CS207 Design and Analysis of Algorithms

Sajith Gopalan

Indian Institute of Technology Guwahati

sajith@iitg.ac.in

January 12, 2022

Divide and Conquer

Divide and Conquer

- solve a problem recursively
    - Divide the problem into a number of smaller instances of the same problem
    - For each of these subproblems, if its size is sufficiently large, then Conquer it recursively, else Conquer it directly
    - Combine the solutions to the subproblems into the solution for the original problem

Merge Sort

- To sort an array of size n
  - Divide the array into two subarrays of almost equal size
  - For each of these subproblems, if its size is at least two, then sort it recursively, else sort it using one compare-exchange
  - Merge the sorted subarrays into a single sorted array

Analysis

- If $T(n)$ is the worst case time complexity of Merge Sort, then
- $T(2) = d$
- For $n > 2$, $T(n) = 2T(n/2) + cn$
- $T(n) = 2T(n/2) + cn = 4T(n/4) + 2cn = 8T(n/8) + 3cn = \ldots$
- $T(n) = 2^k T(n/2^k) + kcn$
- Put $k = \log n - 1$
- $T(n) = \frac{n}{2} T(2) + c\frac{n}{2}(\log n - 1) = O(n \log n)$
- Here we assume that n is a power of 2.
- If not, we can always pad the array with ∞s to a size that is the nearest larger power of 2 to n
- This would increase n by a factor less than 2

Quicksort

Input: To sort an array $A[1, \ldots, n]$ of distinct elements

Algorithm 1 Quicksort

If $(n \leq 2)$ sort using at most one comparison
Pick $A[1]$ as the pivot
$left = 1$; $right = n + 1$;
while $left \leq right$ **do**
    $left$ =address of the leftmost element to the right of $left$ larger than the pivot (or $n + 1$, if that is undefined);
    $right$ =address of the rightmost element to the left of $right$ smaller than the pivot (or 1, if that is undefined);
    if $(left < right)$ interchange $A[left]$ and $A[right]$;
end while
Intechange $A[1]$ and $A[left]$
Recursively sort $A[1, \ldots, right]$
Recursively sort $A[left + 1, \ldots, n]$

Analysis

- If $T(n)$ is the worst case time complexity of Quicksort, then
- $T(0) = T(1) = T(2) = d$
- For $n > 2$, $T(n) \leq T(n-1) + T(0) + cn$
- $T(n) = T(n-1) + cn + d = T(n-2) + c(n+n-1) + 2d = T(n-3) + c(n+n-1+n-2) + 3d = \ldots$
- $T(n) = O(n^2)$
- The worst case happens when the input is already sorted

Analysis

- The average time complexity of Quicksort is $O(n \log n)$