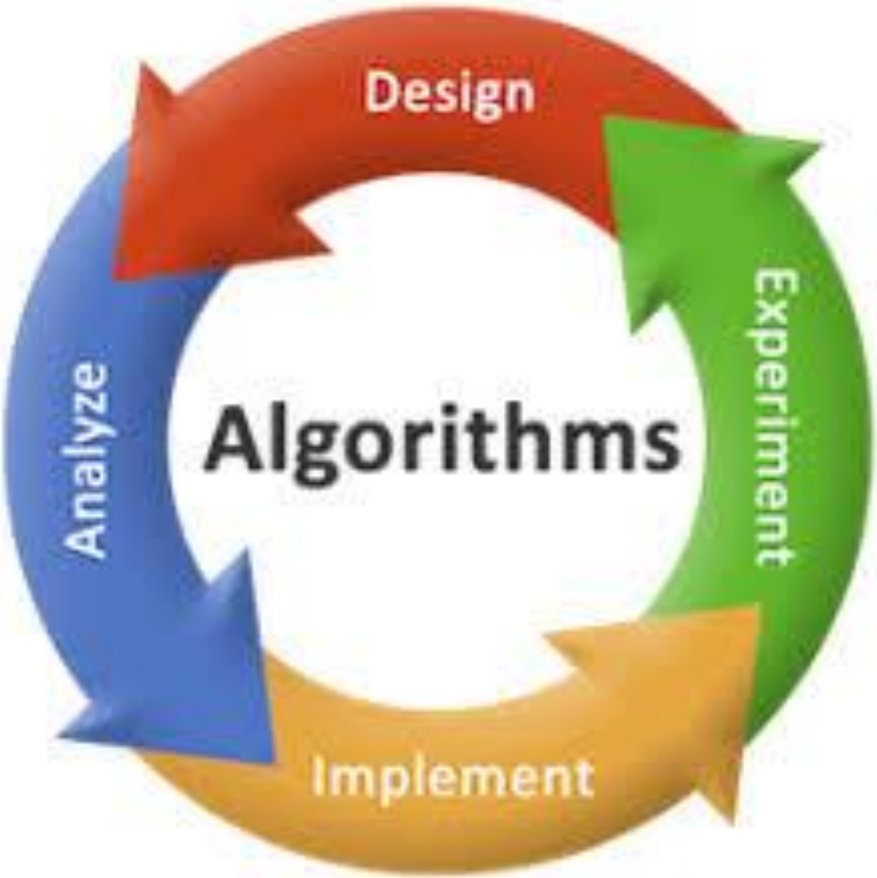


Design & Analysis of Algorithms

Lecture 16

Divide & Conquer

Merge Sort Algorithm

Divide & Conquer

```
Algorithm DAndC(P)
{
    if Small(P) then return S(P);
    else
    {
        divide P into smaller instances P_1, P_2, ..., P_k, k ≥ 1;
        Apply DAndC to each of these subproblems;
        return Combine(DAndC(P_1),DAndC(P_2),...,DAndC(P_k));
    }
}
```

$$T(n) = \begin{cases} g(n) & n \text{ small} \\ T(n_1) + T(n_2) + \cdots + T(n_k) \quad + \quad f(n) & \text{otherwise} \end{cases}$$

Divide & Conquer

- Its complexity can be shown by recurrence relation of the form

$$T(n) = \begin{cases} T(1) & n = 1 \\ aT(n/b) + f(n) & n > 1 \end{cases}$$

where a and b are known constants. We assume that $T(1)$ is known and n is a power of b (i.e., $n = b^k$).

Merge Sort

- **Mergesort** is a perfect example of a successful application of the divide-and conquer technique.
- It sorts a given array A[0..n − 1] by **dividing** it into two halves

  $$A[0..\lfloor n/2 \rfloor - 1] \text{ and } A[\lfloor n/2 \rfloor..n-1],$$

  sorting each of them recursively, and then merging the two smaller sorted arrays into a single sorted one.

