that induces a partition that is not "too" imbalanced.

Choosing the Pivot as a Pseudo-Median

We shall follow the recipe:

0. divide into chunks of 5
1. for each of the $\frac{n}{5}$ chunks, compute its median
2. compute as pivot the median of the $\frac{n}{5}$ medians
3. apply the DUTCH NATIONAL FLAG algorithm
4. recursively call the appropriate partition.

Analysis of running time $T(n)$:

1. $\frac{n}{5} \cdot \Theta(1) \in \Theta(n)$
2. $T(\frac{n}{5})$
3. $\Theta(n)$
4. $T(qn)$ with qn the largest possible partition size

for a recurrence of

$$T(n) = T(\frac{n}{5}) + T(qn) + \Theta(n)$$

Pseudo-Median Gives Linear-Time Selection

$$T(n) = T(\frac{n}{5}) + T(qn) + \Theta(n)$$

will have solution $T(n) \in \Theta(n)$ if

$$\frac{1}{5} + q < 1$$

Let $n = 5k + r$, and p the pseudo-median:

- of the k medians, at least $k/2$ are $\leq p$
- thus at least $3k/2$ elements are $\leq p$.

Hence the fraction of elements $\leq p$ is at least

$$\frac{\frac{3k}{2}}{5k + r} = \frac{3k}{10k + 2r}$$

and we see that for big k,

- at least 29 % of elements will be $\leq p$; similarly
- at least 29 % of elements will be $\geq p$.

Thus a partition contains at most 71 % of the elements, so we can pick $q = 0.71$ which works as $0.2 + 0.71 < 1$.