Task C.

Q1

Original formula: $T(n) = 7T(n/2) + cn^2$ and $T(1) = 0$

Assume $n = 2^k$, so $T(2^k) = 7T(2^{k-1}) + C4^k$, thus when $k = 1$, $T(2) = 4C$

$$
\begin{aligned}
T(2^k) &= 7T(2^{k-1}) + C4^k \\
&= 7[7T(2^{k-2}) + C4^{k-1}] + C4^k \\
&= 7^2T(2^{k-2}) + 7C4^{k-1} + C4^k \\
&= 7^2 [7T(2^{k-3}) + C4^{k-2}] + 7C4^{k-1} + C4^k \\
&= 7^3 T(2^{k-3}) + 7^2\cdot C4^{k-2} + 7\cdot C4^{k-1} + C4^k \quad \text{utill } k = 1;
\end{aligned}
$$

$T(2^k) = 7^kT(2^0) + C[7^0\cdot 4^k + 7^1\cdot 4^{k-1} + 7^2\cdot 4^{k-2} + \ldots.. + 7^{k-2}\cdot 4^2 + 7^{k-1}\cdot 4]$

$R = 7^kT(2^0) + C\cdot[7^0\cdot 4^k + 7^1\cdot 4^{k-1} + 7^2\cdot 4^{k-2} + \ldots.. + 7^{k-2}\cdot 4^2 + 7^{k-1}\cdot 4]$

$R \times \frac{7}{4} = C\cdot[7^1\cdot 4^{k-1} + 7^2\cdot 4^{k-2} + \ldots.. + 7^{k-2}\cdot 4^2 + 7^{k-1}\cdot 4 + 7^k\cdot 4^0]$

$(\frac{7}{4}R - R) \times \frac{1}{C} = 7^k\cdot 4^0 - 7^0\cdot 4^k$

$\frac{3}{4C}R = 7^k - 4^k$ because $\frac{3}{4C}$ is a constant number, it won't affect complexity, when $k = \log_2 n$

$$
\begin{aligned}
&= 7^{\log_2 n} - 4^{\log_2 n} \\
&= n^{\log_2 7} - n^2 < n^{\log_2 7}
\end{aligned}
$$

Prove $7^{\log_2 n} = n^{\log_2 7}$:

$$7^{\log_2 7} = X$$

$$\log_2 n = \log_7 X$$

$$\log_2 n = \frac{\log_2 X}{\log_2 7}$$

$$\log_2 X = \log_2 n \times \log_2 7$$

$$2^{\log_2 x} = 2^{\log_2 n \cdot \log_2 7}$$

$$X = n^{\log_2 7}$$

Thus, $O(n^{\log_2 7}) \approx O(n^{2.81}) < O(n^3)$

Q2.
Distribution sort:
The worst-case analysis for time complexity:
C(n) = $\sum_{j=0}^{n_{max}} 1$ (Initialise freqs) + $\sum_{i=0}^{n-1} 1$(compute freqs) + $\sum_{j=0}^{n_{max}} 1\ (compute\ cumulative\ freq) + \sum_{i=0}^{n-1} 1\ (copy\ values)$
= $2\sum_{i=0}^{n-1} 1 + 2\sum_{j=0}^{n_{max}} 1$
= $2O(n) + 2O(n_{max}) \in O(n)$, if $n > n_{max}$
Space complexity: $O(n) + O(n_{max})$ space.

Merge sort:
The worst-case analysis for time complexity:
C(n) = 2C(n/2) + n-1, for n>1, C (1) = 0
Thus, the worst-case time complexity is O(n*log(n))
Space complexity: O(n)
The merge sort is a stable sort algorithm and it would not care the value scope for the array. The best, average and worst-case time complexity are all the same, O(n*log(n)), and it needs another O(n) space to save a new sorted array.
(a).
For distribution sort:
Because the scope of values selection is between 0 to n^2, $n_{max} = n^2 - 0$ ($n_{max} > n$). The time
The time complexity for distribution sort is $2O(n) + 2O(n^2) \in O(n^2)$ ($n_{max} > n$)
The space complexity should be $O(n) + O(n^2)$.

	Time Complexity	Space Complexity
Distribution sort	$O(n) + O(n^2)$	$O(n) + O(n^2)$
Merge sort	O(n*logn)	O(n)

Thus, in this situation, the merge sort will be selected.

(b).
For distribution sort:
Because it assumes that every element in the array belongs to the set {0, n, 2n, 3n, ... n^2} with clear value types, although the scope for this array value is still 0 to n^2, the n_{max} is needed to been seen as n.
The time complexity for distribution sort is $2O(n) + 2O(n) \in O(n^2)$
The space complexity should be O(n) + O(n).

	Time Complexity	Space Complexity
Distribution sort	O(n)	O(n)
Merge sort	O(n*logn)	O(n)

Thus, in this situation, the distribution sort will be selected.

Reference:
[1]"MyApps Portal", Rmit.instructure.com, 2019. [Online]. Available:
https://rmit.instructure.com/courses/51421/files/8902670?module_item_id=1909569.
[Accessed: 14- Oct- 2019].

[2]"MyApps Portal", Rmit.instructure.com, 2019. [Online]. Available:
https://rmit.instructure.com/courses/51421/files/9266435?module_item_id=1947011.
[Accessed: 14- Oct- 2019].