Question 1) [2 points]
Is $2^{n+1} = O(2^n)$?
Since we have $2^{n+1} = 2^n \times 2$ and $O(2^n)$ exists constant $c \geq 2$vsuch that $0 \leq 2^n \times 2 \leq c \times 2^n$.
Therefore, $2^{n+1} = O(2^n)$ is true.

Question 2) [2 points]
Is $2^{2n} = O(2^n)$?
Observe that $2^{2n} = O(2^n)$, then $2^n \times 2^n = c \times 2^n$. Therefore c has to be larger or equal to 2^n to have $2^n \times 2^n \leq c \times 2^n$. However, 2^n happens to be unbounded which makes no c exists to satisfy the upper bound of $2^n \times 2^n$. In conclusion, $2^{2n} = O(2^n)$ is false.

Question 3) [2 points]
Explain why the statement, "The running time of algorithm A is at least $O(n^2)$ is meaningless.

Suppose the running time of algorithm A is T(n), and according to the statement, we have $T(n) \leq O(n^2)$. Additionally, $O(n^2)$ describes the upper bound of the growth of the running time of A T(n). However, beyond that, it is ambiguous to determine the efficiency or speed of running time T(n). Therefore, the statement is meaningless because it does not give many specifical details about how efficiently the algorithm runs.

Question 4) [2 points]
Using reasoning similar to what we used for insertion sort in class, analyze the running time of the selection sort algorithm from the first homework assignment.

arr.length = n

selectionSort(arr)	cost	time
`for startOfUnsorted in 0 to arr.length`	c_1	n
`  smallestInUnsorted = max value`	c_2	n - 1
`  indexOfSmallest = -1`	c_3	n - 1
`  for i in startOfUnsorted to arr.length`	c_4	$\sum_{i=0}^{n} t_i$
`    int current = arr[i]`	c_5	$\sum_{i=0}^{n} (t_i-1)$
`    if arr[i] < smallestInUnsorted`	c_6	$\sum_{i=0}^{n} (t_i-1)$
`      smallestInUnsorted = arr[i]`	c_7	$\sum_{i=0}^{n} (t_i-1)$

`indexOfSmallest = i`	c_8	$\sum_{i=0}^{n} (t_i-1)$
`swap arr[indexOfSmallest],arr[startOfUnsorted]`	c_9	n - 1

Question 5) [2 points]

Why do all cases have the same runtime in selection sort?

Despite the number of elements and their orders, as we have to loop through the array and determine every element, the program stops when no more element to be compared or the program has finished evaluating every element. Furthermore, as we implement two loops for the program, when it finishes executing, it has a runtime of $O(N^2)$. In conclusion, every case of selection sort has the same runtime of $O(N^2)$ because the program has to finish 2 nested loops in order to sort every element.