1) a) $T(n) = 8T(n/4) + O(n)$

if we use the masters theorem then:

$d=1$, $a=8$, $b=4$

we find that:

now we see $d < \log_b a$ so we find $O(n^{\log_4 8})$

This simplifies down to $\boxed{O(n^{1.5})}$

b) $T(n) = 2T(n/4) + O(n)$

if we use masters theorem then:

$d=0.5$, $a=2$, $b=4$

we see that $d = \log_b a$

this means $\boxed{O(n^{1/2}\log_2(n))}$

c) $T(n-4) + O(n^2) = T\left(\frac{2(n-4)}{2} + O(n^2)\right)$

if we use masters theorem then:

$d=2$, $a=1$, $b=2$

we see that $d > \log_b a$

This means $\boxed{O(n^2)}$

d) $T(n^{1/2}) + O(n) = 2T\left(\frac{n^{1/2}d}{2}\right) + O(n)$

if we use masters theorem then:

$d=1$, $a=2$, $b=2$

we see that $d = \log_b a$

This means $\boxed{O(n\log_2 n)}$

2)
```
def countingSort(arr, n):
    max(arr)
    count = [arr + 1]
    for i = 0 to n
        find count of each element
        store it in new arr in specified index
        for j = 0 to max
            add j and j-1 count vals to get sum
            save in count arr
            for k = n to 0
                decrement count and copy it back
                to original array
```

3)

A) Worst Case: Every number in the array would have to be compared with every other number in the array. This would lead to a comparison count of n-1 for n numbers in an array.

B) To fine the second smallest element one way to do it would be to extend this algorithm, when you do to make it run twice you insert a line that removes the smallest element in the array and run it again which would then find the second smallest number. This isn't the fastest way but this would be the easiest. A faster way would be to find the smallest element and back track with an extra variable that find the number larger than the smallest and go back to make sure that there are no larger than that or smaller than it either.

C) To find the second smallest element in the array in a worst case of n + log(n) - 2 comparisons, you would have to save all of the initial pairs and the array that the algorithm returns from the first round. Search the pairs that the algorithm returned and find the pair with the smallest value in it, replace the other number with the one found in the second array and run that in the algorithm. This will give you a run time of n + log(n) -2 because when you run the algorithm the first time it has a run time of n, then splitting the array has a time of log(n) and the n-1 loop gets the -2. From this we have gotten a worst case of n + log(n) - 2 comparisons.