Merge Sort

ALGORITHM $Mergesort(A[0..n-1])$

//Sorts array $A[0..n-1]$ by recursive mergesort
//Input: An array $A[0..n-1]$ of orderable elements
//Output: Array $A[0..n-1]$ sorted in nondecreasing order
if $n > 1$
copy $A[0..\lfloor n/2 \rfloor - 1]$ to $B[0..\lfloor n/2 \rfloor - 1]$
copy $A[\lfloor n/2 \rfloor..n-1]$ to $C[0..\lceil n/2 \rceil - 1]$
$Mergesort(B[0..\lfloor n/2 \rfloor - 1])$
$Mergesort(C[0..\lceil n/2 \rceil - 1])$
$Merge(B, C, A)$ //see below

Merge Sort

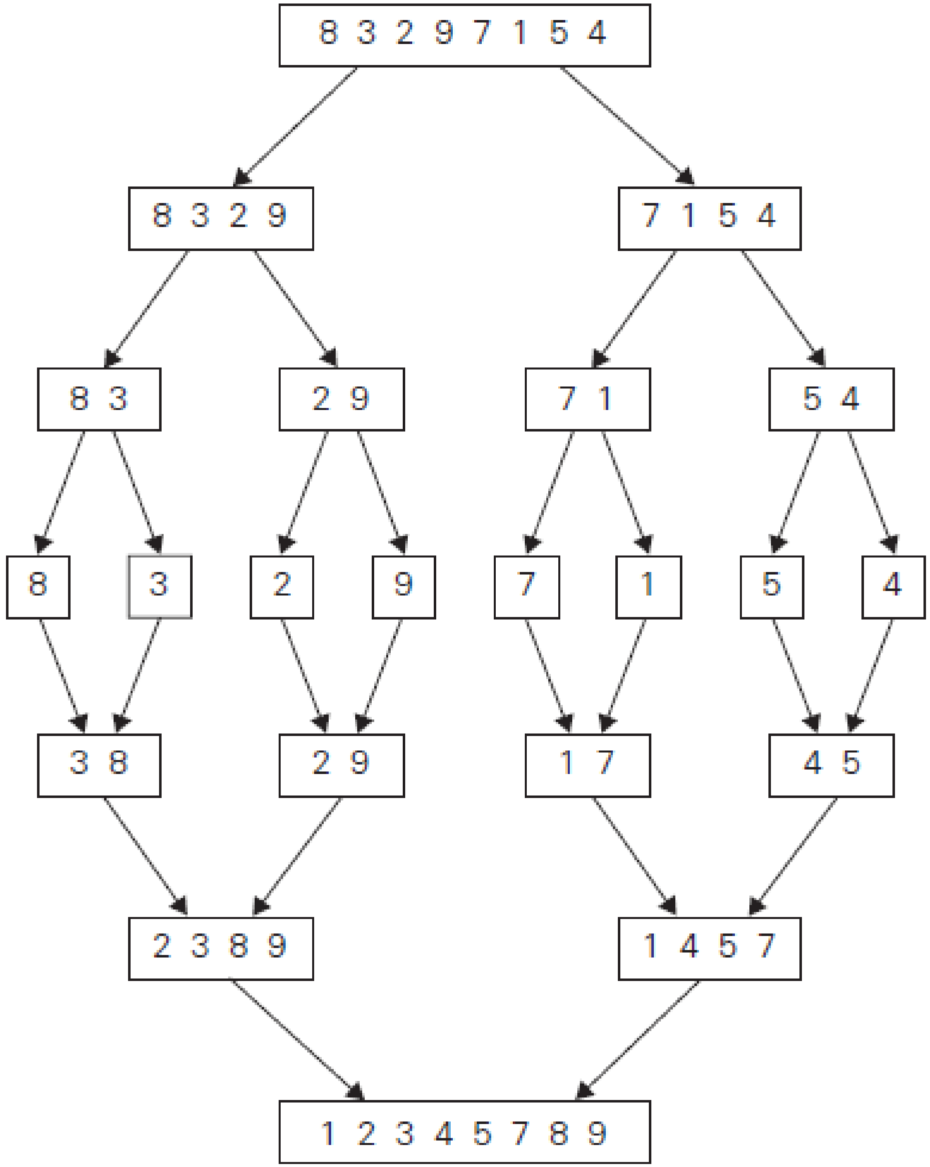
ALGORITHM $Merge(B[0..p-1], C[0..q-1], A[0..p+q-1])$

