CS 255, Spring 2014, SJSU

Introduction to divide and conquer: MergeSort

Fernando Lobo

Divide and Conquer

- This lecture we analyze another algorithm that you probably already know: MergeSort
- The algorithm uses the divide and conquer paradigm.
- In reality the paradigm should be called divide, conquer, and combine.

Divide and Conquer

3 essential steps:

- Divide the problem in several subproblems.
- Conquer (solve) recursively each subproblem.
  (base case: if problem size is sufficiently small, solve it by brute force).
- Combine the solutions of the subproblems to obtain an overall solution for the original problem.

Merge Sort

```
MERGE-SORT(A, left, right)
    if left < right                          // Test base case
        mid = ⌊(left + right)/2⌋             // Divide
        MERGE-SORT(A, left, mid)             // Conquer
        MERGE-SORT(A, mid + 1, right)        // Conquer
        MERGE(A, left, mid, right)           // Combine
```

Initial call: MERGE-SORT$(A, 1, n)$

Merge operation

- **Input:** Array A and indices *left*, *mid* and *right*, such that,
  - $left \leq mid < right$.
  - subarray $A[left \mathinner{..} mid]$ is sorted.
  - subarray $A[mid + 1 \mathinner{..} right]$ is sorted.
- **Output:** array $A[left \mathinner{..} right]$ sorted.

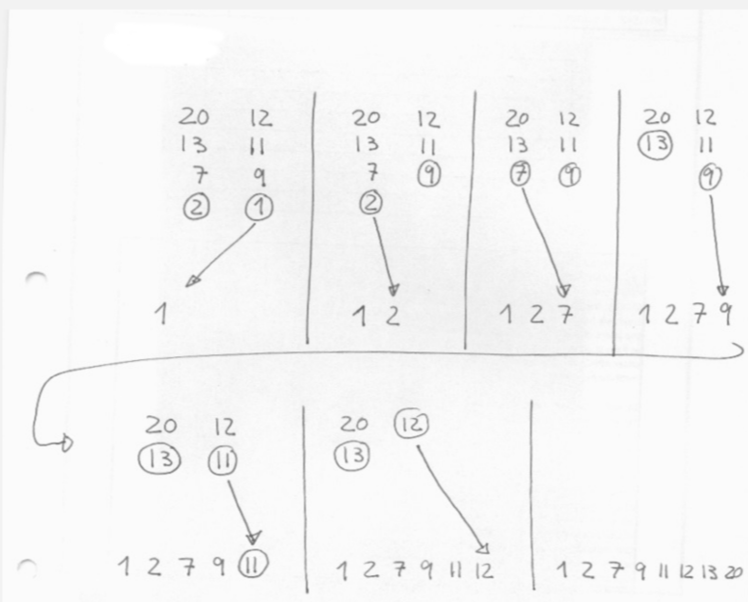
  We aim for $\Theta(n)$ complexity, with $n = right - left + 1$ being the total number of elements.

Merge in linear time

Imagine the 2 subarrays as 2 piles. At the top of each pile we always have the smallest element of each array.

- Compare the tops of the 2 subarrays and remove the smaller one.
- Repeat the previous step until one of the subarrays is empty.
- Each such step takes a constant amount of time, and we have a maximum of n steps. Therefore, this operation can be done in $\Theta(n)$.

Example



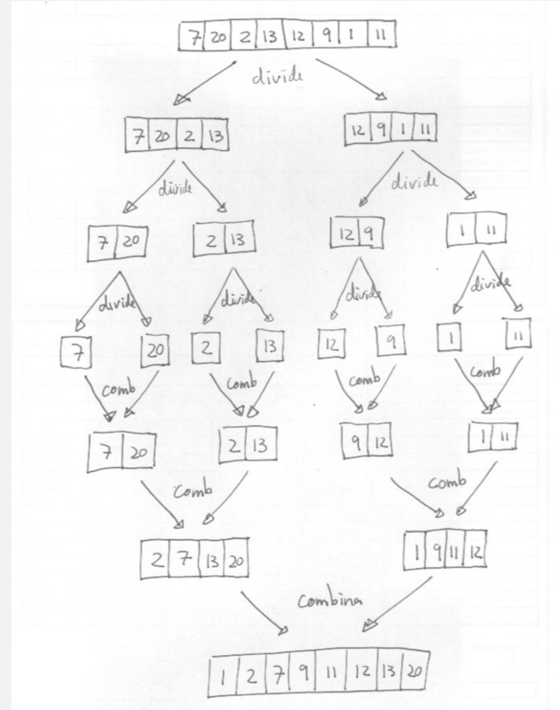
Implementation with sentinels

- Can avoid empty subarray testing by using sentinels.
- Idea: put a very large number (infinity) at the end of each subarray.

