Algorithms and Data Structures

Luke Nigro - Homework 3

1a. A *d-ary* heap is similar in looks to a normal binary heap, with the only caveat being that each node has d children as opposed to 2. The array of the *d-ary* would have the following properties:

   (a) **Root:** $A[1]$

   (b) **Parent of** $A[i] = A[\lfloor \frac{i}{2} \rfloor]$

   (c) **Children of Parent** $A[i] = A[2i, ..., 2i + d - 1]$

1b. Since a binary heap is in the form of a binary tree, a binary heap has between 2^h and $2^{h+1} - 1$ elements, or $2^h \leq n \leq 2^{h+1} - 1$ where h is the height of the heap. Extending this to a *d-ary*, we should have:

$d^h \leq n \leq d^{h+1} - 1 \Rightarrow h \leq \log_d n \leq h + 1 \Rightarrow h = \lfloor \log_d n \rfloor$

1c. The runtime of EXTRACT-MAX for a normal binary heap is $O(lgn)$, which is equivalent to the runtime of MAX-HEAPIFY. This makes sense, since the MAX-HEAPIFY function is the only part of the EXTRACT-MAX function that takes a noticable amount of runtime. Extending to the *d-ary*, the runtime of MAX-HEAPIFY should increase to $O(d * \log_d n)$ since each of the *sub-d-arys* will have to loop through the children d times to find the maximum value of each *sub-d-ary*. Like the binary-heap, this process is repeateda number of times equivalent to the height, which in this case is $\log_d n$. So - the runtime of our EXTRACT-MAX will continue to be equivalent to the runtime of MAX-HEAPIFY, which again in this case is $O(d * \log_d n)$.

1d. The INSERT function is solely dependant on the runtime of the HEAP-INCREASE-KEY function, which is dependant on the height of the heap. So in the example of a binary heap, both functions will have a runtime of $O(lgn)$. The height of our *d-ary* is $\log_d n$, so the runtime of the INSERT function here is $O(\log_d n)$.

2. Minimum of the Young tableau is always in $A[1,1]$
   $min = A[1,1]$
   $i, j = 1$
   while $i < m$ and $j < n$
      if $A[i+1, j] \leq A[i, j+1]$
         exchange $A[i, j]$ and $A[i+1, j]$
         $i = i + 1$
      else
         exchange $A[i, j]$ and $A[i, j+1]$
         j = j + 1
   $A[i, j] = \infty$
   return A, min

   The while loop here will run no more than $m + n$ times, therefore our algorithm has a runtime of $O(m + n)$ as desired.

Interview: Since we know the integers involved here, this problem should be fairly simple to solve with a binary max-heap and a heapsort. Using MAX-HEAPIFY we can easily implement a HEAPSORT algorithm which will place objects of the same color adjacent to eachother in $O(lgn)$ time. To save some runtime, we could include logic that would tell the program to stop itself once the maximum value of the heap is 0, this way the heap will have a runtime of $O(lg(n) - k)$ where$k$ is the number of red objects.