//Merges two sorted arrays into one sorted array
//Input: Arrays $B[0..p-1]$ and $C[0..q-1]$ both sorted
//Output: Sorted array $A[0..p+q-1]$ of the elements of B and C
$i \leftarrow 0$; $j \leftarrow 0$; $k \leftarrow 0$
while $i < p$ **and** $j < q$ **do**
  if $B[i] \leq C[j]$
    $A[k] \leftarrow B[i]$; $i \leftarrow i+1$
  else $A[k] \leftarrow C[j]$; $j \leftarrow j+1$
  $k \leftarrow k+1$
if $i = p$
  copy $C[j..q-1]$ to $A[k..p+q-1]$
else copy $B[i..p-1]$ to $A[k..p+q-1]$

Merge Sort

- **Example**

{8, 3, 2, 9, 7, 1, 5, 4}



Merge Sort

```
Algorithm MergeSort(low, high)
// a[low : high] is a global array to be sorted.
// Small(P) is true if there is only one element
// to sort. In this case the list is already sorted.
{
    if (low < high) then  // If there are more than one element
    {
        // Divide P into subproblems.
            // Find where to split the set.
                mid := ⌊(low + high)/2⌋;
        // Solve the subproblems.
            MergeSort(low, mid);
            MergeSort(mid + 1, high);
        // Combine the solutions.
            Merge(low, mid, high);
    }
}
```

Book: Horowitz & Sahni

Merge Sort

```
Algorithm Merge(low, mid, high)
// a[low : high] is a global array containing two sorted
// subsets in a[low : mid] and in a[mid + 1 : high]. The goal
// is to merge these two sets into a single set residing
// in a[low : high]. b[ ] is an auxiliary global array.
{
    h := low; i := low; j := mid + 1;
    while ((h ≤ mid) and (j ≤ high)) do
    {
        if (a[h] ≤ a[j]) then
        {
            b[i] := a[h]; h := h + 1;
        }
        else
        {
            b[i] := a[j]; j := j + 1;
        }
        i := i + 1;
    }
```

Book: Horowitz & Sahni

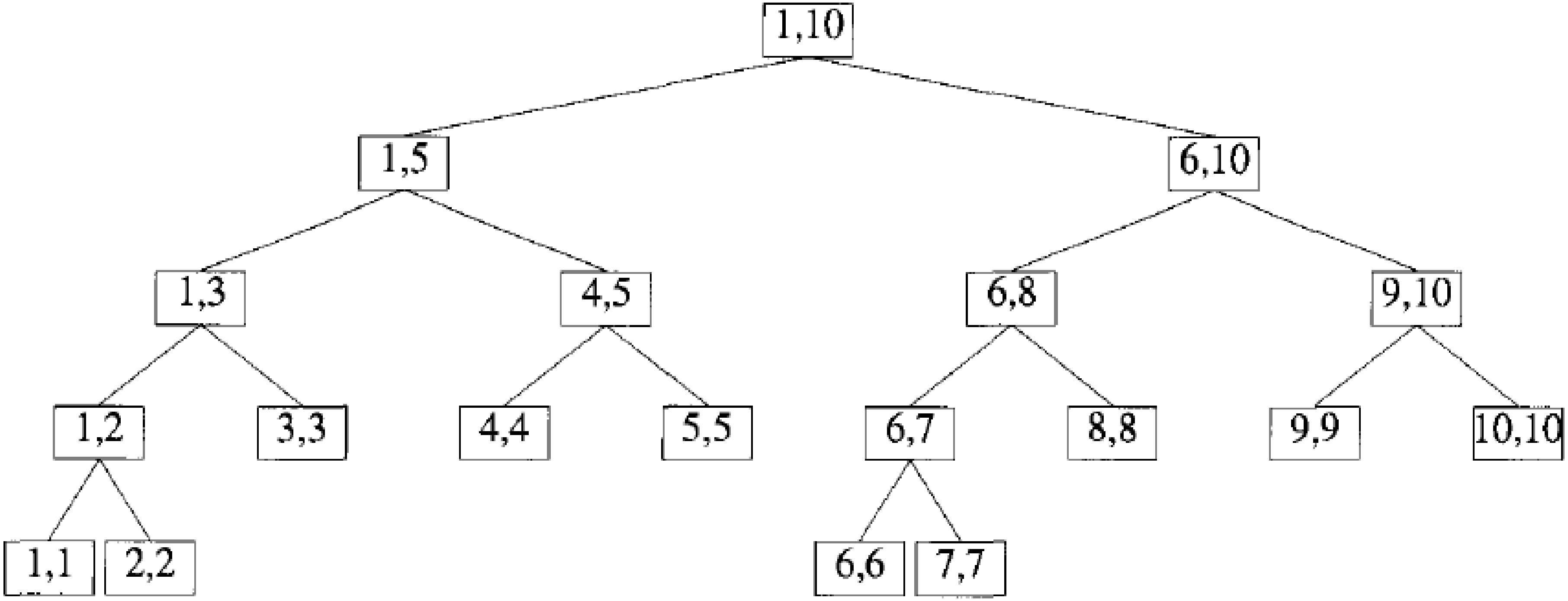
Merge Sort

```
        if (h > mid) then
            for k := j to high do
            {
                b[i] := a[k]; i := i + 1;
            }
        else
            for k := h to mid do
            {
                b[i] := a[k]; i := i + 1;
            }
        for k := low to high do a[k] := b[k];
}
```

