GNG1106
Fundamentals of Engineering Computation

Instructor: **Hitham Jleed**

University of Ottawa

Fall 2023 ~

Outline

- Sorting is one of the most basic procedures often required in computing.
- For a set of objects of any kind, as long as it is possible to compare every two of them, it is possible to sort them.
  - A set of numbers can be sorted according to their values.
  - A set of products can be sorted according to their prices.
  - A set of professors can be sorted according to their ratings on `http://www.ratemyprofessors.com`
  - ...
- We will use sorting a set of integers as an example to formally introduce two sorting algorithms.
- Our objective is to sort an array of n integers (which will be denoted by `A`) in ascending order, namely, from low to high.
- The introduced two algorithms have complexity in the order of n^2, which are actually not the most efficient. The most efficient algorithms have complexity order $n \log n$.

Replacement Sort

Idea: run a number of passes.

- 1st pass:
  find the index `k` of the largest value in `A`;
  swap `A[k]` with the last element of the array.
- 2nd pass:
  find the index `k` of the largest value in sub-array of `A` containing the first $n-1$ elements;
  swap `A[k]` with the last element of the sub-array.
- 3rd pass:
  find the index `k` of the largest value in sub-array of `A` containing the first $n-2$ elements;
  swap `A[k]` with the last element of the sub-array.
- ...
- Last pass:
  find the index `k` of the largest value in sub-array of `A` containing the first 2 elements;
  swap `A[k]` with the last element of the sub-array.

Build the Algorithm

- Obviously, we will need loop over passes.
  - How many passes? $n-1$.
- What do we do in each pass?
  - In the i^{th} path, go through a sub-array of length $n-i+1$ to find the index `k` of the largest value in the sub-array.
    - If the passes are indexed from 0 ("zero-based indexing) rather than from 1, the sub-array length will be $n-i$. I will use this convention!
  - Swap `A[k]` with the last element of the sub-array, which is `A[n-i-1]` (under the zero-based indexing convention)
- We also need to implement "finding the index of the largest element in an (sub-)array."

Highlight

Replacement Sort Algorithm: See Code.

Bubble Sort

Idea: run a number of passes over the array

- 1st pass: slide a two-element window through the array `A`; whenever the first element in the window is greater than the second element, swap the two elements.
- 2nd pass: slide a two-element window through the sub-array of `A` containing $n-1$ elements; whenever the first element in the window is greater than the second element, swap the two elements.
- 3nd pass: slide a two-element window through the sub-array of `A` containing $n-2$ elements; whenever the first element in the window is greater than the second element, swap the two elements.
- ...
- Last pass: slide a two-element window through the sub-array of `A` containing 2 elements; whenever the first element in the window is greater than the second element, swap the two elements.

Watch an animation: `https://en.wikipedia.org/wiki/Bubble_sort`

Build the Algorithm

- We will need to a loop over passes over the array
  - How many passes? $n-1$
- What do we do in each pass?
  - In the i^{th} pass (under zero-based indexing), slide a two-element window through the sub-array of `A` containing $n-i$ elements; whenever the first element in the window is greater than the second element, swap the two elements.
  - In each pass, "sliding the window through and swapping" needs to be implemented by a loop (which could be packaged into a function).
    - How many positions will the window will be located at? when the sub-array has size $n-i$ (namely, in the i^{th} pass), the number of window positions will be $n-i-1$.

Highlight

Bubble Sort Algorithm: See Code

Improved Bubble Sort

- Bubble sort may be made to stop earlier without going through all the passes.
- Suppose that in some pass, no swapping is needed. Then at the end of this pass, we know that the array has already been sorted. Then no further passes are needed.
- To implement this, what we need to do is adding a flag variable, say `swapDone`, to keep track of whether a swapping has occurred in a pass. The variable will then be used as an additional condition (besides the pass index) to control exiting from the loop.
  - Carefully design the initial value of `swapDone` in each pass
  - If the loop over passes is a `while`-loop, make sure the loop is entered.

Highlight

Improved Bubble Sort Algorithm: See Code

- These introduced sorting algorithms may be applied to sort any objects (i.e., variables of some structure type).
  - Sort rectangles according to is area
  - Sort students according to their lastnames
  - Sort professors according to their "toughness rating".
- All it requires is a means/rule of comparing two objects of that type, which can be implemented as a function and called by the sorting algorithm.
- Example, sorts an array of RECTANGLE's, where a smaller rectangle (in area) precedes a larger rectangle.
- We will develop the following function:

  ```
  int shouldAPrecedeB(RECTANGLE A, RECTANGLE B);
  ```

  or

  ```
  int shouldAPrecedeB(RECTANGLE *pA, RECTANGLE *pB);
  ```

  where the function returns 1 (True) if `A` precedes `B` (or `*pA` precedes `*pB`, in the second case) after sorting, and 0 (False) otherwise. Obviously, the function should be designed by comparing the areas of the two rectangles.

Coding Demonstration