Sorting

Eugeniy E. Mikhailov

The College of William & Mary

Lecture 28

Bubble sort method

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

- Assign the `IndexOfTheLastToCheck` to be the *index* of the vector end.

1. Compare the 2 consequent elements starting from the beginning till we reach the `IndexOfTheLastToCheck`.
2. If the left element is larger than the right one, we swap these 2 elements.
3. 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.
   - I.e., the *index* of the last element to check should be decreased by 1.
4. Decrease `IndexOfTheLastToCheck` by 1
5. If `IndexOfTheLastToCheck` > 1, repeat from the second step.

$x = [3, 1, 4, 5, 2]$
the first sweep
$x = [\widehat{3, 1}, 4, 5, 2]$ swap
$x = [1, 3, 4, 5, 2]$ after swap
$x = [1, \widehat{3, 4}, 5, 2]$ no swap
$x = [1, 3, \widehat{4, 5}, 2]$ no swap
$x = [1, 3, 4, \widehat{5, 2}]$ swap
$x = [1, 3, 4, 2, 5]$ sweep is done
new sweep
$x = [\widehat{1, 3}, 4, 2, 5]$ no swap
$x = [1, \widehat{3, 4}, 2, 5]$ no swap
$x = [1, 3, \widehat{4, 2}, 5]$ swap
$x = [1, 3, 2, 4, 5]$ sweep is done
new sweep
$x = [\widehat{1, 3}, 2, 4, 5]$ no swap
$x = [1, \widehat{3, 2}, 4, 5]$ swap
$x = [1, 2, 3, 4, 5]$ sweep is done
the last sweep
$x = [\widehat{1, 2}, 3, 4, 5]$ no swap
$x = [1, 2, 3, 4, 5]$ we are done

Bubble sort properties

- The execution time of this algorithm is $\mathcal{O}(N^2)$
- 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`.

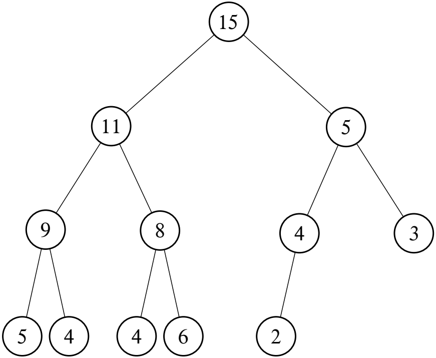
- Choose a pivot point value
  - let's choose the pivot at the middle of the vector
  - pivotIndex=floor(N/2)
  - pivotValue=x(pivotIndex)
- Create two vectors which hold the lesser and larger than pivotValue elements of the input vector.
- Now, concatenate the result as
  xs=[`qsort`(lesser), pivotValue,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

Quick sort summary

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

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

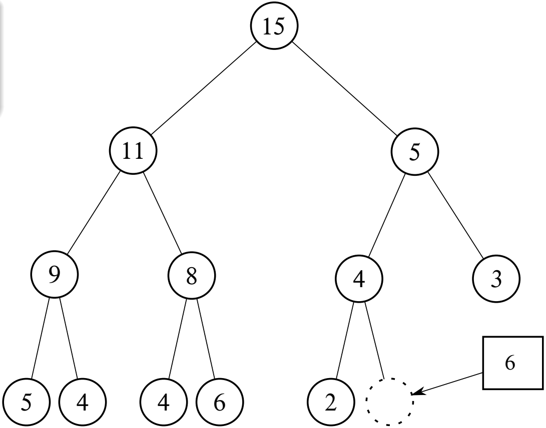
1. Fill the heap from the input vector elements.
   1. Take an element and place it at the bottom of the heap.
   2. Sift-up (bubble up) this element.
   3. Do the same with every following element.
2. Remove the root element, since it is the largest.
3. Rearrange the heap i.e. sift-down.
   1. Take the last bottom element.
   2. Place it at the root.
   3. Check if parent is larger then children.
      1. Find the largest child element.
      2. 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 $\mathcal{O}(N \log_2 N)$.