Merge Sort

- **Example 2: Sort the following 10 elements**

$$(179,\ 254,\ 285,\ 310,\ 351,\ 423,\ 450,\ 520,\ 652,\ 861)$$



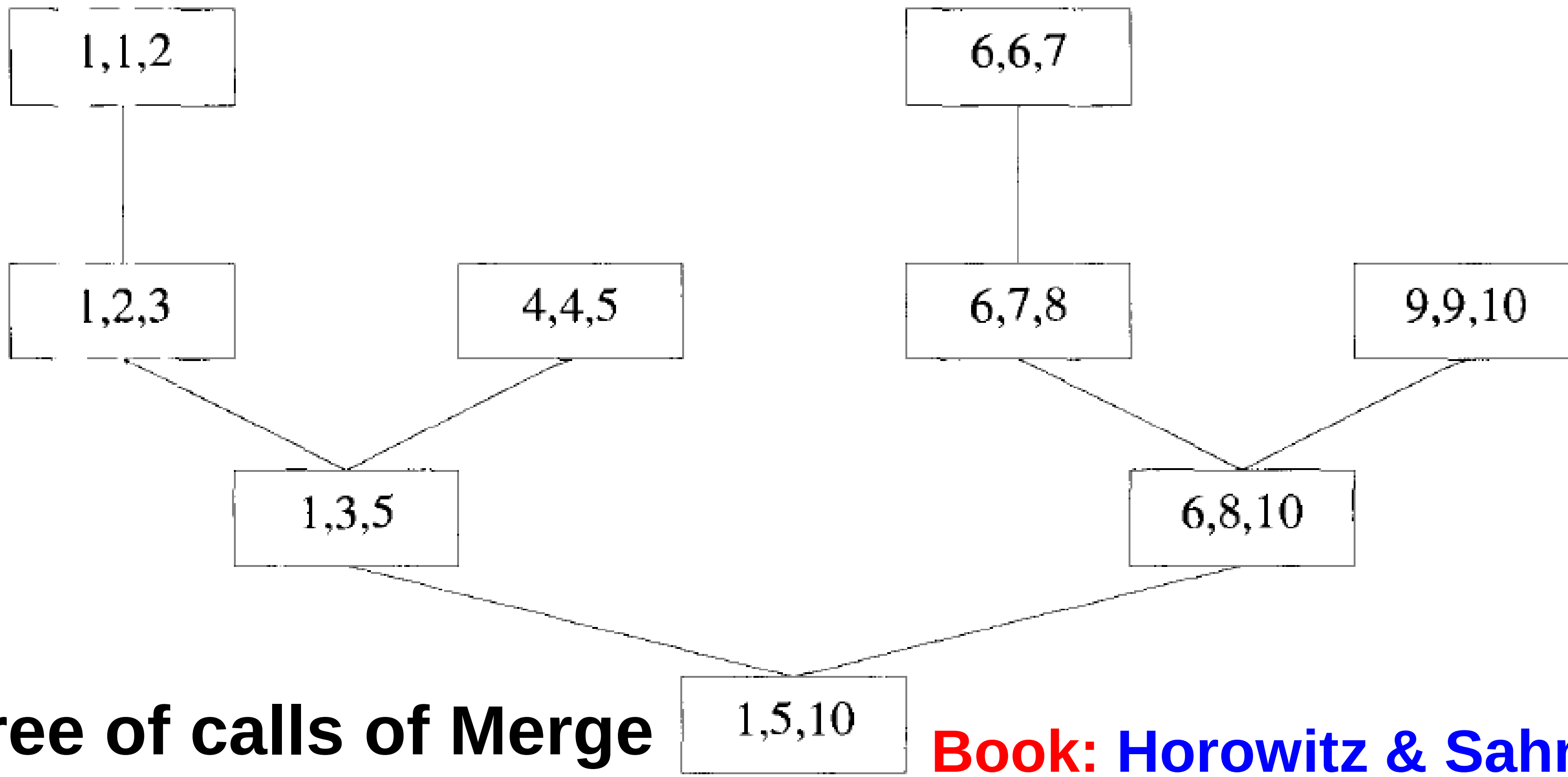
Tree of calls of MergeSort(1,10)

