1 Introduction

Given an instance \( x \) of a problem, the divide-and-conquer method works as follows.

\[
\text{function } DAQ(x) \\
\quad \text{if } x \text{ is sufficiently small or simple then} \\
\qquad \text{solve it directly} \\
\quad \text{else} \\
\qquad \text{divide } x \text{ into smaller subinstances } x_1, \ldots, x_k; \\
\qquad \text{for } i \leftarrow 1 \text{ to } k \text{ do } y_i \leftarrow DAQ(x_i); \\
\qquad \text{combine the } y_i \text{'s to obtain a solution } y \text{ to the original problem } x; \\
\quad \text{return } (y)
\]
2 Mergesort

- Algorithm:

```plaintext
procedure mergesort(A[1..n], i, j)
    /* Sort A[i..j] */
    if i < j then
        m ← (i + j) div 2
        mergesort(A, i, m) /* sort A[i..m] */
        mergesort(A, m + 1, j) /* sort A[m+1..j] */
        merge(A, i, m, j) /* merge A[i..m] with A[m+1..j] */
    end
```

- Initial call: mergesort(A[1..n], 1, n)
- Running time for merge: Θ(n)
- Running time for mergesort:

\[
T(n) = \begin{cases} 
  b, & \text{if } n \leq 1 \\
  2T(n/2) + cn, & \text{if } n > 1 
\end{cases}
\]

- Solving the recurrence yields \( T(n) = \Theta(n \log n) \).
3 Solving Recurrences

3.1 Iteration Method

\[
T(n) = \begin{cases} 
  b, & \text{if } n \leq 1 \\
  2T(n/2) + cn, & \text{if } n > 1
\end{cases}
\]

\[
T(n) = 2 \cdot T\left(\frac{n}{2}\right) + cn \\
= 2 \left[2 \cdot T\left(\frac{n}{4}\right) + cn/2\right] + cn \\
= 4 \cdot T\left(\frac{n}{4}\right) + 2cn \\
= 4 \left[2 \cdot T\left(\frac{n}{8}\right) + cn/4\right] + 2cn \\
= 8 \cdot T\left(\frac{n}{8}\right) + 3cn \\
= 2^3 \cdot T\left(\frac{n}{2^3}\right) + 3cn \\
= \ldots \\
= 2^{\log n} \cdot T\left(\frac{n}{2^{\log n}}\right) + cn \log n \\
= bn + cn \log n \\
= \Theta(n \log n)
\]

3.2 Recursion Tree
3.3 Master’s Theorem

**Theorem 1** If \( T(n) = aT(n/b) + f(n) \), then \( T(n) \) is bounded asymptotically as follows.

1. If \( f(n) = O(n^{\log_b a - \epsilon}) \), then \( T(n) = \Theta(n^{\log_b a}) \).
2. If \( f(n) = \Omega(n^{\log_b a \epsilon}) \), then \( T(n) = \Theta(f(n)) \).
3. If \( f(n) = \Theta(n^{\log_b a}) \), then \( T(n) = \Theta(f(n) \log n) = \Theta(n^{\log_b a \log n}) \).

**Definition 1** \( f(n) \) is polynomially smaller than \( g(n) \), denoted as \( f(n) \ll g(n) \), if \( f(n) = O(g(n)n^{-\epsilon}) \) for some \( \epsilon > 0 \).

Master’s theorem means:

1. If \( f(n) \ll n^{\log_b a} \), then \( T(n) = \Theta(n^{\log_b a}) \).
2. If \( f(n) \gg n^{\log_b a} \), then \( T(n) = \Theta(f(n)) \).
3. If \( f(n) \approx n^{\log_b a} \), then \( T(n) = \Theta(f(n) \log n) = \Theta(n^{\log_b a \log n}) \).

Applying Master’s theorem to
\[
T(n) = \begin{cases} 
  b, & \text{if } n \leq 1 \\
  2T(n/2) + cn, & \text{if } n > 1 
\end{cases}
\]
immediately yields \( T(n) = \Theta(n \log n) \).
3.4 More Examples

- \( T(n) = 9T(n/3) + n. \)
- \( T(n) = T(2n/3) + 1 \)
- \( T(n) = 3T(n/4) + n \log n. \)
- \( T(n) = 7T(n/2) + \Theta(n^2). \)
- \( T(n) = 2T(n/2) + n. \)
- \( T(n) = T(n/3) + T(2n/3) + n. \)
4 Quicksort

4.1 Quicksort

Quicksort(\(A[1..n], l, r\))
/* Sort \(A[l..r]\) */
if \(l < r\) then
    \(p \leftarrow \) Partition(\(A[1..n], l, r\))
    Quicksort(\(A, l, p - 1\)) /* sort \(A[l..p - 1]\) */
    Quicksort(\(A, p + 1, r\)) /* sort \(A[p + 1..r]\) */

