Assume we have an array $A[1..n]$ of $n$ distinct integers

1. Write an algorithm to check whether an integer $x$, exists in $A$. It returns the index of the element with value $x$, or 0 if not found. Do not make any assumptions regarding whether $A$ is sorted.

Here is a possible solution:

Input: Array $A[1..n]$
Output: found = index of entry that has value $x$, or found = 0 if not found

```
found = 0;
i = 1;
while(i <= n)
    if( $A[i] == x$ )
        found = i;
        return(found);
    else
        i = i + 1;
return(found);
```

2. What is the complexity of the algorithm in 1?

The algorithm has to check each of the $n$ elements (in the worst case) so it has a complexity of $O(n)$.

3. What is the best case and worst case runtime?

The best cases to find the required element with the first check so $O(1)$. The worst case is to not find the element at all so $O(n)$.

4. Assume the array is sorted (in ascending order), write an algorithm to check whether an integer $x$ exists in an array $A$. 

```
The function binary_search will be called initially with binary_search(A[], 1, n).

5. Will it make any difference if the array is sorted increasingly or decreasingly? Justify.

No, we just need to reverse the conditions above.

6. Make a search on the web and find the best sorting algorithm. What is the name of that algorithm? What is its complexity?

Heapsort is one of the best with complexity $O(n \lg n)$.

7. If we use the algorithm you found in 6 to sort A then use the algorithm you wrote in 4, will the resulting algorithm be better or worth than the algorithm you wrote in 1? Justify.

It will be worse because sorting takes $O(n \lg n)$ then the search takes $O(\lg n)$ making the whole thing $O(n \lg n)$. The algorithm at 1 has $O(n)$. 

```python
binary_search(A[], start, end):
    found = 0
    middle = (start + end)/2
    if(A[middle] == x):
        found = middle
        return(found)
    if( A[middle] > x):
        found = binary_search(A[], start, middle-1)
    else
        found = binary_search(A[], middle+1, end)
    return(found)
```