Book: Horowitz & Sahni

Merge Sort

- **Example 2: Sort the following 10 elements**

$$(179,\ 254,\ 285,\ 310,\ 351,\ 423,\ 450,\ 520,\ 652,\ 861)$$



Tree of calls of Merge

Book: Horowitz & Sahni

Merge Sort

- **Time Complexity**

$$C(n) = 2C(n/2) + C_{merge}(n) \quad \text{for } n > 1, \quad C(1) = 0.$$

for the worst case, $C_{merge}(n) = n - 1$

$$C_{worst}(n) = 2C_{worst}(n/2) + n - 1 \quad \text{for } n > 1, \quad C_{worst}(1) = 0.$$

$$C_{worst}(n) = n \log_2 n - n + 1$$

- **multiway mergesort? 3 or 4 parts.**

Merge Sort

- **Time Complexity**

If the time for the merging operation is proportional to n, then the computing time for merge sort is described by the recurrence relation

$$T(n) = \begin{cases} a & n = 1, a \text{ a constant} \\ 2T(n/2) + cn & n > 1, c \text{ a constant} \end{cases}$$

Merge Sort

- **Time Complexity**

Solution

$$
\begin{aligned}
T(n) &= 2(2T(n/4) + cn/2) + cn \\
&= 4T(n/4) + 2cn \\
&= 4(2T(n/8) + cn/4) + 2cn \\
&\vdots \\
&= 2^k T(1) + kcn \\
&= an + cn \log n
\end{aligned}
$$

It is easy to see that if $2^k < n \leq 2^{k+1}$, then $T(n) \leq T(2^{k+1})$. Therefore

$$T(n) = O(n \log n)$$

References

- **Chapter 4**: Anany Levitin, “Introduction to the Design and Analysis of Algorithms”, Pearson Education, Third Edition, 2017.

- **Chapter 3**: E Horowitz, S Sahni, S Rajasekaran, “Computer Algorithms”, Computer Science Press, Third Edition, 2008.

Homework

- **Matrix Multiplication**

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

Recurrence Relation

$$T(n) = \begin{cases} b & n \le 2 \\ 8T(n/2) + cn^2 & n > 2 \end{cases}$$

Homework

- **Matrix Multiplication**

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$\begin{aligned} P &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ Q &= (A_{21} + A_{22})B_{11} \\ R &= A_{11}(B_{12} - B_{22}) \\ S &= A_{22}(B_{21} - B_{11}) \\ T &= (A_{11} + A_{12})B_{22} \\ U &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ V &= (A_{12} - A_{22})(B_{21} + B_{22}) \end{aligned}$$

Homework

- **Matrix Multiplication**

$$
\begin{aligned}
C_{11} &= P + S - T + V \\
C_{12} &= R + T \\
C_{21} &= Q + S \\
C_{22} &= P + R - Q + U
\end{aligned}
$$

Recurrence Relation

$$
T(n) = \begin{cases} b & n \le 2 \\ 7T(n/2) + an^2 & n > 2 \end{cases}
$$