Binary heaps: homework (17/3/2020)

Gaia Saveri

1. 2. 3.

The first three exercises were implemented in a live session during Lessons 6, 7 and 8. I report in the folder `AD_binheaps` the code we developed, containing the required functions (Ex. 1-2). Compiling the code will produce two executables, namely `test_insert` and `test_delete_min`, which contain the required tests (Ex. 3).

The following plot shows the difference in the execution time in the task of removing the minimum in both a binary heap and an array:

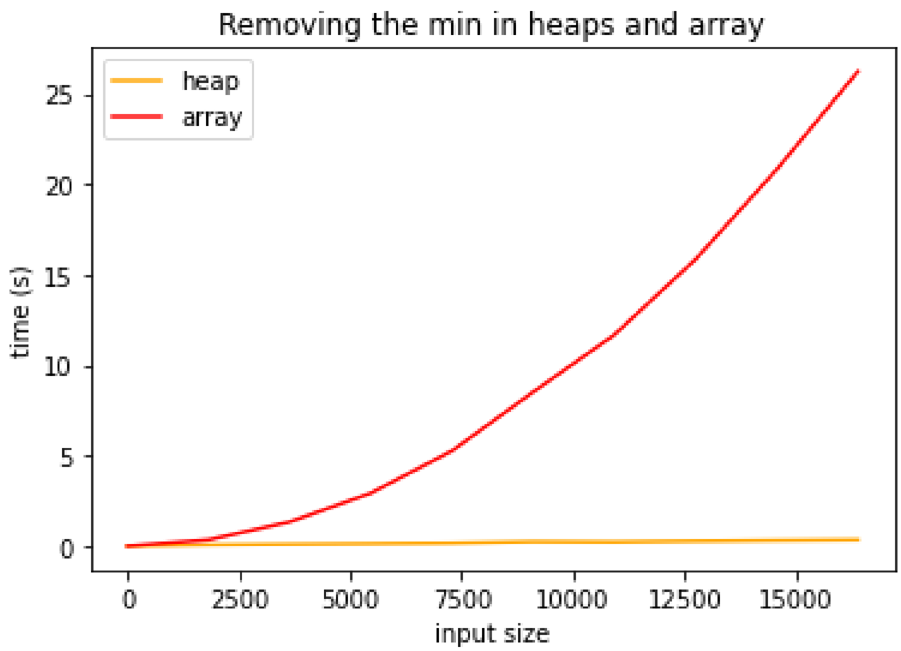

Figure 1: remove min in binary heap and array: comparison of execution time

As expected, an heap data structure is highly more suitable for solving this problem.

4. Ex. 6.1-7 in *Introduction to algorithmic design*

With the array representation of binary heaps, if a heap has n nodes, then for sure the last leaf is the n-th element (it is indexed by n). Hence its parent would be indexed by $\lfloor(\frac{n}{2})\rfloor$ (indeed it is $\frac{n}{2}$ if the node is left child, $\frac{n}{2}-1$ if it is right child). So the node indexed at $\lfloor (\frac{n}{2}) \rfloor$ is not a leaf node. Now consider the successive node, namely the one indexed by $\lfloor (\frac{n}{2}) + 1 \rfloor$, this cannot be a parent node, indeed if it was, than its first child, the let one, would be indexed at $2 \cdot \lfloor (\frac{n}{2}) \rfloor + 1$, which is out of bounds for an array of n elements. So necessarily the first leaf is indexed by $2 \cdot \lfloor (\frac{n}{2}) + 1 \rfloor$.

5. Ex. 6.2-6 in *Introduction to algorithmic design*

The worst case scenario is that we put in the root node a node's key which is $\succeq$ wrt all the other nodes in both the left and the right subtrees. In this case HEAPIFY will be called recursively until a leaf is reached. To make the recursive calls traverse the longest path, we should choose a value (that depends on the order relation $\preceq$) which makes HEAPIFY to be called always on the left subtree (because of the topology of binary heaps). In this case, given h the height of the binary-heap (i.e. number of edges in the longest path from the root to a leaf), HEAPIFY is called h times. Since a single call costs $\Theta(1)$, then h calls cost $\Theta(h) = \Theta(\log_2(n))$, being n the number of nodes in the heap. Since $\Theta(\log_2(n)) = O(\log_2(n)) \cap \Omega(\log_2(n))$, we have that the worst case running time is $\Omega(\log_2(n))$.

6. Ex. 6.3-3 in *Introduction to algorithmic design*

Proceed by induction.

Base case: at level $h = 0$, meaning the level of the leaves, there are $\lceil \frac{n}{2} \rceil$ nodes (proved in exercise 4).

Inductive step: Assume the thesis is valid nodes of height $h-1$. Remove from the original binary heap all its leaves, so that nodes of height h in the original tree have now height $h-1$. Now the new obtained tree has $n - \lceil \frac{n}{2} \rceil = \lfloor \frac{n}{2} \rfloor$ nodes. So, by inductive hypothesis, we have that the number of nodes at height $h-1$ (in the new tree, hence of level h in the old original tree) is $\left\lceil \left\lfloor \frac{\frac{n}{2}}{2^{h-1+1}} \right\rfloor \right\rceil < \left\lceil \frac{\frac{n}{2}}{2^h} \right\rceil = \lceil \frac{n}{2^{h+1}} \rceil$.

Thus we proved the thesis.