Question 1:

1.1

- Let us prove that Radix Sort is correct for any n $numbers$ (with max digits of l) by using Induction on l
- Let us use $l(i)$ for the i-th digit we are currently on:
- First, the base case: $l(i) = 1$
- Every *number* has only 1 digit, that means the first and only digit is actually the real *number* itself, by using Radix Sort, we can sort every number properly, if the two (or more) numbers are the same (for example the first 4 and second 4 in a list), then because Radix Sort is stable and does not change the orginal order, the first 4 will still be on the front, so base case is correct.
- Second, let us assume for $l(i-1)$, Radix Sort can sort properly, then:
- There are two possible scenarios: 1.every *number* has a different number on digit $l(i)$ or 2.some or all of the $numbers$ have same number on digit $l(i)$. For the first scenario, Radix Sort will sort the i-th digit properly and we are done, because if the number on i-th digit for a is bigger than the number on i-th digit for b, a is surely bigger than b (within the first i digits). And for the second scenario, if the two (or more) $numbers$ have the same number on n-th digit, their order remains the same just like the 4 and 4 example in the base case, and because all the $l(n-1)$ digits are already sorted properly, so the two (or more) $numbers$ will also be in the correct order after Radix Sort on $l(n)$ th digit.
- So Radix Sort works well for any n natural $numbers$

1.2

- Yes, $O(l(n+d))$ is the correct time complexity for Radix Sort. Because similar to Counting Sort, for every digit we already know the runtime is $O(n+d)$ and this takes l rounds, so the total runtime is $O(l(n+d))$

Question 2:

2.1

- The runtime is $O(n+n^k)$, so when $k \leq 1$, runtime is $O(n)$; when $k \geq 1$, runtime is $O(n^k)$

2.2

$$l = log_d n^k + 1 \tag{1}$$

- When d increases, $log_d n^k$ will decrease, so l will decrease.

2.3

- When $d = 2$
$$T(n) = O(l(n+d)) \tag{2}$$
$$= O(log_2 n^k + 1)(n+2) \tag{3}$$
$$= O((kn + 2k)log_2 n + (n+2)) \tag{4}$$

- Because n is an integar bigger or equal to 2 and $k = O(1)$, So $T(n) \geq n$, if we can find a d that makes $T(n)$ infinitely close to n, then it is the optimized d we can find.

- When $d = n$:
$$T(n) = O((log_d n^k + 1)(n+d)) \tag{5}$$
$$= O(2(k+1)n) \tag{6}$$

- Because $k = O(1)$, $T(n) = O(n)$ and this is the best runtime we can find (or I can find)

2.4

- The time for conversion is $O(nlog_2n)$
- Because for base $d = 2$, a number has log_2n^k digits so to convert to base $d = 10$ we have to mutliply it log_2n^k times and this also applies to conversion from $d = 10$ to $d = n$, it takes n times, so the total time will be $T(n) = n(log_2n^k) = knlog_2n = O(nlog_2n)$

Question 3:

- Define random variable X = number of times I need to update
- Define $X_k = \begin{cases} 1 & \text{if k-th element is bigger than all elements ahead} \\ 0 & \text{if otherwise} \end{cases}$
- Then the total times for update is $X = X_1 + X_2 + ... + X_n = \sum_{k=1}^{n} X_k$
- So the expectation for k-th element would be:

$$E(X_k) = 0 \cdot P(X_k = 0) + 1 \cdot P(X_k = 1) = \frac{1}{k} \tag{7}$$

- By Linearity of Expectation, we know that:

$$E(X) = E(\sum_{k=1}^{n} X_k) = E(X_1) + E(X_2) + ... + E(X_n) \tag{8}$$

$$= 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n} \tag{9}$$

$$= ln(n) + O(1) \tag{10}$$

- So the expectation for total number of times I need to update the variable is $ln(n) + O(1)$