CPS 130 Homework 9 - Solutions

1. (CLRS 6.1-1) What are the minimum and maximum number of elements in a heap of height h?

   Solution: The minimum number of elements is 2^h and the maximum number of elements is $2^{h+1} - 1$.

2. (CLRS 6.1-4) Where in a max-heap might the smallest element reside, assuming that all elements are distinct?

   Solution: Since the parent is greater or equal to its children, the smallest element must be a leaf node.

3. (CLRS 6.2-4) What is the effect of calling MAX-HEAPIFY(A, i) for $i > size[A]/2$?

   Solution: Nothing, the elements are all leaves.

4. (CLRS 6.5-3) Write pseudocode for the procedures HEAP-MINIMUM, HEAP-EXTRACT-MIN, HEAP-DECREASE-KEY and MIN-HEAP-INSERT that implement a min-priority queue with a min-heap.

   Solution:

```
HEAP-MINIMUM(A)
  return A[1]

HEAP-EXTRACT-MIN(A)
  if heap-size[A] < 1
    then error ``heap underflow''
  min <- A[1]
  A[1] <- A[heap-size[A]]
  heap-size[A] <- heap-size[A] - 1
  MIN-HEAPIFY(A,1)
  return min

HEAP-DECREASE-KEY(A,i,key)
  if key > A[i]
    then error ``new key is larger than current key''
  A[i] <- key
  while  i > 1 and A[parent(i)] > A[i]
    do exchange A[i] <-> A[parent(i)]
      i <- parent(i)

MIN-HEAP-INSERT(A,key)
  heap-size[A] <- heap-size[A] + 1
  A[heap-size[A]] <- +inf
  HEAP-DECREASE-KEY(A,heap-size[A],key)
```

5. (CLRS 6-2) *Analysis of d-ary heaps*
A *d-ary heap* is like a binary heap, but instead of 2 children, nodes have d children.

a. How would you represent a d-ary heap in a array?

b. What is the height of a d-ary heap of n elements in terms of n and d?

c. Give an efficient implementation of EXTRACT-MAX. Analyze its running time in terms of d and n.

d. Give an efficient implementation of INSERT. Analyze its running time in terms of d and n.

e. Give an efficient implementation of HEAP-INCREASE-KEY(A, i, k), which sets $A[i] \leftarrow \max(A[i], k)$ and updates the heap structure appropriately. Analyze its running time in terms of d and n.

Solution:

a. Similarly with the binary heap, a d-ary heap can be represented as an array $A[1..n]$. The children of $A[1]$ are $A[2], A[3], ..., A[d+1]$, the children of $A[2]$ are $A[d+2], A[d+3], ..., A[2d+1]$ and so on. The general rule is:

$$\text{CHILDREN}(i) = \{di - d + 2, di - d + 3, ..., di, di + 1\}.$$

The parent of $A[1]$ is $A[1]$. The parent of $A[i]$ for $2 \leq i \leq d + 1$ is $A[1]$. The parent of $A[i]$ for $d + 2 \leq i \leq 2d + 1$ is $A[2]$. The general rule is:

$$\text{PARENT}(i) = \left\lceil \frac{i-1}{d} \right\rceil.$$

You can check for instance using the rule above that PARENT$(di - d + 2)$ is i and PARENT$(di - d + 1)$ is $i - 1$.

b. The number of nodes at level h is at most d^h. The total number of nodes in a tree of height h is at most $1 + d + \ldots + d^h = \Theta(d^h)$. Setting $d^h = n$ implies the height is $\Theta(\log_d n)$.

c. EXTRACT_MAX is the same as for binary heaps. Its running time is given by the running time of HEAPIFY. The HEAPIFY operat ion on d-ary heaps works very similarly to the one on binary heaps:

> HEAPIFY_D(A, i)
>
> i. find largest element $l = \max\{A[i], \text{CHILDREN}(A[i])\}$
>
> ii. if $l \neq i$ then exchange $A[i] \leftrightarrow A[l]$ and HEAPIFY_D(A, i)

The running time of HEAPIFY_D is $\Theta(d \cdot \log_d n)$. The d term is because at each iteration a node compares its value and the values of its d children to find the maximum, which takes $O(d)$ time.

d. INSERT is the same as for binary heaps. The running time is $\Theta(height) = \Theta(\log_d n)$.

e. The running time is $O(\log_d n)$ if $A[i] < k$.

HEAP_INCREASE_KEY_D(A, i, k)

i. if $A[i] < k$ then

    $A[i] = k$

    while $i > 1$ and $A[\text{PARENT}(i)] < A[i]$ do

        exchange $A[i] \leftrightarrow A[\text{PARENT}(i)]$

        $i = \text{PARENT}(i)$