Filling (sift-up) the heap

Step 1

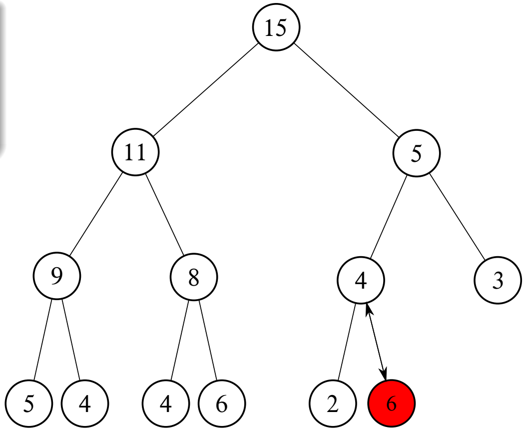
Place a new element at the bottom of the heap.



Filling (sift-up) the heap

Step 2

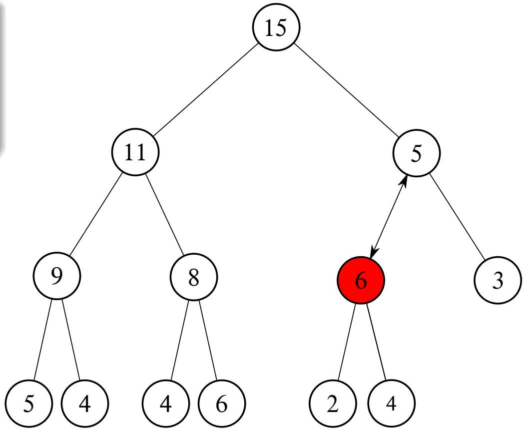
Check if the parent is larger then the child. If so, swap them and repeat the step 2.



Filling (sift-up) the heap

Step 2

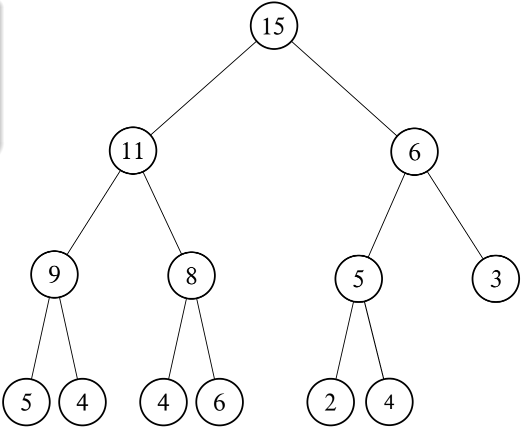
Check if the parent is larger then the child. If so, swap them and repeat the step 2.



Filling (sift-up) the heap

Step 2

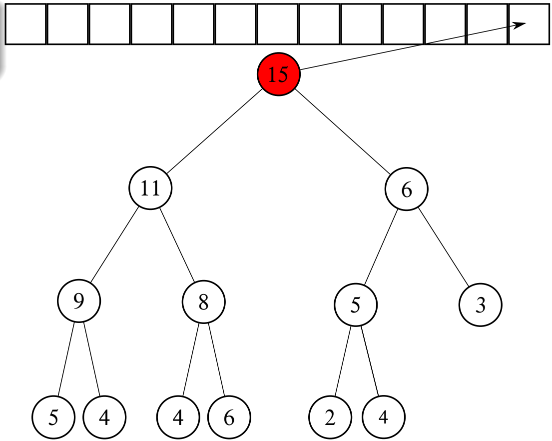
Check if the parent is larger then the child. If so, swap them and repeat the step 2.



Removing from the heap (sift-down) the heap

Step 1

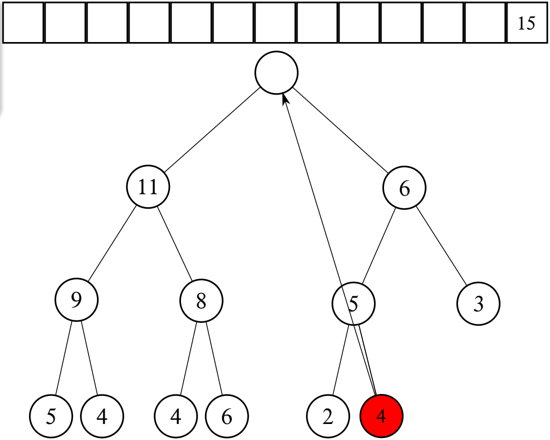
Remove the root element.



