Q1-a
By Complex Induction
Basis:
if A[l, r] has length 1
Line 2 is executed and changes nothing to the single item, as wanted.

if A[l, r] has length 2
Line 4 ~ 5 is executed.
WEIRDSORT would compare the two elements in the list and place the smaller one in front of the larger one, which sorts the two items in increasing order, as wanted.

Induction Hypothesis:
Suppose that WEIRDSORT will sort the array A[p, q] in increasing order, where A[p, q] has a length k = (q - p + 1) < n.

Induction Steps:
Want to Prove: WEIRDSORT will sort A[l, r] in increasing order, where A[l, r] with length n = (r - l + 1).
For simplicity assume that the length ($r - l + 1$) of A[l, r] is a power of 3.
Since A[l, r] has length more than 2, then the algorithm goes to line 6.
Let t = $\frac{2(r-l+1)}{3}$.
Claim: After executing line 9, all the largest one-third items in A has been sorted and placed in A[r - m, r] (i.e. the last one-third of the length of A)

> Proof of claim:
> Suppose one of the largest one-third element y in A appears in A[l, l+t] (i.e. the first two-thirds of A)
> Let x is initially located at the first two-thirds of A without loss the generalization and
> x is one of the largest one-third elements of A
> ⇒ x is smaller than at most one-third elements of A (*)
> In line 8, WEIRDSORT would sort A[l, $l + t$] in increasing order. [By I.H. since t < n]
> ⇒ x would be placed at A[$r - t$, $l + t$] (i.e the **second one-third** of the length of A) [By (*)]
> In line 9, WEIRDSORT would sort A[$r - t$, r] in increasing order. [By I.H. since t < n]
> ⇒ x would be placed at A[$l + t$, r] (i.e the **last one-third** of the length of A) [By (*)]
> Therefore, x would be placed at last one-third of A.
> Since x is an arbitrary number from the largest one-third of A, all the largest one-third elements in A would appear in the last one-third of A.
> Elements in A[$l + t$, r](i.e the **last one-third** of the length of A) would be sorted in increasing order in Line 9. [By I.H.]

Hence, **up to line 9, all the largest one-third items in A has been sorted and placed in A[r - m, r].**
All the smallest two-thirds items in A are sorted and placed in A[l, $l+2m$] [By Line10+I.H. since 2m<n]
Therefore, WEIRDSORT will sort A[l, r] of length n in increasing order.

Q1-b

Running time:

# of sub problem = a = 3

n/b = size of sub problem = $\frac{2n}{3}$ => b = $\frac{3}{2}$

c*n^d = time of divide input and combine output

= time of calculating t or time of compare two numbers => constant => d = 0

a = 3 > b^d = 1

=> T(n) = $\Theta(n^{log_b a})$ = $\Theta(n^{log_{3/2}3})$ [by Master Theory]

Therefore, the running time of the algorithm is $\Theta(n^{log_{3/2}3})$

Q1-c

Worst Average Time complexity for Bubble Sort = $O(n^2)$

$O(n^2) = O(n^{log_{3/2}9/4}) < O(n^{log_{3/2}3})$ in general [since $\frac{9}{4} < 3$ and exponential function is increasing]

Therefore it would be slower than Bubble Sort.

Q2-a

Line 3:

PowerOfTenToBinary(n/2)

Proof of Correctness:

Base case n = 1:

PowerOfTenToBinary(1) = $(1010)_2 = (10^1)_{10}$ by line 1

IH: suppose PowerOfTenToBinary(n) = binary[10^n] for an arbitrary n is a power of two

IS: show PowerOfTenToBinary(2n) = binary[10^{2n}] (since by assumption n is a power of 2)

x = PowerOfTenToBinary(2n/2) = PowerOfTenToBinary(n) = binary[10^n] by line 3

PowerOfTenToBinary(2n) = FastMult(x,x) = binary[10^{2n}] showed in lecture

Therefore the algorithm is correct.

