Divide-and-Conquer

Example: For \( n = 2^k \), fill the grid with "\( \Box \)".

Solution: Similar to mathematical induction.

Assume we know how to solve \( n' = 2^{k-1} \).

Algorithm:
1. Divide the area into 4 areas of roughly the same size. (Divide)
2. Put a "\( \Box \)" in the middle, s.t. one square is missing in each subarea. (Combine)
3. Fill each subarea recursively. (Conquer)

Mergesort

Problem: Sort \( n \) numbers from small to large. (Assume \( n = 2^k \).

Algorithm:

1. Terminate if \( n = 1 \).
2. Divide the array into two arrays of size \( \frac{n}{2} \).
3. Sort these two subarrays recursively.
4. Merge the sorted subarrays.

Overall runtime: \( T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \Theta(n) \)

In other words, \( T(n) = 2 \cdot T\left(\frac{n}{2}\right) + cn \)

\( T(n) = 2^{\log_2 n} \cdot T(1) + cn \cdot \log_2 n \)

Overall runtime = \( \Theta(n \log n) \)
Counting Inversions.

Definition: Given an array $a_1, a_2 \ldots a_n$. (all $a_i$'s are distinct)

If $i < j$ and $a_i > a_j$, we call $(a_i, a_j)$ an inversion.

Goal: Count the total number of inversions.

Example: $n=4$, $a=[4, 1, 3, 2]$.

# inversions = $3 \cdot (4, 3) + (4, 2) + (3, 2)$

A simple $O(n^2)$ algorithm:

For $i = 1$ to $n-1$

For $j = i+1$ to $n$

If $a_i > a_j$ then $inv = inv + 1$

End for

End for

Return $inv$.

An $O(n \log n)$ Solution:


$\text{# of inv. in } A = \text{# of inv. in } L + \text{# of inv. in } R + \text{# of inv. } (i, j)$ where $i \in L$ and $j \in R$.

A simple algorithm:

Merge and Count $(L, R)$:

- merge the two (sorted) lists
- output # of inv. in $L \times R$.

Note that # of inv. in $L \times R$ does not change if we shuffle elements in $L$ or $R$, so we can count this # after sorting $L, R$.

When we merge (the sorted version of) $L$ and $R$,

each element $x$ is responsible for counting the # of inversions between $x$ and every element that hasn't been processed.

Case 1: $a_i < b_j$

Append $a_i$ to $C$

Case 2: $b_j < a_i$

Append $b_j$ to $C$

# of inv. = # of inv. + (# of remaining elements in $a$)

(see page 224 of textbook).

Runtime = $O(n \log n)$

Because comparing to MergeSort, the extra work takes $O(1)$ time for each element we process in the "merge" step.