Recursive Algorithms
and
Recurrence Relations

Selection Sort (Recursive)

Input : Array A of n elements.

Result : Permutation of A such that
$A[1] \leq A[2] \leq A[3] \leq \ldots \leq A[n]$.

```
procedure SelectionSort(A[ ],n)
if (n ≤ 1) then
    return;
else
    for i ← 1 to n − 1 do
        if (A[i] > A[n]) then Swap(A[i], A[n]) ;
    end
    SelectionSort(A[ ],n − 1);
end
```

Recurrence Relations

Methods for solving recurrence relations:

- Expansion into a series;
- Induction (called the substitution method by the text);
- Recursion tree;
- Characteristic polynomial (not covered in this course);
- Master's Theorem (not covered in this course).

Select Max (Recursive)

Input : Array A of n integers.

Output : Maximum of $\mathsf{A}[1], \mathsf{A}[2], \ldots, \mathsf{A}[n]$.

```
function SelectMax(A[ ],n)
if (n = 1) then
    return (A[1]);
else
    for i = 1 to ⌊n/2⌋ do
        A[i] ← max(A[i], A[n − i + 1]);
    end
    x ← SelectMax (A[ ],⌈n/2⌉);
    return (x);
end
```

Locate in Sorted Array

Given a sorted array

$$\mathsf{A}[\,] = [2, 3, 7, 9, 14, 17, 32, 35, 36, 38, 51],$$

and a key K,
determine if key K is in array A and report its location.

Binary Search: Recursive Version

Output : p such that ($\mathsf{A}[p] = \mathsf{K}$ and $i \leq p \leq j$) or -1 if there is no such p.

```
function BinarySearchRec(A[ ],i,j,K)
if (i ≤ j) then
    midp ← ⌊(i + j)/2⌋;
    if (K = A[midp]) then index ← midp;
    else if (K < A[midp]) then
        index ← BinarySearchRec(A,i,midp − 1,K);
    else    /* K > A[midp] */
        index ← BinarySearchRec(A,midp + 1,j,K);
    return (index);
else
    return (−1);
end
```

Fibonacci Numbers

Definition:

$f(0) = 0;$
$f(1) = 1;$
$f(n) = f(n-1) + f(n-2)$ for $n > 1$.

Fibonnaci numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Fibonacci Numbers

Definition:
$f(0) = 0$;
$f(1) = 1$;
$f(n) = f(n-1) + f(n-2)$ for $n > 1$.

Output : The n'th Fibonacci number, $f(n)$.

function `fib`(n)
if $(n = 0)$ **then** **return** (0) ;
if $(n = 1)$ **then** **return** (1) ;
$\mathsf{f1} \leftarrow \texttt{fib}(n-1)$;
$\mathsf{f2} \leftarrow \texttt{fib}(n-2)$;
return $(\mathsf{f1} + \mathsf{f2})$;

Example

function `Func1`(n)

if $(n = 0)$ **then** **return** (0) ;

if $(n = 1)$ **then** **return** (1) ;

$x \leftarrow 0$;

for $i \leftarrow 1$ **to** $n - 1$ **do**

  $x \leftarrow x +$ `Func1` (i);

end

return (x);

Merge Sort

Input : Array A of at least j elements.
Integers i and j.

Result : A permutation of the i through j elements of A such that $A[i] \leq A[i+1] \leq A[i+2] \leq \ldots \leq A[j]$.

procedure `MergeSort`(A[ ],i,j)

if $(i < j)$ **then**
    midp $\leftarrow \lfloor (i+j)/2 \rfloor$;
    `MergeSort`(A[ ],i,midp);
    `MergeSort`(A[ ],midp $+ 1$,j);
    /* *Merge* A[$i, i+1, \ldots,$ midp] *with* A[midp $+ 1, \ldots, j$] */
    `Merge`(A[ ],i,midp,j);
end

Copy Array

Input : Array A of at least j elements.
Integers i and j.

Output : Array B containing A$[i, i+1, \ldots, j]$ followed by ∞.

procedure `Copy`(A[ ],i,j, B[])

$p \leftarrow 1$;

for $k \leftarrow i$ **to** j **do**

  B$[p] \leftarrow A[k]$;

  $p \leftarrow p + 1$;

end

/* *Add* ∞ *at the end of* B[] */

$B[p] \leftarrow \infty$;

Merge

```
procedure Merge(A[ ],first,midp, last)
Copy(A[ ],first,midp, L[]);
Copy(A[ ],midp + 1,last, R[]);
i ← 1;
j ← 1;
for k ← first to last do
    if (L[i] < R[j]) then
        A[k] ← L[i];
        i ← i + 1;
    else
        A[k] ← R[j];
        j ← j + 1;
    end
end
```

Merge Sort

Input : Array A of at least j elements.
Integers i and j.

