Please send any comments or criticisms to dwharder@alumni.uwaterloo.ca
with the subject ECE 250 Questions 8.4.
Assistances and comments will be acknowledged.

8.4*a* Demonstrate how the sum

$$\sum_{k=0}^{h} 2^k (h-k)$$

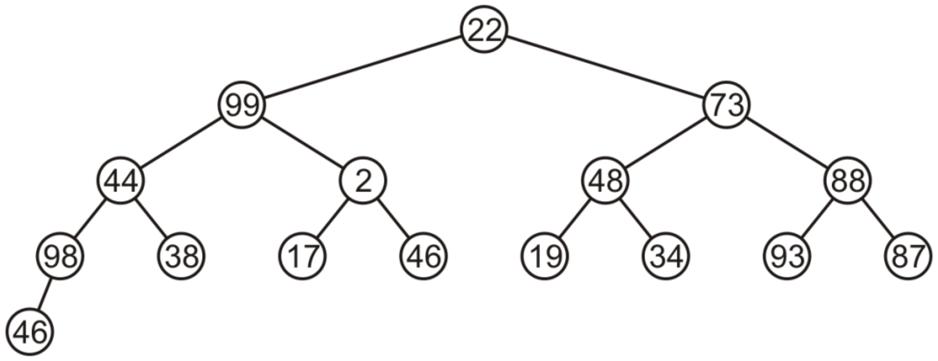
is formulated in calculating the expected number of inversions.

8.4*b* The above sum evaluates to $2^{h+1} - h - 2$. How is this used to demonstrate that heapification can be performed in $\Theta(n)$ time.

8.4*c* Suppose we started with the root instead of at the leaves to heapify a tree. At each step, we would compare the value of the node with its parent (if any) and swap it and the parent if the value of the node is larger than that in the parent and then continue percolating it up, as necessary. What are the best-case and worst-case run times of such an algorithm? Give examples that achieve your claimed run times.

8.4*d* Suppose we use a strategy similar to insertion sort: treat the root node as a heap, and then successively increase the size of the heap by inserting the next node. Recall the argument that the insertion of a random element into a heap would run in $\Theta(1)$ time on average but $O(\ln(n))$ time in general. What would the average run time of this algorithm be, and what would the worst-case run time be? Again, give an example that would achieve the worst-case run time.

8.4*e* Convert the following complete tree into a max heap:



8.4*f* Convert the following array representation of a complete tree into a max heap:

52	69	38	79	66	64	72	3	16	89	15	37	0	28	73	95

8.4*g* Suppose we want to implement heap sort as follows:

```
template <typename Type>
void heapsort( Type *array, int n ) {
    max_heapify( array, n );

    for ( int i = n - 1; i > 1; --i ) {
        array[i] = max_heap_pop( array, i + 1 );
    }
}
```

Implement the two functions `max_heapify` and `max_heap_pop`.

```
// Convert the array representation of a complete tree into a heap of size n
template <typename Type>
void max_heapify( Type *array, int n ) {









}
```

```
// Pop and return the top of a heap defined in an array of size n
template <typename Type>
Type max_heap_pop( Type *array, int n ) {









}
```