Question 1. $(\log \log n)^2$, $(\log n)^{0.3}$, $\sqrt{n}$, $\{n \log n, \log(n!)\}$, $n^{1.1}$, n^2, 2^n, $n!$

How to figure out $\log(n!)$: In such cases it often helps to "sandwich" the troublesome function (in this case $n!$) between two other functions, like this:

$$(n/2)^{(n/2)} < n! < n^n$$

Taking logarithms does not change the inequalities (because log is an increasing function) and gives:

$$(n/2)\log(n/2) < \log(n!) < n \log n$$

which shows that $\log(n!)$ has same order of growth as $n \log n$. (Another way of reaching the same conclusion is by using the Stirling approximation for $n!$)

Question 2.

1. At level i of the recursion tree, the problem size associated with a node is $n/2^i$, therefore the deepest level (i.e., the height h) corresponds to $n/2^h = 1$, which gives $h = \log_2 n$.

2. If the local work at a node is $c_2 n^2$ then for its 3 children it is

$$3c_2(n/2)^2 = 3c_2 n^2/4 = (3/4)c_2 n^2$$

which is 3/4 of the parent's work. So the work done for level i is $(3/4)^i c_2 n^2$.

3. The total work done over all levels is no more than:

$$c_2 n^2(1 + (3/4) + (3/4)^2 + (3/4)^3 + \cdots) = c_2 n^2(1/(1 - (3/4))) = 4c_2 n^2$$

and therefore $T(n)$ is $O(n^2)$.

Question 3. Obtain from A a set B where

$$B[k] = x - A[k]$$

and note that, because the elements of A are distinct, so are the elements of B (i.e., just as in A, no two entries of B are equal). Create in $O(n \log n)$ an array C of $2n$ elements that is a sorted version of $A \cup B$. Go through C and check whether any two adjacent elements in the sorted C are equal: If yes then one of them comes from A (say it is $A[i]$) and the other comes from B (say it is $B[j] = x - A[j]$), and the fact that they are equal implies that $A[i] = x - A[j]$ and hence $A[i] + A[j] = x$. If no two adjacent elements of the sorted C are equal then x is not the sum of two elements of A.

Note. What if the problem formulation was changed so that the elements of A are not necessarily distinct (i.e., A might contain repeated elements)? Then we would simply pre-process A to remove from it all duplicate elements (by sorting it, etc) and then use the above algorithm on the duplicate-free version of A.

Question 4. The first recursive call is now on a set $\hat{S}$ of size $n/3$. The second recursive call is on a set that does *not* contain half of $\hat{S}$, and each of the elements of $\hat{S}$ so excluded causes another element from its group of 3 to also be excluded, i.e., the number of elements *excluded* from the second recursive call is at least

$$(|\hat{S}|/2)(1+1) = 2(n/6) = n/3$$

and therefore the number of elements included in the second recursive call is no greater than

$$n - (n/3) = 2n/3$$

This means the recurrence is $T(n) = c_1$ for small n (say, for $n \leq 20$), otherwise

$$T(n) = T(n/3) + T(2n/3) + c_2 n$$

whose solution is $O(n \log n)$ because (as explained in class) the work stays same from one level of the recursion tree to the next level and there is a logarithmic number of levels (specifically, $\log_{3/2} n$ levels).

Question 5. The returned quantities are computed as follows.

1. Compute $M = \max\{M', M'', R' + L''\}$. If the maximum turns out to be M' then set $i = i'$ and $j = j'$, if it turns out to be M'' then set $i = i''$ and $j = j''$, and if it turns out to be $R' + L''$ then set $i = r'$ and $j = l''$.
2. Compute $L = \max\{L', W' + L''\}$. If the maximum turns out to be L' then set $l = l'$, if it turns out to be $W' + L''$ then set $l = l''$.
3. Compute $R = \max\{R'', W'' + R'\}$. If the maximum turns out to be R'' then set $r = r''$, if it turns out to be $W'' + R'$ then set $r = r'$.
4. Compute $W = W' + W''$.