Questions

1. Suppose you have an array with the following characters in it.

   `f b u e l a k d w`

   Show how the array evolves after each of the $n = 9$ loops of the `buildHeap` method.

2. (a) Similar to the previous question, build a heap out of the characters of the word: `computed`.
   (b) What is the result of applying the `removeMin()` method to the heap in (a) ?
   (c) Give an example of a heap that uses the characters of the word `computed`, and that has the following additional property: for any node with two children, the left child is less than the right child.
   (d) Give an example of a complete binary tree that contains the characters of the word `computed` and is also a *binary search tree*. (Such a tree is not a heap.)

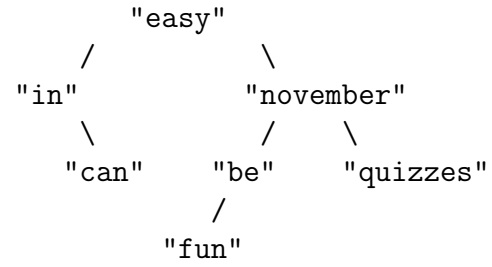
3. A *d-heap* is defined as follows. It is a tree that has up to d children per node, and each level of the tree is full except possibly for the last level. In the last level, nodes are filled from the left. The heaps we saw in the lectures were 2-heaps.

   (a) Give a precise parent-child indexing scheme for representing the elements of a *4-heap* using an array. Unlike in the 2-heap case we saw in class where the root was at array element 1, here I am requiring that the root is at array element 0.
   Note: Questions (b) and (c) do *not* require that you answer (a) correctly.
   (b) Consider a 4-heap with ten elements, $a[i] = i$, for $i = 0, 1, \ldots, 9$.
   What is the result of applying `removeMin()` to this 4-heap ? Note that you need to generalize the `removeMin()` method from class which applied only to 2-heaps.
   You may write your answer either as a tree or as an array.
   (c) Give an upper bound on the maximum number of nodes in a d-heap of height h.
   Do *not* leave your answer as a summation.

4. The `removeMin()` method puts the last element of the heap into the root. Will this element necessarily need to be moved down in the heap? The answer is obviously no if there were only two elements in the heap before the call. Consider the more interesting case that the heap had at least four elements in it when the `removeMin()` was called.

   If this is obvious to you, then you should still try to write down your argument why.

5. Consider the following binary tree:



   (a) Transform this binary tree into a *binary search tree* of height 2, defined by the natural ordering on strings.

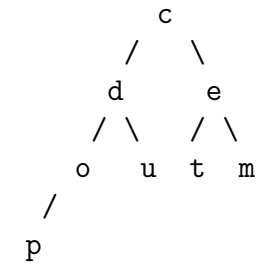
   (b) Represent this binary tree (shown above i.e. *not* the answer to (a)) using an array of references to strings, such that the parent/child indices in the array are the same as that used in a *heap*.

Answers

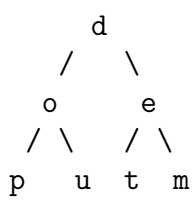
1. The boundary between the heap and non-heap elements is marked with |.

```
    1   2   3   4   5   6   7   8   9
   -----------------------------------
|   f   b   u   e   l   a   k   d   w
    f | b   u   e   l   a   k   d   w    (added f)
    b   f | u   e   l   a   k   d   w    (added b)
    b   f   u | e   l   a   k   d   w    (added u)
    b   e   u   f | l   a   k   d   w    (added e)
    b   e   u   f   l | a   k   d   w    (added l)
    a   e   b   f   l   u | k   d   w    (added a)
    a   e   b   f   l   u   k | d   w    (added k)
    a   d   b   e   l   u   k   f | w    (added d)
    a   d   b   e   l   u   k   f   w |  (added w)
```

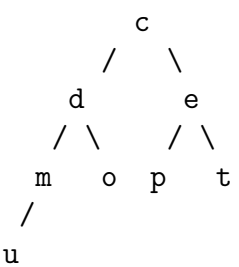
2. (a)



(b)



(c) The easiest way to solve this one is to sort the elements from smallest to largest. The resulting sequence, if written as a array, defines a heap.



(d) The tree node structure is determined by the number of nodes. The only question is where to place the elements. But remember that when you traverse a binary search tree "in order", you visit the elements in order. So you need to sort the elements of the string: `cdemoptu` and place them into the tree in order.

```
          o
        /   \
      e       t
     / \     / \
    d   m   p   u
   /
  c
```

3. (a)
```
child  =  4 *parent + i,  where i = 1,2,3,4
parent =  (child - 1) / 4
```
   (b) `1 5 2 3 4 9 6 7 8`

   (c)

$$1 + 4 + 4^2 + 4^3 + \cdots + 4^h = \frac{4^{h+1} - 1}{4 - 1}$$

4. Yes, it will necessarily need to be moved down. If the last element in the heap is at level l where $l > 1$, then it will have an ancestor that is a child of the root. (This ancestor may be its parent, or grandparent, etc.) But in a heap, a node is always greater than all of its ancestors. So, when the last element is moved to the root and becomes a parent of one of its (former) ancestors, this new root will be greater than its (former) ancestor, which is now its child, and so this new root will not satisfy the heap property.

5. (a)
```
          fun
        /     \
    can         november
   /  \       /      \
  be  easy   in      quizzes
```
   (b)

1	2	3	4	5	6	7	8	9	10	11	12
easy	in	november	*	can	be	quizzes	*	*	*	*	fun

where * is a null reference.

