4.5-5
Let $T(n) = T(n/2) + n(sin(n\pi/2) + 1)$, it's obvious that
$f(n) = n(sin(n\pi/2) + 1) = \Omega(n^{\epsilon})$,
but we can't find $c < 1$ such that
$n/2(sin(n\pi/4) + 1) \leq cn(sin(n\pi/2) + 1)$
holds for all sufficient large n. The reason is that for any $n = 4k + 3$, the right hand equals 0, while the left hand is strictly greater than 0.
4.1
For a-f, we can easily apply master theorem to get the following answers:
$a.\Theta(n^4)$, case 3
$b.\Theta(n)$, case 3
$c.\Theta(n^2 logn)$, case 2
$d.\Theta(n^2)$, case 2
$e.\Theta(n^{log_2 7})$, case 1
$f.\Theta(n^{1/2} logn)$, case 2
For g, we have
$T(n) = \sum_{i=0}^{\lfloor n/2 \rfloor}(n - 2i)^2$
$= \sum_{i=0}^{\lfloor n/2 \rfloor}(n^2 - 4in + 4i^2)$
$= n^2 \sum_{i=0}^{\lfloor n/2 \rfloor}(1) - 4n \sum_{i=0}^{\lfloor n/2 \rfloor}(i) + 4 \sum_{i=0}^{\lfloor n/2 \rfloor}(i^2) = \Theta(n^3)$
6.1-4
Leaves, since the smallest element is not supposed to have children otherwise its children is smaller than it.
6.3-2
In MAX-HEAPIFY, the binary tree rooted at LEFT(i) and RIGHT(i) are required to be max-heaps. If you do increase, the requirements are not guaranteed.
6.5-7
Simply associate an index with each incoming element and use the index as the key in min-heap. To avoid index overflow, when the index reaches a certain pre-defined threshold, decrease the index for each element in the priority queue. For stack, use max-heap instead
6.5-9
Form a k-element heap using the biggest number of each sorted list, extract the top of the heap each time, e.g., the biggest one from list i, the put the second largest one from list i to the heap, and heapify it. Do this until all elements have been extracted, the complexity is clearly $O(nlgk)$
6-1
a. No, consider a three element array [1,2,3], the two procedures create different heaps
b. It's easy to see the upper bound is $O(nlgn)$, for the lower bound, consider an array with elements in increasing order, each insertion will cost $\Theta(lgn)$, so the lower bound is $\Omega(nlgn)$, therefore we have the complexity of $\Theta(nlgn)$
9.2-4
The worst case happens when the biggest element is selected as pivot each time.
9.3-1

If elements are divided into group of 7, similar with the analysis in 9.3, the recurrence is $T(n) = T(\lceil n/7 \rceil) + T(5n/7 + 8) + O(n)$, which is bounded by $O(n)$
If elements are divided into group of 3, similar with the analysis in 9.3, the recurrence is $T(n) = T(\lceil n/3 \rceil) + T(2n/3 + 4) + O(n) \geq T(n) + O(n)$, which means the divide-and-conquer method actually makes things worse. Therefore, it cannot be linear.

9.3-8

Please refer to: http://www.geeksforgeeks.org/archives/2105