Luvisutto Alberto, Algorithmic design exam

Homework: Binary Heaps

Ex.1

Implement the array-based representation of binary heap together with the functions `heap_min`, `remove_min`, `heapify`, `build_heap`, `decrease_key` and `insert_value`.

All the previous functions have been implemented during the lectures of the course. They are reported in the `AD_bin_heaps/src/binheap.c` file.

Ex.2

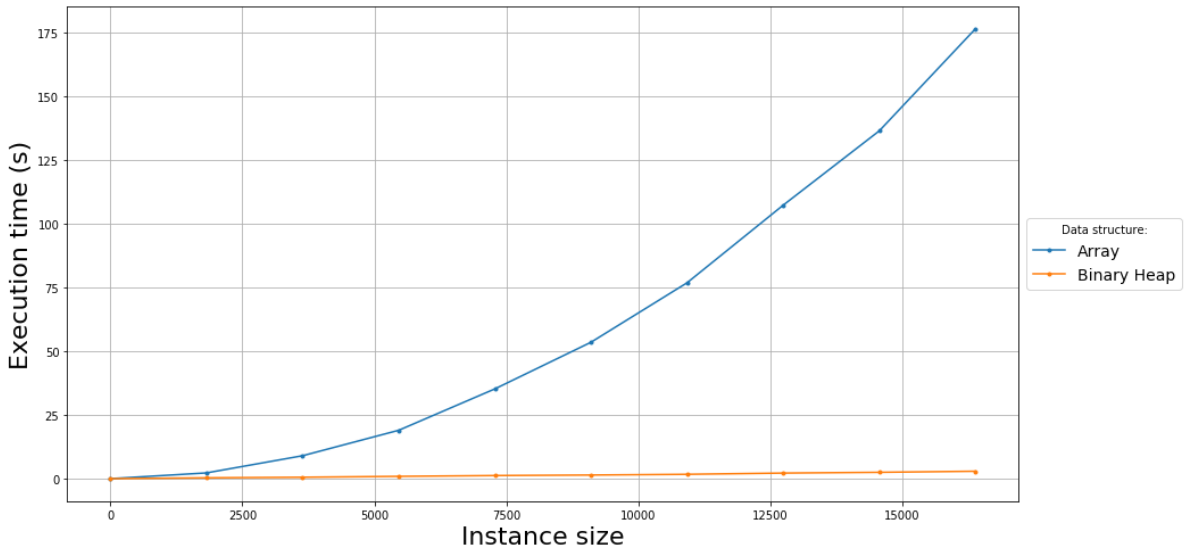
Implement an iterative version of `heapify`

Again, an iterative version of `heapify` has been implemented during the lectures of the course and can be found in the previously mentioned file.

Ex.3

Test the implementation on a set of instances of the problem and evaluate the execution time.

A test has been performed using the already implemented test `test_delete_min`. The following plot reports the results obtained running it by using the two different data structures:



It is clearly visible that the Binary Heap version is a lot faster with respect to the Array version.

Ex.4

Show that, with the array representation, the leaves of a binary heap containing *n* nodes are indexed by $\lfloor \frac{n}{2} \rfloor + 1, \lfloor \frac{n}{2} \rfloor + 2, \ldots, n$.

Assume by contradiction that the node indexed by $\lfloor \frac{n}{2} \rfloor + 1$ is not a leaf.

This means that this node has at least one child (the left one), indexed by $(\lfloor \frac{n}{2} \rfloor + 1) * 2$

But this means that the left-child index is

$$\left(\left\lfloor \frac{n}{2} \right\rfloor + 1\right) * 2 > \left(\frac{n}{2} - 1 + 1\right) * 2 = n$$

Which is obviously a contradiction, since we know that the binary heap contains *n* nodes. The same result is obtained if we apply the same procedure to all the nodes with higher indexes. In fact, in a binary heap, if the node indexed by $\left\lfloor \frac{n}{2} \right\rfloor + 1$ is a leaf, all the nodes with higher indexes are leaves too.

To finish the proof we have to prove that the node indexed by $\left\lfloor \frac{n}{2} \right\rfloor$ is not a leaf. As before, it is obvious to observe that:

- if *n* is even, this node has one child (the left one), indexed by $\left\lfloor \frac{n}{2} \right\rfloor * 2 = \frac{n}{2} * 2 = n$
- if *n* is odd, this node has for sure two children, indexed by $\left\lfloor \frac{n}{2} \right\rfloor * 2 = \frac{n-1}{2} * 2 = n - 1$ (left child) and by $\left\lfloor \frac{n}{2} \right\rfloor * 2 + 1 = \frac{n-1}{2} * 2 + 1 = n$ (right child)

We can then conclude that this node is not a leaf. This concludes the proof.

Ex.5

Show that the worst-case running time of `heapify` on a binary heap of size *n* is $\Omega(\log n)$. (Hint: For a heap with *n* nodes, give node values that cause `heapify` to be called recursively at every node on a simple path from the root down to a leaf.)

Let's assume for simplicity that we are working with a min heap. This means that the worst-case scenario for a call of `heapify` would happen if we had the maximum value of the heap on the root.

Let's also assume that we have a complete binary tree, with height $h = \log n$.

It can be easily proven that, having to move the maximum value from the root to a leaf of the tree, we call the function at least once for each level of the tree, performing a swap each time (since the value we are moving is the maximum one). This brings to a running time of $\Omega(h) = \Omega(\log n)$.

If we instead don't assume that the tree is complete, we are in the general case in which the tree is almost complete, so it is complete up to the second last level. In this case, the worst-case scenario is having the smallest possible values on the longest branch, which means that the algorithm has to bring the value down to the leaves on one of the branches of "length" h. This brings again to the problem solved before.

Ex.6

Show that there are at most $\left\lceil \frac{n}{2^{h+1}} \right\rceil$ nodes of height *h* in any *n*-element binary heap

Let's prove this statement by induction.

Base case: $n_0 = \left\lceil \frac{n}{2^{0+1}} \right\rceil = \lceil \frac{n}{2} \rceil$

This is true due to the statement we proved in Ex.4. In fact, we proved that the leaves in an n-element binary heap are indexed by $\lfloor \frac{n}{2} \rfloor + 1, \lfloor \frac{n}{2} \rfloor + 2, \ldots, n$.

It is easy to prove, given the previous statement, that the number of these nodes is $n - \lfloor \frac{n}{2} \rfloor = \lceil \frac{n}{2} \rceil$.

Inductive case: given $n_{h-1} = \left\lceil \frac{n}{2^{h-1+1}} \right\rceil = \left\lceil \frac{n}{2^h} \right\rceil$ with $h > 1$, we want to prove that $n_h = \left\lceil \frac{n}{2^{h+1}} \right\rceil$.

Since a binary heap is a nearly complete binary tree we know that, excluding the last level of the tree (with height $h = 0$, considered in the base case), each level is full.

Also, since we are considering n_{h-1}, we are for sure not considering the root, otherwise the upper level h wouldn't even be defined.

This means that, for sure, n_{h-1} is even, and since each node can have at most two children, we know that

$$n_h = \frac{n_{h-1}}{2} = \frac{1}{2}\left\lceil \frac{n}{2^h} \right\rceil = \left\lceil \frac{n}{2^{h+1}} \right\rceil$$

and this ends the proof.