Assignment 10

Alex Clemmer
Student number: u0458675

1

This solution is pretty simple in practice. We use a variant of quickselect, which is sort of like quick sort.

We begin by looking at just the first bit of all the numbers. If it's a 1, it goes at the end; if it's a 0, it goes at the beginning. Because one number is missing, the two "halves" we have created should actually be unequal. This is because one of them is missing an element.

To find out which half is unequal, we look at the leading bit of the number at position $2^k - 1$. We expect that this should be the last element with a 0 as this bit. If it is indeed a 0, then the second half of the array (with 1's in the leading bits) are missing a value. If it is not, then the first half of the array (with 0's in the leading bits) are missing a value.

We recurse down, repeating this operation with the next-most-significant bit each time until we have recursed to the $k - 1^{th}$ bit. The smaller "half" will then have a 1 or a 0. This will have given us k bits for our unknown number, which is coincidentally what we set out to do.

Now what is the worst case? Assuming we're using the Master Theorem, $a = 1$. Each level would require $n/2$ swaps (as this is the worst case), so $T(n) = T(n/2) + f(x)$. The $f(x)$ at the end describes for the time required to find the i-th bit of every element, and to compare them, which totals out to be $O(n)$. Clearly, $a = 1$, since we compare and swap only half the elements (remember that this is worst case). Since we then have $T(n) = T(n/2) + O(n)$, by the definitions provided by the Master Theorem, $1 > \log_2 1$, and therefore our running time is:

$$O(n^1) \tag{1}$$