Result : A permutation of the i through j elements of A such that $A[i] \leq A[i+1] \leq A[i+2] \leq \ldots \leq A[j]$.

procedure `MergeSort`(A[ ],i,j)

if $(i < j)$ **then**

  midp $\leftarrow \lfloor (i+j)/2 \rfloor$;

  `MergeSort`(A[ ],i,midp);

  `MergeSort`(A[ ],midp $+ 1$,j);

  /* *Merge* A[$i, i+1, \ldots,$ midp] *with* A[midp $+ 1, \ldots, j$] */

  `Merge`(A[ ],i,midp,j);

end

Recurrence Relations

Methods for solving recurrence relations:

- Expansion into a series;
- Induction (called the substitution method by the text);
- Recursion tree;
- Characteristic polynomial (not covered in this course);
- Master's Theorem (not covered in this course).

Merge Sort: Version 2: Split into 3 Parts

Result : A permutation of the i through j elements of A such that $A[i] \leq A[i+1] \leq A[i+2] \leq \ldots \leq A[j]$.

procedure `MergeSortII`(A[ ],i,j)

if $(i < j)$ **then**
    $n \leftarrow j - i + 1$;
    $m1 \leftarrow i + \lfloor n/3 \rfloor$;
    $m2 \leftarrow i + \lfloor 2n/3 \rfloor$;
    `MergeSortII`(A[ ],i,$m1$);
    `MergeSortII`(A[ ],$m1+1$,$m2$);
    `MergeSortII`(A[ ],$m2+1$,j);
    /* *Merge* A[$i,\ldots,m1$] *and* A[$m1+1,\ldots,m2$] */
    `Merge`(A[ ],i,$m1$,$m2$);
    /* *Merge* A[$i,\ldots,m2$] *and* A[$m2+1,\ldots,j$] */
    `Merge`(A[ ],i,$m2$,j);
end

