COSC 302: Analysis of Algorithms — Spring 2018
Prof. Darren Strash
Colgate University
Problem Set 5 — Heaps, Non-comparison sorts, Red-black trees, Hashing

Suggested practice problems, from CLRS: Ch 11.1 (1 and 2); 11.2-3; 12.2 (3, 4, and 5); 12.3-5; 13.3 (1, 2, and 4)

1. In this problem, we will investigate d-ary max-heaps: A d-ary heap is one in which each node has at most d children, whereas, in a binary heap, each node has at most 2 children.

   (a) We can represent a d-heap in an array which the second element is the root. Then for any parent node x it's children are located $(x*d)+1, (x*d)+2, ...(x*d)+d$. An for any child can find it's parent by $(x-1)/d$.

   (b) If the d-ary heap is completely filled then the ith level will be d^i. So the to find the nodes up to the last level of the heap at level l would be:

   $$\sum_{i=0}^{l} d^i = \frac{d^{l+1}-1}{d-1} \tag{1}$$

   This function describes the amount of nodes, or n, so we need to solve for l which would be the height.

   $$\begin{aligned} n &= \frac{d^{l+1}-1}{d-1} \\ n(d-1) &= d^{l+1}-1 \\ log_d(n(d-1)+1)-1 &= l \end{aligned} \tag{2}$$

   This height is $\Theta(log_d(n(d-1)+1)-1)$.

   (c) Re-write function PARENT(i) for d-ary heaps, and give a new function CHILD(i,j) that gives the j-th child of node i (where $1 \leq j \leq d$).

---

function PARENT(i)
    return $(i-1)/d$
function CHILD(x, j)
    return $(i*d)+j$

(d) Describe, and give pseudocode for, the algorithm MAX-HEAPIFY(A,i) for d-ary heaps and give a tight analysis for the worst-case running time of your algorithm.

```
function MAX-HEAPIFY(A,i)
    largest ← i
    for x ← 1 to d do
        if Child(A, x) > i then
            largest ← Child(A,x)
    if largest ≠ i then
        exchange A[i] with A[largest]
        Max-heapify(A,largest)
```

The worst-case running time would be $\Theta(log_d(n(d-1)+1)-1)$ if the value floats to the bottom of the tree. If the tree is balanced.

(e) Describe (semi-formally) how to implement MAX-HEAPIFY(A,i) in $O((\log_d n)\lg d)$ time. *(Hint: you need auxiliary data structures; the heap itself is not sufficient.)*

One can make the function take $O((\log_d n)\lg d)$ if you store the children of each node using a binary search tree. Finding the largest between i and the children would take $O(lg(d))$ since the tree would have d nodes and in the worst case bubble down to the leaves which would be $log_d(n)$ recursive calls. During each call you would traverse the children in $O(lg(n))$ which is $log_d(n)lg(n)$.

2. **(From homework 4, skip if already submitted)** Problem 8.2-4 from CLRS: Describe (semi-formally) an algorithm that, given n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range $[a..b]$ in $O(1)$ time. Your algorithm should use $\Theta(n+k)$ preprocessing time.

```
function PRE-PROCESS(A,k)
    C[0...k]
    for i = 0 to k do
        C[i] = 0
    for j = 1 to A.length do
        C[A[j]] = C[A[j]] + 1
    for i = 1 to k do
        C[i] = C[i] + C[i − 1]
    A = C
function RANGE(A,k,a,b)
    Pre-Process(A,k)
    return A[b] − A[a]
```

3. Problem 13.3-5 from CLRS. (Describe semi-formally.) *(Hint: Follow the structure for an invariant.)*