8a) Bubble sort has best-case complexity $\Theta(n)$.

This happens if the input array is already sorted.

`[0, 1, 2, 3, 4, 5, 6, 7, 7, 7]`

The worst case complexity is $\Theta(n^2)$.

This happens if the input array is reverse-sorted.

`[7, 7, 7, 6, 5, 4, 3, 2, 1, 0]`

b) Heapsort has best-case complexity $\Theta(n \lg n)$

This happens when the array is already arranged into a binary heap.

`[7, 7, 7, 6, 5, 4, 3, 2, 1, 0]`

The worst case complexity is also $\Theta(n \lg n)$, but with different lower-order terms.

This happens when the input array forms a min-heap.

`[0, 1, 2, 3, 4, 5, 6, 7, 7, 7]`

c) Quicksort has best-case complexity $\Theta(n \lg n)$

This happens when the median element of each sub-array starts at the end before partitioning.

`[0, 3, 2, 1, 7, 5, 6, 7, 7, 4]`

The worst case complexity is $\Theta(n^2)$

This happens when the array is already sorted.

`[0, 1, 2, 3, 4, 5, 6, 7, 7, 7]`

d)

```python
def removeDuplicates(list):
    currentNode = list.head
    # outer loop
    while currentNode.next != None:
        scanNode = currentNode
        # inner loop
        while scanNode.next != None:
            if currentNode.value == scanNode.next.value:
                scanNode.next = scanNode.next.next
            else:
                scanNode = scanNode.next

        currentNode = currentNode.next
    return
```

In the worst case (when there are no duplicates), for a list of size n, the outer loop of the above function runs exactly $n - 1$ times, since the `currentNode` progresses one node down the list each time. With each run of the inner loop, the `scanNode` progresses, so the inner loop runs $n-1, n-2, \ldots, 1, 0$ times, giving a total of $(n-1)^2$ passes of the inner loop. The contents of the inner loop run in constant time, so the complexity is $\Theta(n^2)$ in the worst case.

In the best case (when all elements are the same), for a list of size n, the outer loop runs only once - since after one pass the tail will be empty - and the inner loop runs $n - 1$ times, so the complexity is $\Theta(n)$.

e)

$$f(n) \in \Omega(g(n)) \iff \exists\, n_0, c \in \mathbb{R}_{>0} \text{ such that } \forall\, n > n_0 : 0 < c \cdot g(n) \leq f(n)$$

In order to detect duplicates, elements must be compared to other elements. The worst case will therefore always be the case where there are no elements duplicated, so that all elements must be mutually compared, rather than being removed.

The mutual comparison of n elements requires $(n-1)(n-2)$ comparisons, so the theoretical minimum complexity is $\Omega(n^2)$.