Bubble sort method

Someone gives us a vector of unsorted numbers. We want to obtain the vector sorted in ascending order.

1. Assign the `IndexOfTheLastToCheck` to be the index of the vector end.
2. Compare the 2 consequent elements starting from the beginning till we reach the `IndexOfTheLastToCheck`.
3. If the left element is larger than the right one, we swap these 2 elements.
4. Move to the next pair to the right, i.e., move to the item 2.
   - Notice that at the end of the sweep, the index of the last element to check holds the largest element.
   - So, the next sweep is shorter by one element.
5. Decrease `IndexOfTheLastToCheck` by 1
6. If `IndexOfTheLastToCheck` > 1, repeat from the second step.

Bubble sort properties

- The execution time of this algorithm is $O(N^2)$
- This is the worst of all working algorithms!
- Never use it in real life!
- However, it is quite intuitive and a very simple to program.
- It does not require extra memory during the execution.

Quick sort method

A much better, yet still simple algorithm. We will discuss the recursive realization. The name of our sorting function is `qsort`.

1. Choose a pivot point value
   - let’s choose the pivot at the middle of the vector
   - `pivotIndex=ceil(N/2)`
   - `pivotValue=x(pivotIndex)`
2. Create two vectors which hold the lesser and larger than `pivotValue` elements of the input vector.
3. Now, concatenate the result as `xs=[qsort(lesser), pivotValue, qsort(larger)]`
4. The sorting is done.
Quick sort summary

- It is very easy to implement.
- It is usually fast.
- A typical execution time is $O(N \log_2 N)$.
- This is not guaranteed.
  - For certain input vectors the execution time could be as long as $O(N^2)$.

Heap

The heap is a structure where a parent element is larger or equal to its children.

The top most element of a heap is called the root.

Heap sorting method

1. Fill the heap from the input vector elements.
   - Take an element and place it at the bottom of the heap.
   - Sift-up (bubble up) this element.
   - Do the same with every following element.
2. Remove the root element, since it is the largest.
3. Rearrange the heap i.e. sift-down.
   - Take the last bottom element.
   - Place it at the root.
   - Check if parent is larger then children.
     - Find the largest child element.
     - If the largest child is larger then parent, swap them and repeat the check in the sub heap of this child element.
4. Repeat step 2 until no elements are left in the heap.

The heap sorting complexity is $O(N \log_2 N)$.

Filling (sift-up) the heap

Step 1
Place a new element at the bottom of the heap.
Filling (sift-up) the heap

Step 2
Check if the parent is larger then the child. If so, swap them and repeat the step 2.

Notes

Removing from the heap (sift-down) the heap

Step 1
Remove the root element.
Removing from the heap (sift-down) the heap

**Step 2**

Place the last element of the heap to the root position.

```
11 6
9 8
5 4 4 6
5 3
2 4
15
```

**Step 3**

Check if the parent is smaller than the largest child. If so, swap and repeat the step 3, otherwise go to the step 1.

```
4 6
9 8
5 4 4 6
5 3
2
15
11
```

The sequence repeats.

**Step 1**

Remove the root element

```
9 6
4 8
5 4 4 6
5 3
2
15
11
```
The vector heap representation

- Heap nodes are numbered consequently. These numbers represent the nodes positions in the vector (i.e., the linear array).
- Notice that the parent and its children have a very simple relationship:
  - If a parent node index is \( i \), the 1st child index is \( 2i \).
  - The 2nd child index is \( 2i+1 \).
  - If we know a child index (\( i \)), the parent index is \( \text{floor}(i/2) \).

Matlab built-ins 'issorted' and 'sort'

An easy check if an array is sorted can be done with \texttt{issorted} which returns \texttt{true} or \texttt{false}.

```matlab
>> x=[1,2,3];
>> issorted(x)
an\s = 1
```

\texttt{issorted} checks only for the \texttt{ascending} order, for example.

```matlab
>> x=[3,2,1];
>> issorted(x)
an\s = 0
% Recall that '0' is equivalent of false in Matlab
```

Also, if you want to sort an array, the Matlab has the \texttt{sort} function to do it.

```matlab
>> sort([5,3,2])
an\s = 2 3 5
```