Binary Heaps (2): Homework Solutions

Emanuele Ballarin

July 13, 2020

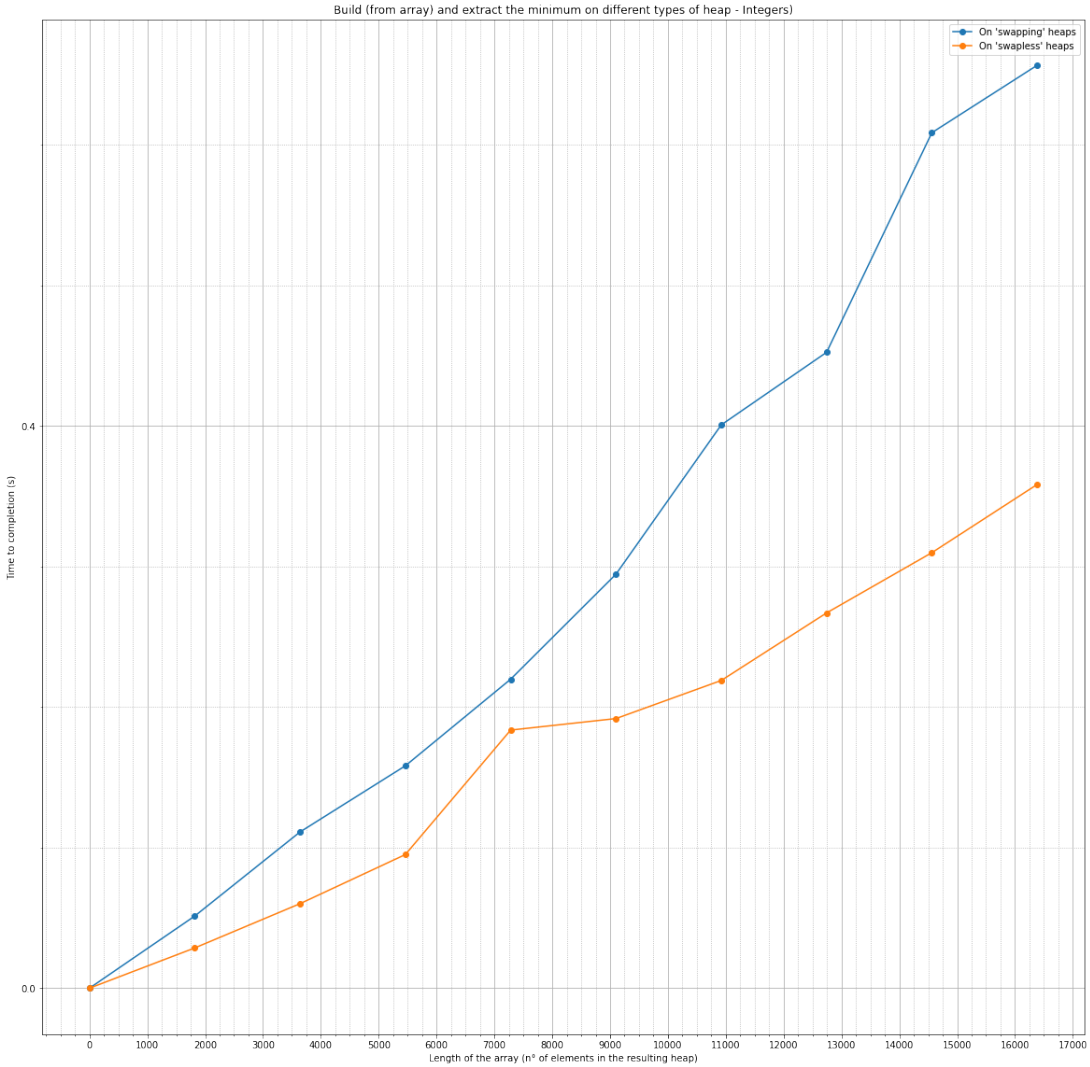
Exercise 1

Text:
By modifying the code written during the last lessons, provide an array-based implementation of binary heaps which avoids to swap the elements in the array A.
(Hint: use two arrays, key pos and rev pos, of natural numbers reporting the position of the key of a node and the node corresponding to a given position, respectively).

Solution:
The *new* version of the *binary heap* has been implemented by following the hint provided, trying to modify the least possible the code already in place.

Benchmarks:

As it is possible to see from the graph – though partially showing the effects of measurement noise – the *swapless heap* implementation is consistently faster across the entire set of heap sizes.



Exercise 2

Text:
Consider the next algorithm:

```
Ex2(A):
  D ← build(A)
  while ¬ is_empty(D) do
     extract_min(D)
  end while
```

where `A` *is an array. Compute the time-complexity of the algorithm when*:

1. `build`, `is_empty` $\in \Theta(1)$, `extract_min` $\in \Theta(|D|)$;
2. `build` $\in \Theta(|A|)$, `is_empty` $\in \Theta(1)$, `extract_min` $\in O(\log |D|)$;

Solution:

The total time-complexity of the algorithm, in both cases, can be seen as the sum of the complexity of the `build` routine and that of the entire **while** loop repeated some times.

The latter, specifically, can be seen as the sum of the complexity of the *check* function `is_empty` that verifies if the loop must be executed, and that of the `extract_min` function. The number of repetitions is such that the *check* function is repeated always one time more than the instructions in he loop (the time the loop needs not to be executed).

To better frame the situation, we can make our assumptions more explicit, by stating that:

- $|A| = |D|$;
- The `extract_min` function removes one element from `D`, thus decreasing its cardinality by one.

This set of assumptions is compliant with the hypothetical situation in which `A` is an array and `D` a heap built from it.

With respect to *case* 1, the total time-complexity of the algorithm is determined by $\Theta(1) + (|D| + 1)\Theta(1) + \sum_{i=1}^{|D|} \Theta(|D|) \sim \Theta(|D|^2)$.

With respect to *case* 2, the total time-complexity of the algorithm is determined by $\Theta(|D|) + (|D| + 1)\Theta(1) + \sum_{i=1}^{|D|} O(log(|D|)) \sim O(|D|log(|D|))$.