C programming

Lecture 10 – sorting methods

Based on slides designed by Shay Artsi, Aythan Avior, Gitit Rockstein and Saher Ismir for the department of Computer Science, Technion, Israel Institute of Technology
Translated and updated by Anne Weill-Zrahia

גכתב עליד, טל כהן, עדיר אבראהים. © כל הזכויות שמורות – מרכז טכנולוגי לישראל
Sorting

- **Sorting**: reorganization of a list of numbers so that the result answer a certain order
- **Examples**: greater or equal for numbers, alphabetic for words
- **Main operations in a sort will be**
  1. **Comparison** between 2 values according to the predefined relation
  2. **Exchange** between two values
- In this lecture we will describe some methods for the following problem
  1. Given an array of n values
  2. Goal: sort it so that

\[
a[0], \ a[1], \ldots, \ a[n-1] \\
a[0] \leq a[1] \leq \ldots \leq a[n-1]
\]
Max Sort

- This algorithm has n-1 cycles
- In each cycle the maximum element will be found and the array will be shortened by 1, this maximum element will be transferred to right position
  - Cycle 1: Find maximum element in the array, and exchange it with element a[n-1]
  - Cycle 2: Find maximum element in a[0]-a[n-2] and exchange it with a[n-2]
  - Cycle k: Find maximum element in a[0]-? exchange it

At this point the k greatest elements are in their final places. Continue till all elements have been set in final place(n-1)
Max Sort

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st stage</td>
<td>7</td>
<td>9</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>2nd stage</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>3rd stage</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>4th stage</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>
Max Sort - implementation

```c
/* Helper function: finds the index of maximal element */
int index_of_max(int a[], int m)
{
    int i, i_max = 0;
    for ( i = 1; i < m; i++ )
        if ( a[i_max] <= a[i] )
            i_max = i;
    return i_max;
}

void max_sort(int a[], int n)
{
    int length;
    for ( length = n; length > 1; length-- ) {
        int i_max = index_of_max(a, length);
        swap(&a[i_max], &a[length-1]);
    }
}
```
Bubble sort

• Look at the following code fragment

    ```c
    for ( i = 1; i < m; i++ )
        if ( a[i] < a[i-1] )
            swap(a + (i-1), a + i);
    ```

• Bubble sort performs n-1 stages
• At each stage the largest element is advanced to its final position (in an array which gets smaller and smaller)
• Stage k: the largest element from a[0]...a[n-k] is transferred to position a[n-k]
## Example of bubble sort

<table>
<thead>
<tr>
<th>Stage</th>
<th>Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1&lt;sup&gt;st&lt;/sup&gt; stage</td>
<td>7 9 2 5 3</td>
</tr>
<tr>
<td>2&lt;sup&gt;nd&lt;/sup&gt; stage</td>
<td>7 2 5 3 9</td>
</tr>
<tr>
<td>3&lt;sup&gt;rd&lt;/sup&gt; stage</td>
<td>2 5 3 7 9</td>
</tr>
<tr>
<td>4&lt;sup&gt;th&lt;/sup&gt; stage</td>
<td>2 3 5 7 9</td>
</tr>
<tr>
<td></td>
<td>7 9 2 5 3</td>
</tr>
</tbody>
</table>
/* Helper function: bubble the maximal element up */
void bubble(int a[], int m)
{
    int i;
    for (i = 1; i < m; i++)
        if (a[i] < a[i-1])
            swap(&a[i], &a[i-1]);
}

void bubble_sort(int a[], int n)
{
    int length;
    for (length = n; length > 1; length--)
        bubble(a, length);
}
Bubble Sort - improved implementation

```c
int bubble(int a[], int m) {
    int i, swapped = 0;
    for (i = 1; i < m; i++)
        if (a[i] < a[i-1]) {
            swap(&a[i], &a[i-1]);
            swapped = 1;
        }
    return swapped;
}

void bubble_sort(int a[], int n) {
    int length;
    for (length = n; length > 1; length-- )
        if (!bubble(a, length))
            break;
}
```

- sometimes the array is already ordered
- Or partially ordered
- So that not all the n-1 stages are necessary
- In such cases we can be more efficient by suppressing unnecessary cycles:
Max sort – improved implementation

- As for bubble sort, a way to save on unnecessary cycles can be found
- Idea: count how many times previous element was not the maximum element
- If the exchange has happened “enough times” then the array was already ordered
- Question: how many times should we check this in a stage where the length of the remaining array is m?
- How can the function signal that the remaining array left is ordered and we can finish?
Bubble Sort - improved implementation

```c
/* Helper function: finds the index of maximal element */
int index_of_max(int a[], int m)
{
    int i, i_max = 0, held = 0;
    for ( i = 1;  i < m;  i++ )
        if ( a[i_max] <= a[i] )
            i_max = i;
        else
            held = 1;
    return held ? i_max : -1;
}

void max_sort(int a[], int n)
{
    int length, i_max;
    for ( length = n;  length > 1;  length-- ) {
        if ( (i_max = index_of_max(a, length)) < 0 )
            return;
        if ( i_max < length-1 )
            swap(&a[i_max], &a[length-1]);
    }
}
```
Time complexity of both sorts

- Worst case for **max sort** is $O(n^2)$
- Worst case for **bubble sort** is also $O(n^2)$
- Why? which operations we are counting
- What is the worst case for both examples
- What is the time complexity if the array was ordered “upside down”
- It can be proven that the best possible sort can be obtained in $O(n \cdot \log n)$ (quicksort)