1. A $d$-ary heap is like a binary heap, but (with one possible exception) non-leaf nodes have $d$ children instead of 2 children.

   (a) How would you represent a $d$-ary heap in an array?

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

   (c) Assume the heap operations are only BUILD-HEAP, EXTRACT-MAX, HEAP-INSERT. Now Dr. D claimed that 3-ary heaps are better (faster) than binary heaps because they need less comparisons. Check whether the claim is right or wrong and explain your answer.

2. Bottom-up(A)
   1 for $i = \lfloor n/2 \rfloor$ down to 1
   2 Heapify(A, i)

   (a) If we rewrite the line1 of Bottom-up as ”for $i = 1$ to $\lfloor n/2 \rfloor$”, is the output still correct? Please explain your answer.

   (b) We define the **height of a node** in a heap to be the number of edges on the longest simple downward path from the node to a leaf. Show that there are **at most** $\lceil n/2^{h+1} \rceil$ nodes of height $h$ in any $n$-element heap. (Viewing a heap as a binary tree)

   (c) Prove that the running time of Bottom-up is $O(n)$ by (b). (The running time of Heapify is $O(\lg n)$)
3. We can build a max-heap by repeatedly inserting elements into the heap. Consider the following implementation:

```
3. We can build a max-heap by repeatedly inserting elements into the heap. Consider the following implementation:

BUILD-MAX-HEAP'(A)
1  heapsize[A] = 1
2  for i = 2 to length[A]
3     do MAX-HEAP-INSERT(A, A[i])

MAX-HEAP-INSERT(A, key)
1  heapsize[A] = heapsize[A] + 1
2  i = heapsize[A]
3  A[i] = key
4  while (i > 1 and A[⌊i/2⌋] < A[i])
5     do exchange A[i] <-> A[⌊i/2⌋]
6     i = ⌊i/2⌋
```

(a) Do the procedures BUILD-MAX-HEAP' and Bottom-up always create the same heap when run on the same input array? Prove that they do, or provide a counterexample.

(b) Show that in the worst case, BUILD-MAX-HEAP' requires Θ(n lg n) time to build an n-element heap.

4. Show that, with the array representation for storing an n-element heap, the leaves are the nodes indexed by ⌊n/2⌋ + 1, ⌊n/2⌋ + 2, . . . , n.

5. Ref. to section 6.2 of textbook, inorder to show MAX-HEAPIFY runs in O(/lg n), show that the children’s subtrees each have size at most 2n/3.