4041 Homework 1

Fletcher Gornick

September 20, 2021

6.1

6.1-1

What are the minimum and maximum numbers of elements in a heap of height h?

Starting with the max number of nodes, we first know that a tree of height 0 has 1 node. Also, any time height goes up by 1, we add double the nodes of the previous level, so the number of nodes is $\sum_{k=0}^{h} 2^k = 2^{h+1} - 1$. Since there are 2^h nodes in the bottom level of a full heap of height h, and only 1 in the most incomplete heap, then there are $2^h - 1$ less nodes, so the least number of elements is $(2^{h+1} - 1) - (2^h - 1) = 2^h$.

6.1-2

Show that an $n-$element heap has height $\lfloor \lg n \rfloor$.

From the last problem, we found that the number of nodes in a heap must be in this interval: $2^h \leq n \leq 2^{h+1} - 1$. Since $\lg(2^h) = h$, and $\lg(2^{h+1} - 1) < \lg(2^{h+1}) = h + 1$, we know that $\lfloor \lg(2^h) \rfloor = \lfloor \lg(2^{h+1} - 1) \rfloor = h$ as well as for any value n between them.

6.1-3

Show that in any subtree of a max-heap, the root of the subtree contains the largest value occurring anywhere in that subtree.

The property of a max-heap is that the value of the root is greater than the values of it's children nodes. We can treat the children nodes as roots of their own subtrees, and follow this logic until we reach leaves. And since the children of a node must be less than the parent, it follows that all the descendents of the root of a subtree must also be less than the root. This is a transitive property by the way.

6.1-4

Where in a max-heap might the smallest element reside, assuming that all elements are distinct?

While it is guarunteed by the property of a max heap that children nodes are smaller than parents, there's no rule among nodes on the same level, so all we can be sure of is that the smallest node will be a leaf, because if the smallest node had children, then they must be smaller which is a contradiction.

6.1-5

Is an array that is in sorted order a min-heap?

Yes, Assuming the heap is constructed where a node is `arr[i]`and it's child nodes are `arr[2i]`and `arr[2i+1]`. Under this assumption, the property of a min-heap must be satisfied, because `arr[2i]`and `arr[2i+1]`must be greater than (or equal to) `arr[i]`for all $i \geq 1$.

6.1-6

Is the array with values $\langle 23, 17, 14, 6, 13, 10, 1, 5, 7, 12 \rangle$ a max heap?

No. `arr[4] = 6`, this value has two child nodes, `arr[4*2] = arr[8]` and `arr[4*2+1] = arr[9]`. Since `arr[9] = 7` $>$ `arr[4] = 6`, this array cannot represent a max-heap because the property of a max-heap has been violated (parent node is less than child node).

6.1-7

Show that, with the array representation for storing an n-element heap, the leaves are the nodes indexed by $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, ..., n$.

In order to show that the listed indexes represent leaves, we must show that starting from $\lfloor n/2 \rfloor + 1$ there exists no child node. Say `arr[1]` represents the first index of the array, and `arr[n]` represents the last element. Take an arbitrary node `arr[i]`, it's left child is indexed at $2i$ and it's right child is indexed at $2i + 1$. So if we can prove that the left child of $\texttt{arr}[\lfloor n/2 \rfloor + 1]$ is out of the array's bounds, then it follows that any node indexed after it also cannot have a child node, thus making it a leaf.

$$\texttt{left}(\texttt{arr}[\lfloor n/2 \rfloor + 1]) = \texttt{arr}[2(\lfloor n/2 \rfloor + 1)] = \texttt{arr}[2\lfloor n/2 \rfloor + 2] \geq \texttt{arr}[n+1] \quad \text{(which does not exist).}$$

6.2

6.2-3

What is the effect of calling MAX-HEAPIFY(A, i) when the element $A[i]$ is larger than its children?

Nothing. Since the element is already in it's correct place, heapify will just immediately return.

6.2-4

What is the effect of calling MAX-HEAPIFY(A, i) for $i >$ *A.heap-size/2*?

Again, nothing. Since the element is already a leaf, there's no children to compare it to, so heapify will just return.

6.2-5

The code for MAX-HEAPIFY is quite efficient in terms of constant factors, except possibly for the recursive call in line 10, which might cause some compilers to produce inefficient code. Write an efficient MAX-HEAPIFY that uses an iterative control construct (a loop) instead of recursion.

```
max-heapify(A,i)
  largest = i
  while true
    left = left(i)    // 2i
    right = right(i)  // 2i+1

    if (left < A.length and A[left] > A[largest])
      largest = left
    if (right < A.length and A[right] > A[largest])
      largest = right

    if (largest != i)
      swap(A[i], A[largest])
      i = largest
    else return
```

6.3

6.3-3

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

From 6.1-7, we know that nodes indexed by $\lfloor n/2 \rfloor + 1$, $\lfloor n/2 \rfloor + 1$, ... , n are leaves. This means that there are $n/2+1$ leaves if n is odd, and $n/2$ leaves if n is even, therefore the number of leaves is equivalent to $\lceil n/2 \rceil$. We can use this property to inductively prove that for any height h of a heap, there are at most $\lceil n/2^{h+1} \rceil$ nodes.

Starting with the base case, we can let n_0 denote the number of nodes at height $h = 0$ on the heap. Since any nodes with a height of 0 are considered leaves (because there's nothing below them), then we can conclude that $n_0 = \lceil n/2 \rceil = \lceil n/2^{0+1} \rceil$, therefore the base case holds.

Now for the inductive step. Assuming, for some $k \in \mathbb{N} : 0 \leq k < h$, that $n_k \leq \lceil n/2^{k+1} \rceil$, we will show that $n_{k+1} \leq \lceil n/2^{k+2} \rceil$. First note that if n_k is even, then the nodes at height $k+1$ have exactly two children per node, so $n_{k+1} = n_k/2 = \lceil n_k/2 \rceil$. And if n_k is odd, then one of the nodes at height $k+1$ has only one child, and the rest have two, so $n_{k+1} = \lfloor n_k/2 \rfloor + 1 = \lceil n_k/2 \rceil$. Knowing this, we can now proceed with our inductive step.

$$\begin{aligned} n_{k+1} &= \left\lceil \frac{n_k}{2} \right\rceil \\ &\leq \left\lceil \frac{\lceil n/2^{k+1} \rceil}{2} \right\rceil \\ &= \left\lceil \frac{n}{2^{k+2}} \right\rceil \end{aligned}$$

Since the claim holds for both the base case and the inductive step, the claim that there are at most $\lceil n/2^{h+1} \rceil$ nodes in a tree of height h holds for all $h \geq 0$.