Homework 2: Binary Heaps(1)

Azza M. N. Abdalghani

Exercise 1

Implement the array-based representation of binary heap together with the functions `HEAPMIN`, `REMOVE MIN`, `HEAPIFY`, `BUILD HEAP`, `DECREASE KEY` and `INSERT VALUE`.

As they were implemented during the lectures, the implementation can be found in `AD_bin_heaps/src/binheap.c` file.

Exercise 2

Implement an iterative version of `HEAPIFY`.

Same as previous question, it can be found in the same file `AD_bin_heaps/src/binheap.c` at `heapify` function.

Exercise 3

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

Figure 1 shows a comparision between the elapsed time for each array and heap on different input sizes, and it can be observed that `heap` is much much faster than `array`.

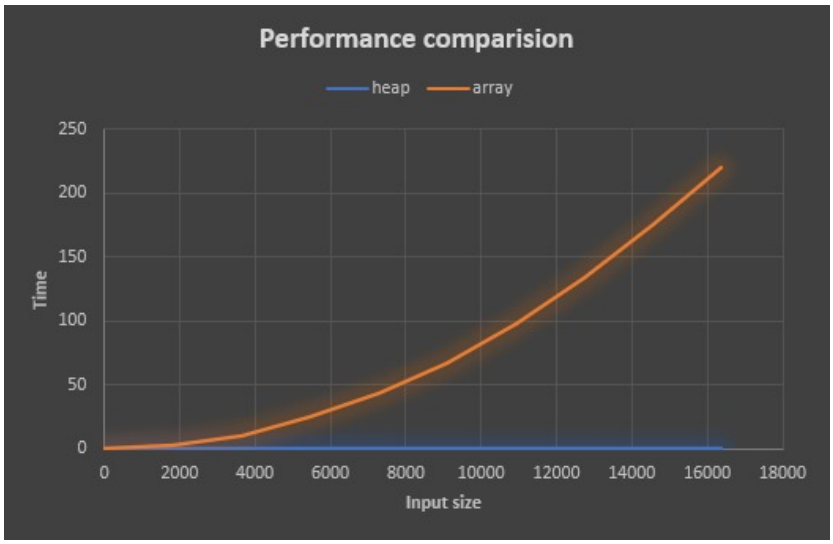

Figure 1: Compare Execution time of array and heap

Exercise 6.1-7

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

To prove that, let's suppose that we have a node i indexed by $\lfloor n/2 \rfloor + 1$, then the left child for this node will be indexed at $(2 * i)$, which is in this case :

$$\texttt{LEFT}(\lfloor n/2 \rfloor + 1) = 2(\lfloor n/2 \rfloor + 1) > 2(n/2 - 1) + 2 = n - 2 + 2 = n.$$

We can see that the index of the LEFT child is greater than the number of the elements (n) in the heap -out of the scope of the heap-. Thus we can conclude that this node i does not have children, which means it is a leaf node. And, this goes for other nodes with larger indicies.

Exercise 6.2-6

Show that the worst-case running time of `MAX_HEAPIFY` on a heap of size n is $\Omega(\log_2 n)$. (Hint: For a heap with n nodes, give node values that cause `MAX_HEAPIFY` to be called recursively at every node on a simple path from the root down to a leaf).

As mentioned in the hint, this worst-case can happen when the `MAX_HEAPIFY` is called once at each level in the heap. For example, considering the leftmost

path in the heap A , if the root A[1] = 1 (or having the smallest value) and all other nodes have larger value than the root, e.g A[i] = 2 for $i \in [2, n]$, then if we call `MAX_HEAPIFY` , it will be invoked once at each level to swap the value of the root until it becomes a leftmost leaf. Moreover, as we know that the height of the heap is $\lfloor \log_2 n \rfloor$, then the time complexity for this case is $\Omega(\log_2 n)$.

Exercise 6.3-3

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

Referring to the first exercise (Exercise 6.1-7), an element indexed by $\lfloor n/2 \rfloor$ is not a leaf, its LEFT child will be indexed at n (if n is an even number) or at n-1 (if n is an odd number), and its RIGHT child with index n at both cases. In addition, this makes the number of the leaves in a heap of size n equals to $\lceil n/2 \rceil$.

So, we can prove that by induction on h :

- Base : when h = 0 , then the number of leaves = $\lceil n/2 \rceil = \lceil n/2^{0+1} \rceil$.
- Step: Now assume that it holds for nodes of height $h - 1$, consider a tree B with removed leaves, then the number of nodes of that tree is $n - \lceil n/2 \rceil = \lfloor n/2 \rfloor$ nodes. Note that the nodes of height h in the old tree have a height $h - 1$ in the new one (B).

  Then, by the inductive assumption, the number of nodes with height $h-1$ in B is :

$$N_{h-1} = \lceil \frac{\lfloor n/2 \rfloor}{2^{h-1+1}} \rceil < \lceil \frac{(n/2)}{2^h} \rceil = \lceil \frac{n}{2^{h+1}} \rceil$$

  which is also the number of nodes with height h in the old tree.