Initial call: Quicksort(\(A[1..n], 1, n\))
4.2 Partition

- Partition $A[l..r]$ into two subarrays $A[l..p - 1]$ and $A[p + 1..r]$ such that
  

- Lumoto Partition: partition $A[l..r]$ into two subarrays $A[l..p - 1]$ and $A[p + 1..r]$ such that
  

```plaintext
Partition(A[1..n], l, r)
    x ← A[r]  /* pivot element */
    i ← l
    for j ← l to r - 1 do
        if A[j] < x then
            i ← i + 1
    return(i)
```
4.3 Time Complexity

- Best case: $\Theta(n \log n)$
- Worst case: $\Theta(n^2)$
- Average: $\Theta(n \log n)$
4.4 Improvements

- Use Insertion_Sort when the array is small (e.g. $M = 10$):

  \[
  \text{Quicksort}(A[1..n], l, r) \\
  \quad /* Sort } A[l..r] */ \\
  \quad \text{if } r - l < M \text{ then} \\
  \quad \quad \text{Insertion_Sort}(A[1..n], l, r) \\
  \quad \text{else} \\
  \quad \quad p \leftarrow \text{Partition}(A[1..n], l, r) \\
  \quad \quad \text{Quicksort}(A, l, p - 1) /* sort } A[l..p-1] */ \\
  \quad \quad \text{Quicksort}(A, p + 1, r) /* sort } A[p+1..r] */ \\
  \]

- Use a random element in $A[l..r]$ as the pivot:

  \[
  p \leftarrow \text{random}(l..r) \\
  \quad \text{exchange } A[p] \leftrightarrow A[r] \\
  \quad x \leftarrow A[r] /* pivot element */ \\
  \]

- Median-of-three: use the following as the pivot element.

  \[
  \text{median}\{A[l], A[m], A[r]\}, \text{ where } m = \lfloor (l + r)/2 \rfloor \\
  \]
5 Majority Element Problem

Let $A[1..n]$ be an array of integers. An element in $A$ is said to be a majority element if it appears in $A$ for more than $n/2$ times. Write an $O(n \log n)$ divide-and-conquer algorithm that determines whether or not $A[1..n]$ has a majority element and, if it does, finds the majority element. You cannot sort the array.
6 Convex Hull (Another Example of Divide-and-Conquer)

6.1 Problem Statement

- Given a set \( A \) of \( n \) points in the plane, we want to find the convex hull of \( A \).
- The convex hull of \( A \) is the smallest convex polygon that contains all the points in \( A \).
- For simplicity, assume no two points have the same \( x \) or \( y \) coordinate.

6.2 Sketch of Algorithm

- Let \( A = \{p_1, p_2, \ldots, p_n\} \). Denote the convex hull of \( A \) by \( CH(A) \).
- Observation: the segment \( p_ip_j \) is an edge of \( CH(A) \) if all other points of \( A \) are on the same side of \( p_ip_j \).
- Straightforward method: \( \Omega(n^2) \)
- Divide and conquer: \( O(n \log n) \)
- Basic ideas:
  1. Sort \( A \) by \( x \)-coordinate.
  2. If \( |A| \leq 3 \), solve the problem directly. Otherwise, apply divide-and-conquer as follows.
  3. Divide \( A \) into two subsets: \( A = B \cup C \).
  4. Find \( CH(B) \), the convex hull of \( B \).
  5. Find \( CH(C) \), the convex hull of \( C \).
  6. Combine the two convex hulls.
6.3 Combine $CH(B)$ and $CH(C)$ to get $CH(A)$

1. We need to find the “upper bridge” and the “lower bridge” that connect the two convex hulls.

2. The upper bridge is the edge $vw$, where $v \in CH(B)$ and $w \in CH(C)$, such that all other vertices in $CH(B)$ and in $CH(C)$ are below $vw$.

3. Suffices to check if both neighbors of $v$ in $CH(B)$ and both neighbors of $w$ in $CH(w)$ are all below $vw$.

4. Find the upper bridge as follows:

   (a) $v :=$ the rightmost point in $CH(B)$;
       $w :=$ the leftmost point in $CH(C)$.

   (b) Loop
       
       if counterclockwise_neighbor($v$) lies above the line $vw$ then
           $v :=$ counterclockwise_neighbor($v$)
       else if clockwise_neighbor($w$) lies above the line $vw$ then
           $w :=$ clockwise_neighbor($w$)
       else
           exit from the loop
       End of loop

   (c) $vw$ is the upper bridge.

5. Find the lower bridge similarly.