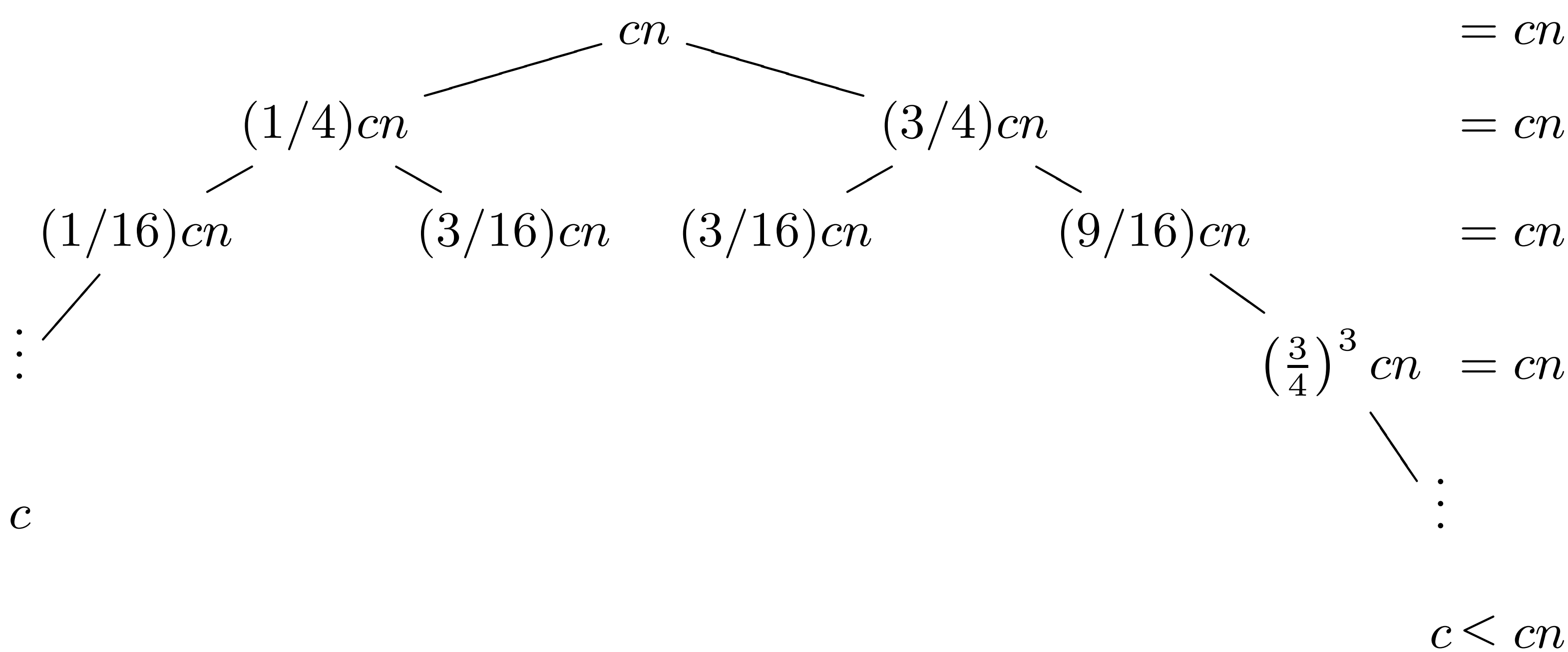
Merge Sort: Version 3: Imbalanced Split

Result : A permutation of the i through j elements of A such that $A[i] \leq A[i+1] \leq A[i+2] \leq \ldots \leq A[j]$.

```
procedure MergeSortIII(A[ ],i,j)
if (i < j) then
    n ← j − i + 1;
    m1 ← i + ⌊n/4⌋;
    MergeSortIII(A[ ],i,m1);
    MergeSortIII(A[ ],m1,j);
    /* Merge A[i,...,m1] and A[m1+1,...,j]                    */
    Merge(A[ ],i,m1,j);
end
```

Solving $T(n) = cn + T(n/4) + T(3n/4)$.

Recursion tree:



Tree height:

Length of shortest path from root to leaf:

Running time:

Chip and Conquer

$$T(n) = T(n-a) + f(n)$$

$$\begin{aligned}
T(n) &= T(n-1) + c, & T(n) &\in \Theta(n); \\
T(n) &= T(n-1) + cn, & T(n) &\in \Theta(n^2); \\
T(n) &= T(n-1) + cn^2, & T(n) &\in \Theta(n^3).
\end{aligned}$$

Divide and Conquer

$$T(n) = aT(n/b) + f(n), \qquad (a \geq 1 \text{ and } b > 1).$$

$$\begin{array}{ll}
T(n) = T(n/2) + c, & T(n) \in \Theta(\log_2(n)); \\
T(n) = T(n/3) + c, & T(n) \in \Theta(\log_2(n)); \\
T(n) = T(n/2) + cn, & T(n) \in \Theta(n); \\
T(n) = T(n/3) + cn, & T(n) \in \Theta(n);
\end{array}$$

$$\begin{array}{ll}
T(n) = 2T(n/2) + cn, & T(n) \in \Theta(n \log_2(n)); \\
T(n) = 3T(n/3) + cn, & T(n) \in \Theta(n \log_2(n)).
\end{array}$$

More Divide and Conquer

$$T(n) = aT(n/b) + f(n), \qquad (a \geq 1 \text{ and } b > 1).$$

$$T(n) = 3T(n/2) + cn, \qquad T(n) \in \Theta(n^{\log_2(3)});$$
$$T(n) = 4T(n/2) + cn, \qquad T(n) \in \Theta(n^{\log_2(4)}) = \Theta(n^2);$$

$$T(n) = 2T(n/2) + cn^2, \qquad T(n) \in \Theta(n^2);$$
$$T(n) = 4T(n/2) + cn^2, \qquad T(n) \in \Theta(n^2 \log(n)).$$

Asymmetric Recurrence Relations

$$
\begin{aligned}
T(n) &= T(n/3) + T(2n/3) + cn, && T(n) \in \Theta(n \log_2(n)); \\
T(n) &= T(n/4) + T(3n/4) + cn, && T(n) \in \Theta(n \log_2(n)); \\
T(n) &= T(n/5) + T(4n/5) + cn, && T(n) \in \Theta(n \log_2(n)); \\
T(n) &= T(2n/5) + T(3n/5) + cn, && T(n) \in \Theta(n \log_2(n)); \\
T(n) &= T(n/6) + T(2n/6) + T(3n/6) + cn, && T(n) \in \Theta(n \log_2(n)).
\end{aligned}
$$

$$
\begin{aligned}
T(n) &= T(n/4) + T(2n/4) + cn, && T(n) \in \Theta(n); \\
T(n) &= T(n/5) + T(2n/5) + cn, && T(n) \in \Theta(n); \\
T(n) &= T(n/5) + T(3n/5) + cn, && T(n) \in \Theta(n); \\
T(n) &= T(n/6) + T(4n/6) + cn, && T(n) \in \Theta(n).
\end{aligned}
$$

Exponential Functions

Assume $f(n) \geq 0$ and $T(1) > 0$.

$$T(n) = 2T(n-1) + f(n), \qquad T(n) \in \Omega(2^n);$$
$$T(n) = 3T(n-1) + f(n), \qquad T(n) \in \Omega(3^n);$$
$$T(n) = 4T(n-1) + f(n), \qquad T(n) \in \Omega(4^n);$$

$$T(n) = 2T(n-2) + f(n), \qquad T(n) \in \Omega(2^{n/2});$$
$$T(n) = 2T(n-3) + f(n), \qquad T(n) \in \Omega(2^{n/3});$$

$$T(n) = T(n-1) + T(n-2) + f(n), \qquad T(n) \in \Omega(2^{n/2});$$
$$T(n) = T(n-1) + T(n-2) + T(n-3) + f(n),$$
$$T(n) \in \Omega(2^{n/2});$$
$$T(n) = f(n) + \sum_{i=1}^{n-1} T(i), \qquad T(n) \in \Omega(2^{n/2}).$$