Merge pseudocode (with sentinels)

MERGE(A, *left*, *mid*, *right*)

$n1 = mid - left + 1$
$n2 = right - mid$
// Create arrays $L[1\,..\,n1+1]$ and $R[1\,..\,n2+1]$
for $i = 1$ **to** $n1$
  $L[i] = A[left + i - 1]$
for $j = 1$ **to** $n2$
  $R[j] = A[mid + j]$
$L[n1+1] = R[n2+1] = \infty$ // Sentinels
$i = j = 1$
for $k = left$ **to** $right$
  if $L[i] \leq R[j]$
    $A[k] = L[i]$
    $i = i + 1$
  else $A[k] = R[j]$
    $j = j + 1$



Analysis of MergeSort

- Base case, $n = 1$ and algorithm does nothing.
- When $n \geq 2$, the running time of each step of the algorithm is as follows:
  - **Divide:** Calculate *mid*. $\Rightarrow \Theta(1)$
    ($\Theta(1)$ means constant time, independent of n.)
  - **Conquer:** Need to solve 2 subproblems of size $n/2$.
    $\Rightarrow 2T(n/2)$.
  - **Combine:** Execute MERGE. $\Rightarrow \Theta(n)$.

Analysis of MergeSort

$$T(n) = \begin{cases} \Theta(1) & , \text{if } n = 1 \\ \Theta(1) + 2T(n/2) + \Theta(n) & , \text{if } n > 1 \end{cases}$$

$$= \begin{cases} \Theta(1) & , \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & , \text{if } n > 1 \end{cases}$$

Recurrence for MERGE-SORT

O and Θ notation in expressions

$$T(n) = n^3 + \Theta(n^2)$$

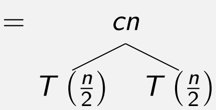
is equivalent to

$$T(n) = n^3 + h(n) \text{ , with } h(n) = \Theta(n^2)$$

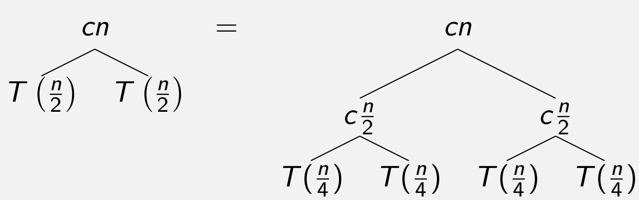
Solving the recurrence

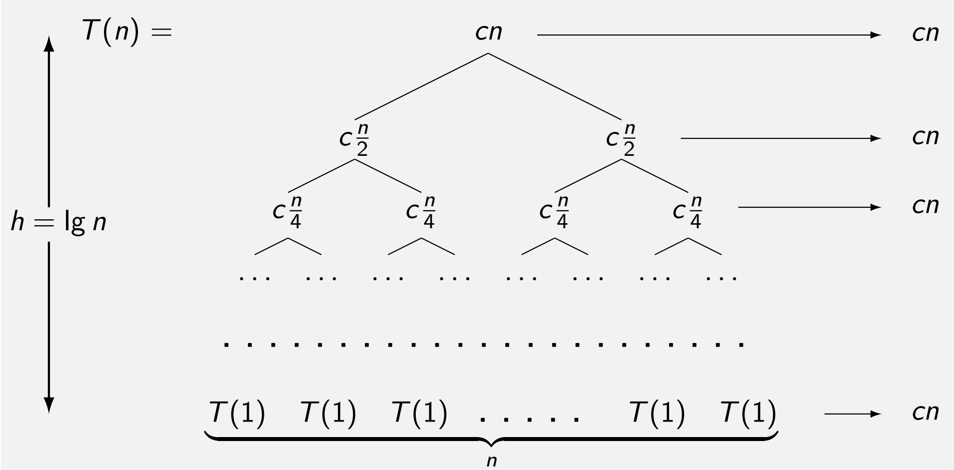
- Intuitive method: "unroll" the recursion with the help of a tree.
- $T(n) = 2T(n/2) + cn$, with some constant $c > 0$.
  (for now let's ignore the base case.)
- Let's write this sum as a tree:

$T(n) =$



$T(n) =$





Tree has $h + 1$ levels.

Total: $\overline{cn(1 + \lg n)}$

$= \Theta(n \lg n)$

Merge Sort vs. Insertion Sort

- $\Theta(n \lg n)$ grows slower than $\Theta(n^2)$.
- MERGE-SORT is asymptotically faster than INSERTION-SORT.
- In practice, INSERTION-SORT is better for small values of n.
  For large n, MERGE-SORT is much much better.

Merge Sort vs. Insertion Sort