Recitation: Basic algorithms with arrays
Finding minimum in array of integers

Input:

A // array of integers
n // number of elements in array (array size)

Output:

Min // value of element with smallest value

Minimum(A, n) // name of algorithm and parameters
Min = A[0] // initialize minimum as first element
for i = 1 to n-1 // look at remaining elements
  if A[i] < Min then
    Min = A[i]
  endif
endfor
return Min

endMinimum

// or # introduces comment
what follows is not part of algorithm, it rather explains algorithm
Finding min index in specified range of array of integers

Input:

A // array of integers
i//
j //

Output:

MinIndex // index of element with smallest value
MinIndex(A, i, j) // name of algorithm and parameters
Min = A[i] // initialize minimum as first element
MinIndex = i
for k = i+1 to j-1 // look at remaining elements
    if A[k] < Min then
        Min = A[k]
        MinIndex = k
    endif
endfor
return MinIndex
endMinimum

// or # introduces comment
what follows is not part of algorithm,
it rather explains algorithm
<table>
<thead>
<tr>
<th>n</th>
<th>i</th>
<th>A[i]</th>
<th>Min</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>-2</td>
<td>-1</td>
</tr>
<tr>
<td>3</td>
<td>-4</td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>89</td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>-9</td>
<td>-9</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>-9</td>
<td></td>
</tr>
</tbody>
</table>
• How about find the minimum of a two 2-D array?

```
3  4  8  1
3  0  2  3
2  5  6 -1
9  7  4  5
```
Input:
A //2-D array
m//
n//

Output:
Min // index of element with smallest value
Min2D(A, m, n) // name of algorithm and parameters
Min = Minimum(A[0], n) // initialize minimum as first element
for k = 1 to m-1 // look at remaining elements
temp = Minimum(A[k], n)
if temp < Min then
    Min = temp
endif
endfor
return Min
endMin2D
- Min = 1 initialization
- Min = 0 when $k = 1$
- Min = -1 when $k = 2$
- Min = -1 when $k = 3$

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>4</th>
<th>8</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>6</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
How about finding the maximum?

Input:

- A // array of integers
- n // number of elements in array (array size)

Output:

- Max // value of element with largest value

Maximum(A, n) // name of algorithm and parameters

Max = A[0] // initialize minimum as first element

for i = 1 to n-1 // look at remaining elements

  if A[i] > Max then
    Max = A[i]
  endif

endfor

return Max

endMaximum
<table>
<thead>
<tr>
<th>n</th>
<th>i</th>
<th>A[i]</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>-2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>-4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>89</td>
<td>89</td>
<td>89</td>
</tr>
<tr>
<td>8</td>
<td>-9</td>
<td>89</td>
<td>89</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>89</td>
<td>89</td>
</tr>
</tbody>
</table>
Sorting

• Input: 3, 8, 1, -7, 6
• Expected output: -7, 1, 3, 6, 8
Input:
A // array of integers
n // number of elements in array (array size)

Output:
B // array with elements of A, sorted in ascending order

SortMin(A, n) // idea is to repeatedly extract minimum from original array
for i = 0 to n-1
endfor
for i = 0 to n-2
    i_min = MinIndex(B, i, n)
    Swap(B[i], B[i_min])
endfor
return B
endSortMin

SortMin uses two sub-algorithms
- MinIndex
- Swap
• A: 3, 8, 1, -7, 6
• Since there are 5 elements, in the algorithm there are 5 iterations, in each of which the algorithm makes sure that one more element is in the correct position
• 3, 8, 1, -7, 6 → -7, 8, 1, 3, 6 → -7, 1, 8, 3, 6 → -7, 1, 3, 8, 6 → -7, 1, 3, 6, 8
• Possible applications:
  – Assume each webpage has an importance associated with it
  – The browser may have to return the webpages in the order with descending importance.
Inserting element in sorted array

Input:

A // sorted array of integers (increasing from left to right)
n // number of elements in array (array size)
B // integer to be inserted

Output:

A // sorted array with n+1 elements (original element and B)

InsertSorted(A, n, B) // idea is to find where to insert B and then to insert B

\[ i_{ins} = n \]

for i = 0 to n-1

\[
\text{if } B < A[i] \text{ then }
\]

\[ i_{ins} = i \]

break // It means find the correct position to insert the new element

endfor

for i = n to \[ i_{ins} + 1 \]

\[ A[i] = A[i-1] \]

endfor

A[\[ i_{ins} \]] = B

return A, n+1

endInsertSorted
• Basically, you need to find the correct position, and then move the elements behind that position.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
<td>7</td>
<td>10</td>
<td>44</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>