Running time:

# of sub problem = a = 1

n/b = size of sub problem = n/2 => b = 2

c*n^d = time of divide input and combine output

= time of FastMult(x,y) = $\Theta(n^{\log 2(3)})$ showed in lecture => d = $\log_2 3$

a = 1 < b^d = 3

=> T(n) = $\Theta(n^d)$ = $\Theta(n^{\log 2(3)})$ by Master Theory

Therefore, the running time of the algorithm is **$\Theta(n^{\log 2(3)})$**

Q2-b

Line 6:

SUM(FastMult(DecimalToBinary(x_L), PowerOfTenToBinary(n/2)), DecimalToBinary(x_R))

Proof of Correctness:
Base case length(x) = 1:
DecimalToBinary(x) = binary[x] by line 1
IH: suppose DecimalToBinary(x) = binary[x] for length(x) = k < n where k is a power of 2
IS: show DecimalToBinary(x) = binary[x] when length(x) = n where n is a power of 2
Claim: $x = (x_L * 10^{n/2} + x_R)$
Since x_L is the most n/2 significant digits and x_R is the least n/2 significant digits

Split line 6 in to the below equivalent 5 steps:

1. binary[x_L] = DecimalToBinary(x_L) [by IH]
2. binary[$10^{n/2}$] = PowerOfTenToBinary(n/2) [by part a]
3. binary[x_L*$10^{n/2}$] = FastMult(binary[x_L], binary[$10^{n/2}$]) [shown in lecture]
4. binary[x_R] = DecimalToBinary(x_R) [by IH]
5. binary[x_L*$10^{n/2}$ + x_R] = SUM(binary[x_L*$10^{n/2}$], binary[x_R]) [given]

Therefore DecimalToBinary(x) will return binary[x_L*$10^{n/2}$ + x_R] = binary[x] [by claim]
Therefore the algorithm is correct.

Running time:
a = # of sub problem = 2 by step1 and step4
n/b = size of sub problem =n/2 length(x_L) and length(x_R) = n/2

Before we determine d, we need to find out the running time in step 2, 3 and 5:
Claim1: if j and k are binary numbers, length(jk) <= length(j) + length(k)
Proof: suppose length(j) = m and length(k) = n
=> k <= 2^n (eg: $(111)_b = (7)_{10} <= 2^3 = 8$)
=> jk < j*2^n = j followed by n 0s
=> length(jk) <= length(j) + n = length(j) + length(k)
Claim2: if x is a decimal, length(binary[x]) <= c * length(x), c is a constant
(i.e. when decimal convert to binary, the length are bounded)
Proof: suppose length(x) = m
=> x < 10^{m+1}
=> binary[x] < binary[10^{m+1}]
=> length(binary[x]) <= length(binary[10^{m+1}]) <= length(binary[10]) * (m+1)
since: binary[10^{m+1}] = binary[10]*binary[10]*...*binary[10] (m+1 terms)
=> length(binary[10^{m+1}]) <= binary[10] * (m+1) **by claim 1**
=> length(binary[x]) <= 4(m+1) since binary[10] = 1010
=> length(binary[x]) <= c*length(x) since 4(m+1) is linear

Running time of PowerOfTenToBinary(n/2) in step 2:
$\Theta(n^{\log2(3)})$ shown in part a

Running time of FastMult((binary[x_L], binary[$10^{n/2}$]) in step3:

The running time is dependent on the input length is Θ ((length)$^{log2(3)}$)
length(binary[x_L]) <= c_1 * length(x_L) = c_1 * (length(x)/2) = c_1'n [by Claim2]
length(binary[$10^{n/2}$]) <= c_2 * length($10^{n/2}$) = c_2 * (n/2+1) <=c_2'n [by Claim2]
Let c^* = max(c_1', c_2')
Therefore, running time is:Θ ((c^*n)$^{log2(3)}$) = (c^*)$^{log2(3)}$$\Theta$($n^{log2(3)}$) $\in$ Θ($n^{log2(3)}$)

Running time of SUM(binary[x_L*$10^{n/2}$], binary[x_R]) in step 5:
Θ(max(length(binary[x_L*$10^{n/2}$]), length(binary[x_R]))) is given
max(length(binary[x_L*$10^{n/2}$]), length(binary[x_R]))
<= length(binary[x_L*$10^{n/2}$ + x_R]) = length(binary[x])
<= c * length(x) = cn [by claim 2]
Therefore, running time is: Θ(cn) $\in$ Θ(n)

Adding the running time of step 2, 3, 5 we have:
c*n^d = time of divide input and combine output = Θ($n^{log2(3)}$) + Θ($n^{log2(3)}$) + Θ(n) $\in$ Θ($n^{log2(3)}$)
=> d = $log_2 3$

a = 2 < b^d = 3
=> T(n) = Θ(n^d) = Θ($n^{log2(3)}$) [by Master Theory]
Therefore, the running time of the algorithm is **Θ($n^{log2(3)}$)**