Removing from the heap (sift-down) the heap

Step 2

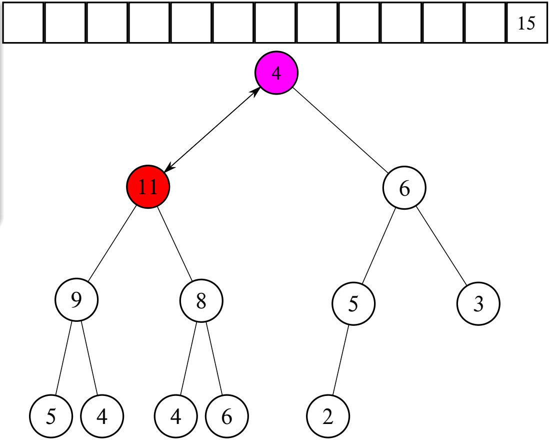
Place the last element of the heap to the root position.



Removing from the heap (sift-down) the heap

Step 3

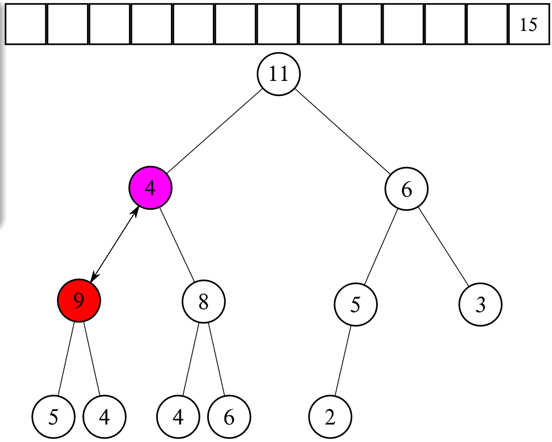
Check if the parent is smaller than the largest child. If so, swap and repeat the step 3, otherwise go to the step 1.



Removing from the heap (sift-down) the heap

Step 3

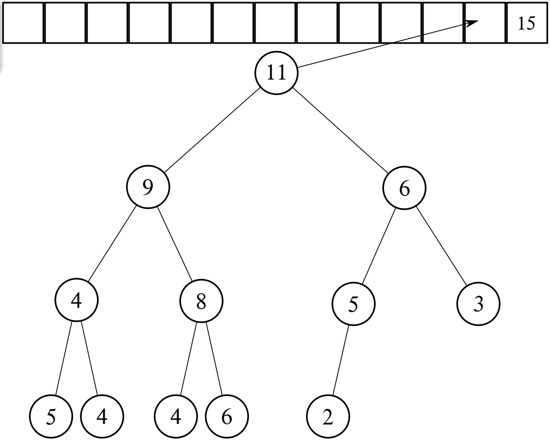
Check if the parent is smaller than the largest child. If so, swap and repeat the step 3, otherwise go to the step 1.



The sequence repeats.

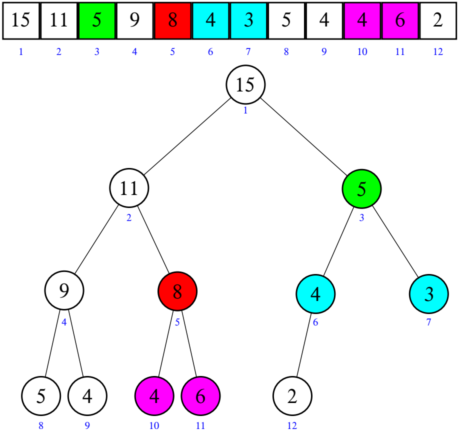
Step 1

Remove the root element



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`) then
    - the parent index is `floor(i/2)`



Matlab built-ins 'issorted' and 'sort'

An easy check if an array is sorted can be done with `issorted` which returns `true` or `false`.

```
>> x=[1,2,3];
>> issorted(x)
ans = 1
```

`issorted` checks only for the ascending order, for example

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

Also, if you want to sort an array, the Matlab has the `sort` function to do it.

```
>> sort([5,3,2])
ans = 2     3     5
```