Q1. (20 points) Compute tight big-Oh bounds for the following recurrences:

a. $T(n)=8T(n/4)+O(n)$
Use Master's theorem. a=8, b=4,d=1. Since $\log_4 8=1.5 > 1$, the complexity is $O(n^{1.5})$

b. $T(n)=2T(n/4)+O(n^{1/2})$
Using master's theorem, we have a=2, b=4, d=1/2. Now, $\log_4 2=0.5 = 0.5$,
therefore the complexity is $O(n^d \log_b n) = O(n^{0.5} \log_4 n)$.

c. $T(n)=T(n-4)+O(n^2)$
We can see that in each call n decreases by 4, so it takes at most n/4 steps to reduce to size n <= 3. Each call takes $O(n^2)$ time, so total time is n/2 x $n^2 = O(n^3)$.

d. $T(n)=T(n^{1/2})+O(n)$
Expanding the recursion, we have
$T(n)=T(n^{1/2})+O(n) = T(n^{1/4})+O(n^{1/2}) + O(n) = T(n^{1/8})+O(n^{1/4}) + O(n^{1/2}) + O(n) = \ldots$

In general we keep on taking square root of the previous input, so that in step k, we are trying to compute a term of the form $n^{\frac{1}{2^k}}$. Let us assume for simplicity that n is a power of 2. So the number of steps to reduce the problem size to 2 is given as $n^{\frac{1}{2^k}} = 2$, which implies $\frac{1}{2^k}\log n = 1$, or $\log n = 2^k$, i.e.,
k = log log n.
Therefore the total cost is:

$$\sum_{i=0}^{\log\log n} n^{\frac{1}{2^i}} \quad \leq n \log\log n$$

Therefore the time is $O(n \log\log n)$.

Actually, we can get a tighter bound of O(n) by noting that the cost is n + $(n^{0.5})$ log log n, if we pull out the n and bound the remaining log log n terms by $n^{0.5}$. However $n^{0.5}$ log log n = O(n), therefore the total cost is simply O(n).

Q2. (10 points) Let *A* be an array of *n* integers, and let *R* be the range of values in *A*, i.e., $R=max(A)-min(A)$. Give an $O(n+R)$ time algorithm to sort all the values in A.

Let m = min(A), which can be computed in O(n) time.

Since there are at most R+1 distinct values, we keep an array C of counters of size R+1 all initially zero. We make one pass through the array A, compute V[i] = A[i]-m, and increment the count at index C[V[i]].

This takes time O(n). Note that the values V[i] range between 0 to R, so all we are doing is checking how many of these values are seen in the array A after subtracting the min value.

Next, we make a pass through the array C in increasing order, and print out C[i] occurrences of item i+m. This takes time O(R) and prints out all values in A in sorted order. The total time is O(n+R).

Example: A = [5, 15, 13, 6, 5, 8, 6, 8, 10]
R = 15-5 = 10. Min value m=5.
The transformed values of A are given as V = [0, 10, 8, 1, 0, 3, 1, 3, 5]
Now the count array C is given as: C[0:2, 1: 2, 3: 2, 5:1, 8:1, 10:1]
So we print out: 0+5, 0+5, 1+5 1+5, 3+5, 3+5, 5+5, 8+5, 10+5 = 5, 5, 6, 6, 8, 8, 10, 13, 15

Q3. (15 points) Let A be an array of n distinct integers. Consider an algorithm to find the minimum value, where we pair up the elements, and retain the smaller of the values from each pair. This will result in an array of half the size (actually the resulting size will be $\lceil n/2 \rceil$). We can then recursively apply the same approach, until we get a final array with just two elements. We compare these two values and return the minimum of those values as the answer. Answer the following questions:

a. How many comparisons are done in the above algorithm in the worst case.
The worst case happens when n is a power of 2.
With n items, we do n/2 comparisons in the first call, yielding a new array of size n/2
With n/2 items, we do n/4 comparisons in the second call, yielding an array of size n/4
And so on
Until we have only 2 items left, and we do 1 comparison, and return the min element.
Total cost: n/2 + n/4 + n/8 + … + 2 + 1 = $\sum_{i=1}^{\log n-1} 2^i = (2^{\log n} - 1)/(2-1) = n-1$
There are n-1 comparisons in the worst case.

b. Show how to modify/extend this method to find the second smallest element.
Since we always return the minimum from a given comparison, the only way that the second smallest element will ever be eliminated is if it is compared directly to the min element, otherwise, the second smallest element will trickle down the levels, until it is compared to the min element at some point.
So all we have to do is, trace back the elements that min is ever compared with, and just pick the min element from that set.

Example: A = [5, 9, 8, 6, 2, 10, 3, 7]
We just take pairs of elements and compare then, recording the smaller one each time.
So, we compare [(5,9), (8,6), (2,10), (3,7)] and we get [5, 6, 2, 3]
Next we compare [(5,6), (2,3)] to get [5, 2]
Finally we compare [(5,2)] to get 2 as the min element. We can see we did 7 comparisons.

Now, was compared with the following elements: 5, 3, and 10. The min of these is 3, so 3 is the 2nd smallest element.

c. Prove that we can find the second smallest element in $n+\lceil \log(n) \rceil - 2$ comparisons in the worst case.
Note that the above questions are not asking for the big-Oh complexity, but rather the exact number of comparisons. For example, it is easy to find the 2nd smallest element in at most 2n comparisons, but that is $n+n$ which is larger than $n+\lceil \log(n) \rceil - 2$ comparisons.

Since the recursion depth is $\lceil \log(n) \rceil$ in the worst case, we have at most $\lceil \log(n) \rceil$ other elements that min was compared with (we can omit the last level that only has min as the 1 final element).
Now finding the min on an array of size n takes n-1 comparisons, so finding the min of these $\lceil \log(n) \rceil$ elements takes $\lceil \log(n) \rceil - 1$ comparisons. So the total number of comparisons is
n-1 + $\lceil \log(n) \rceil - 1$ = n + $\lceil \log(n) \rceil - 2$