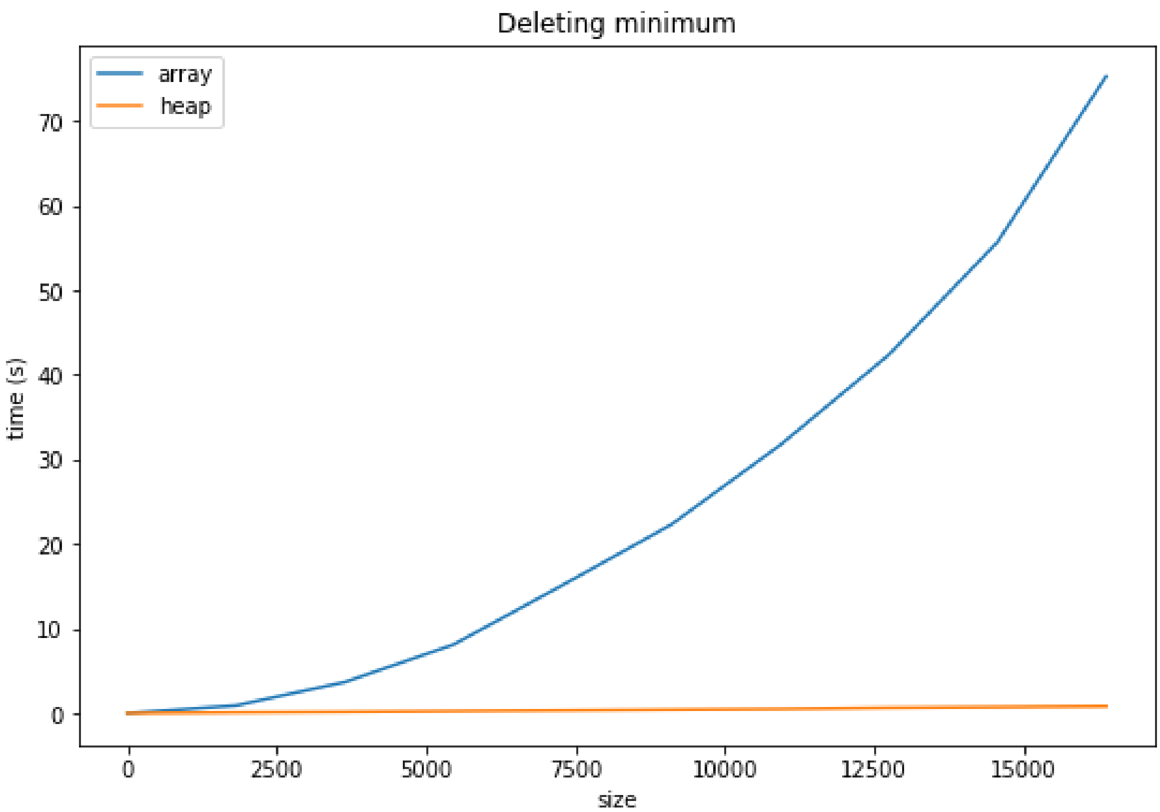
# Heaps: homework 1

### Ex. 2

An iterative version of `HEAPIFY` is implemented in the source file `binheap.c`.

### Ex. 3




We can see that the extraction of the minimum is much faster on a heap than it is on an array, with differences that are already noticeable for small sizes (the array time is $10\times$ the heap time for $size = 1820$).

### Ex. 4

In a heap represented with an array, all the nodes of the two lowest levels after the father of the last node are leaves. So, if there are $n$ nodes, since the index of the father of the last node is $\lfloor \frac{n}{2} \rfloor$, the leaves have indexes between $\lfloor \frac{n}{2} \rfloor + 1$ and $n$.

### Ex. 5

The worst-case running time of `HEAPIFY` is $\Omega(log_2 n)$. In fact, if the maximum element is on the root of a min-heap, it has to be swapped through the entire heap until it becomes a leaf. `HEAPIFY` then has to be called recursively $h = log_2 n$ times before the restoration of heap property.

## Ex. 6

In an $n$ elements heap the number of nodes at height $h$ is at most $\lceil \frac{n}{2^{h+1}} \rceil$.

From exercise 4 we know that there are $n - \lfloor \frac{n}{2} \rfloor = \lceil \frac{n}{2} \rceil$ leaves (which have $h = 0$), hence the proposition is true for $h = 0$.

Now let's assume that the proposition holds for $h = i - 1$. If the $\lceil \frac{n}{2} \rceil$ leaves are removed, a new heap with $\lfloor \frac{n}{2} \rfloor$ nodes is created and the nodes previously at height $i$ are now at height $i - 1$, then their number is at most $\lceil \frac{\lfloor \frac{n}{2} \rfloor}{2^{(i-1)+1}} \rceil < \lceil \frac{\frac{n}{2}}{2^i} \rceil = \lceil \frac{n}{2^{i+1}} \rceil$ .