1. Counting Sort

Use counting sort to sort the following array of characters:

```
[a, b, a, a, c, d, a, a, f, c, b]
```

How much space is required if the array has n characters and our alphabet has k possible letters?

2. Radix Sort

Use radix sort to sort the following strings:

```
abc bab cba ccc bbb aac abb bac bcc cab aba
```

As a reminder radix sort works on strings of length k by doing k passes of some other (stable) sorting algorithm, each pass sorting by the next most significant element in the string. For example in this case you would first sort by the 3rd character, then the 2nd character and then the 1st character.

3. Stable Counting Sort

Which property is required to use counting sort to sort an array of tuples by only the first element, leaving the original order for tuples with the same first element. For example the input may be:

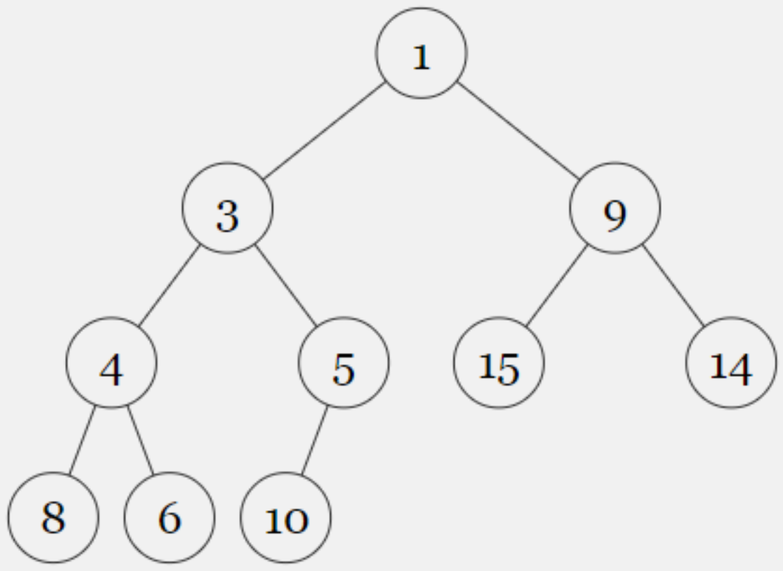
$$(8, \texttt{campbell}),\ (6, \texttt{tal}),\ (3, \texttt{keir}),\ \ldots (6, \texttt{gus}),\ (0, \texttt{nick}),\ (8, \texttt{tom})$$

Discuss how you would ensure that counting sort satisfies this property. Can you achieve this using only arrays? How about using auxiliary linked data structures?

A *Heap* (for this question we will discuss a min-heap) is a complete binary tree which satisfies the heap property:

Heap Property: each node in the tree is no larger than its children.

Suppose we start out with the following heap:



a. Show how this heap would be stored in an array as discussed in lectures, *i.e.*, the root is at index 1 and a node at index i has children in indices $2i$ and $2i + 1$.

b. Run the RemoveRootFromHeap algorithm from lectures on this heap by hand (*i.e.*, swap the root and the "last" element and remove it. To maintain the heap property we then SiftDown from the root).

c. Run the InsertIntoHeap algorithm and insert the value 2 into the heap (*i.e.*, add the new value to the end of the heap, and compare it with its parent, swapping the new value and its parent until its parent is smaller than it, or it is the root).

5. (Homework) Using a min-heap for *k*th-smallest element

The *kth-smallest element problem* takes as input an array A of length n and an index $k \in \{0, \ldots, n-1\}$ and returns as output the kth-smallest element in A (with the 0th-smallest being the smallest, the 1th-smallest being the second smallest and so on).

How can we use a the min-heap data structure to solve the kth-smallest element problem? What is the time-complexity of this algorithm?

When would we use the QuickSelect algorithm from last week's Question 3 instead of the heap based algorithm?

6. (Revision) Recurrence Relations

Solve the following recurrence relations. Give both a closed form expression in terms of n and a Big-Theta bound.

a. $T(1) = 1$, $T(n) = T(n/2) + 1$.

b. $T(0) = 0$, $T(n) = T(n-1) + n/5$.