Binary Search

Algorithm 5.2 BINARYSEARCHREC

Input: An array $A[1..n]$ of n elements sorted in nondecreasing order and an element x.

Output: j if $x = A[j], 1 \leq j \leq n$, and 0 otherwise.

1. $binarysearch(1, n)$

Procedure $binarysearch(low, high)$

1. **if** $low > high$ **then return** 0
2. **else**
3. $\quad mid \leftarrow \lfloor (low + high)/2 \rfloor$
4. $\quad$ **if** $x = A[mid]$ **then return** mid
5. $\quad$ **else if** $x < A[mid]$ **then return** $binarysearch(low, mid - 1)$
6. $\quad$ **else return** $binarysearch(mid + 1, high)$
7. **end if**

Merge Sort

Algorithm 5.3 MERGESORT
Input: An array $A[1..n]$ of n elements.

Output: $A[1..n]$ sorted in nondecreasing order.

$\quad$ $mergesort(A, 1, n)$

Procedure $mergesort(A, low, high)$
$\quad$ **if** $low < high$ **then**
$\quad\quad$ $mid \leftarrow \lfloor (low + high)/2 \rfloor$
$\quad\quad$ $mergesort(A, low, mid)$
$\quad\quad$ $mergesort(A, mid + 1, high)$
$\quad\quad$ MERGE $(A, low, mid, high)$
$\quad$ **end if**

Algorithm 1.3 MERGE
Input: An array $A[1..m]$ of elements and three indices p, q, and r, with $1 \leq p \leq q < r \leq m$, such that both the subarrays $A[p..q]$ and $A[q+1..r]$ are sorted individually in nondecreasing order.

Output: $A[p..r]$ contains the result of merging the two subarrays $A[p..q]$ and $A[q+1..r]$.

$\quad$ **comment:** *$B[p..r]$ is an auxiliary array.*
$\quad$ $s \leftarrow p; \quad t \leftarrow q + 1; \quad k \leftarrow p$
$\quad$ **while** $s \leq q$ **and** $t \leq r$
$\quad\quad$ **if** $A[s] \leq A[t]$ **then**
$\quad\quad\quad$ $B[k] \leftarrow A[s]$
$\quad\quad\quad$ $s \leftarrow s + 1$
$\quad\quad$ **else**
$\quad\quad\quad$ $B[k] \leftarrow A[t]$
$\quad\quad\quad$ $t \leftarrow t + 1$
$\quad\quad$ **end if**
$\quad\quad$ $k \leftarrow k + 1$
$\quad$ **end while**
$\quad$ **if** $s = q + 1$ **then** $B[k..r] \leftarrow A[t..r]$
$\quad$ **else** $B[k..r] \leftarrow A[s..q]$
$\quad$ **end if**
$\quad$ $A[p..r] \leftarrow B[p..r]$

Quick Sort

Algorithm 5.5 SPLIT

Input: An array of elements $A[low..high]$.

Output: (1)A with its elements rearranged, if necessary, as described above.
(2) w, the new position of the splitting element A$[low]$.

```
i ← low
x ← A[low]
for j ← low + 1 to high
    if A[j] ≤ x then
        i ← i + 1
        if i ≠ j then interchange A[i] and A[j]
    end if
end for
interchange A[low] and A[i]
w ← i
return A and w
```

Algorithm 5.6 QUICKSORT

Input: An array $A[1..n]$ of n elements.

Output: The elements in A sorted in nondecreasing order.

```
quicksort(A, 1, n)
```

Procedure *quicksort*$(A, low, high)$

```
if low < high then
    SPLIT(A[low..high], w)    {w is the new position of A[low]}
    quicksort(A, low, w − 1)
    quicksort(A, w + 1, high)
end if
```

