COMP 212
Written Assignment 3
Solutions

1. (Pretty tricky!) For the case $n = 15$, we note that level 0 may contain $k = 1$, level 1 may contain $k = 2, \ldots, 9$, level 2 may contain $k = 3, \ldots, 13$ and level 3 may contain $k = 4, \ldots, 15$. To generalize this we formalize two principles: First, small values must be near the top of the heap, e.g., the second smallest must be on level 1, the third smallest must be on level 1 or 2, etc. Second, large values must be near the bottom of the heap, e.g., the largest must be a leaf, the second largest must be a leaf unless $n$ is even in which case it can be the father of the last leaf which must be the largest, etc. Assume the heap has $n$ nodes and is such that the first $h - 1$ levels are completely full and the last level has between 0 and $2^h - 1$ leaves from left to right. The first condition is easy to test for: the $k$th smallest may appear on level $l$ only if there are at least $l - 1$ elements smaller than it. This occurs only if $k > l$. The second condition is harder to test. It is equivalent to insuring that there are enough elements larger than the the $k$th element to fill a subtree below it. The smallest subtree at the $l$th level is the one rooted at the rightmost position in the level. It has (if I did my math correctly) $X = 2^{h-l} - 2 + \max \{0, n - 2^{h+1} + 2^{h-l} + 1\}$ nodes below it. This implies that for $k$ to appear on the $l$th level it must have $X$ values larger than it. This occurs only if $n - k \geq X$. The two conditions together determine if a value can appear on the $l$th level (both must hold).

2. In general the heap property can not be used to print out the keys of an $n$ node tree in sorted order in $O(n)$ time. As we saw in the previous question the elements can appear almost anywhere in the tree so once the $i$th element is reported, finding the $(i+1)$st is not an easy problem. (Of course, if the tree also has the shape property, one can report the elements in sorted order in $O(n \log n)$ time but this destroys the heap in the mean time.) If we require both the heap and the shape property to hold and insist the elements are distinct then only the empty or one node binary trees have both properties. For distinct elements, only the degenerate tree where each node has only right children, has both properties. If we do not require distinct elements then any binary tree can have both properties by assigning the same key to all nodes.
3. The idea is to use a vector or array with precisely 1001 entries. This acts as a hashtable with the “perfect” hash function that maps the association with key $i$ to position $i$ in the table. Note that all operations are $O(1)$ since at most one element is sent to anyone array position. This will still work if the key space is large but it is extremely wasteful of memory space, if only 500 associations are to be stored. (Note: If what is being stored is actually a subset of the integers from 0 to 1000 then one can use an vector of bits where 0 means the number is not in the set and 1 means the number is in the set. This is called a bit-vector and is extremely space efficient in this special case, which occurs not infrequently.)

4. (a) Using binary search on array with $n+1$ elements for a value that is not in the array uses at least $\lceil \log(n + 1) \rceil$ comparisons.

(b) If interpolation search on the input described is performed the first step will be to compare $2k - 1$ to $a(m)$ where $m = 0 + (2k - 1 - 0) \times (n - 0)/(2n - 0) = k$ (assuming the ceiling of the $m$ is taken - a similar analysis holds for the floor). Since $a(m) = 2k$, in the next call $l = 0$ and $r = k - 1$. In this case $2k - 1$ will be compared to $a(m)$ where $m = 0 + (2k - 1 - 0) \times (k - 0)/(2k - 0) = k$. At this point the algorithm can conclude that the item is not in the array. The total number of items examined was three including $a(0)$, $a(n)$ and $a(k)$!

5. There are 14 different trees. For 8 of the trees, there is a single permutation that generates them (these are all paths of height 3). For 2 of the trees, there are two permutations that generate them (in these, the root has one child which has two children). For 4 of the trees, there are 3 permutations that generate them (in these, the root has two children one of which has a child).