Strassen's Matrix Multiplication

Algorithm 3 Strassen's Algorithm

function STRASSEN(M,N)
    if M is 1×1 **then**
        return $M_{11}N_{11}$
    end if
    Let $M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$ and $N = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$
    Set $S_1 = \text{STRASSEN}(B - D, G + H)$
    Set $S_2 = \text{STRASSEN}(A + D, E + H)$
    Set $S_3 = \text{STRASSEN}(A - C, E + F)$
    Set $S_4 = \text{STRASSEN}(A + B, H)$
    Set $S_5 = \text{STRASSEN}(A, F - H)$
    Set $S_6 = \text{STRASSEN}(D, G - E)$
    Set $S_7 = \text{STRASSEN}(C + D, E)$
    return $\begin{pmatrix} S_1 + S_2 - S_4 + S_6 & S_4 + S_5 \\ S_6 + S_7 & S_2 - S_3 + S_5 - S_7 \end{pmatrix}$
end function

Tiling the Defective Chessboard

```
// n is size of given square, p is location of missing cell
Tile(int n, Point p)

1) Base case: n = 2, A 2 x 2 square with one cell missing is nothing
   but a tile and can be filled with a single tile.

2) Place a L shaped tile at the center such that it does not cover
   the n/2 * n/2 subsquare that has a missing square. Now all four
   subsquares of size n/2 x n/2 have a missing cell (a cell that doesn't
   need to be filled).

3) Solve the problem recursively for following four. Let p1, p2, p3 and
   p4 be positions of the 4 missing cells in 4 squares.
   a) Tile(n/2, p1)
   b) Tile(n/2, p2)
   c) Tile(n/2, p3)
   d) Tile(n/2, p4)
```

`

Find Maximum and Minimum

```
int max, min;
void maxmin(int i, int j)
{
 int max1, min1, mid;
 if(i==j)
 {
  max = min = a[i];
 }
 else
 {
  if(i == j-1)
  {
   if(a[i] <a[j])
   {
    max = a[j];
    min = a[i];
   }
   else
   {
    max = a[i];
    min = a[j];
   }
  }
  else
  {
   mid = (i+j)/2;
   maxmin(i, mid);
   max1 = max; min1 = min;
   maxmin(mid+1, j);
   if(max <max1)
    max = max1;
   if(min > min1)
    min = min1;
  }
 }
}
```

max = min= a[0]

The array size = 1

The array size = 2

The array size > 2

Radix Sort

Algorithm 4.5 RADIXSORT

Input: A linked list of numbers $L = \{a_1, a_2, \ldots, a_n\}$ and k, the number of digits.

Output: L sorted in nondecreasing order.

```
for j ← 1 to k
    Prepare 10 empty lists L_0, L_1, ..., L_9.
    while L is not empty
        a ← next element in L. Delete a from L.
        i ← jth digit in a. Append a to list L_i.
    end while
    L ← L_0
    for i ← 1 to 9
        L ← L, L_i        {append list L_i to L}
    end for
end for
return L
```

Bubble Sort (2 methods)

```
bubbleSort(array)
  for i <- 1 to indexOfLastUnsortedElement-1
    if leftElement > rightElement
      swap leftElement and rightElement
end bubbleSort
```

Algorithm 1.17 BUBBLESORT

Input: An array $A[1..n]$ of n elements.

Output: $A[1..n]$ sorted in nondecreasing order.

$i \leftarrow 1;$ $sorted \leftarrow false$
while $i \leq n-1$ **and not** $sorted$
    $sorted \leftarrow true$
    for $j \leftarrow n$ **downto** $i+1$
        if $A[j] < A[j-1]$ **then**
            interchange $A[j]$ and $A[j-1]$
            $sorted \leftarrow false$
        end if
    end for
    $i \leftarrow i+1$
end while

Selection Sort

Algorithm 1.4 SELECTIONSORT

Input: An array $A[1..n]$ of n elements.

Output: $A[1..n]$ sorted in nondecreasing order.

for $i \leftarrow 1$ **to** $n - 1$
    $k \leftarrow i$
    for $j \leftarrow i + 1$ **to** n {*Find the ith smallest element.*}
        if $A[j] < A[k]$ **then** $k \leftarrow j$
    end for
    if $k \neq i$ **then** *interchange* $A[i]$ *and* $A[k]$
end for