1.

$T(n) = 3T(\frac{n}{4}) + 4n$

We have to observe the pattern of recurrence.

1. $T(\frac{n}{4}) = 3T(\frac{n}{16}) + \cancel{4}(\frac{n}{\cancel{4}}) = \underbrace{3(T)(\frac{n}{16}) + n}$
2. $T(n) = 3(3T(\frac{n}{16}) + n) + 4n$ //Substitute back
3. $T(n) = 9T(\frac{n}{16}) + 3n + 4n = 9T(\frac{n}{16}) + 7n$ //Simplify
4. Now, Let's substitute for even smaller input size: $(\frac{n}{16})$
5. $T(\frac{n}{16}) = 3T(\frac{n}{4} \cdot \frac{1}{16}) + 4(\frac{n}{16}) = \underbrace{3T(\frac{n}{64}) + \frac{n}{4}}$
6. $T(n) = 9(3T(\frac{n}{64}) + \frac{n}{4}) + 7n$ //Substitute back
7. $T(n) = 27T(\frac{n}{64}) + \frac{9n}{4} + 7n = 27T(\frac{n}{64}) + \frac{9n}{4} + 7n$. //simplify
8. Now, we can see that with each recursive step, the argument is divided by 4, but the number of recursive calls increases by a factor of 3 at each step.

So the coefficient of $T(\frac{n}{4^k})$ at the k^{th} level will be 3^k.

The non-recursive term increases by a new factor at each level. These terms can be represented as summation of the form $\sum_{i=0}^{k-1} 3^i \cdot 4^{-(i-1)}$ which grows and reflects the depth of recursion and the smaller size of a problem that has been broken down.

Therefore, we can represent the recursive pattern as:

$$\underbrace{T(n) = 3^k T(\frac{n}{4^k}) + \sum_{i=0}^{k-1} 3^i \cdot 4 \cdot (\frac{n}{4^i})} \Rightarrow$$

$$\Rightarrow \sum_{i=0}^{k-1} 3^i \cdot \frac{n}{4^{i-1}} = \frac{n}{4} \sum_{i=0}^{k-1} 3^i \cdot 4^{-i}$$

#1 Continued: We stop recurring when $T(n) = T(1)$.

$T(1) = \frac{n}{4^k} = 1 \Rightarrow n = 4^k$; $\log_4(n) = \log_4(4)^k$,

$k = \log_4 n$.

$3^k T(1) = 3^{\log_4 n}$.

$3^k = 3^{\log_4 n} \Rightarrow$ Using $a^{\log_b n} = n^{\log_b a}$,

$\Rightarrow 3^{\log_4 n} = n^{\log_4 3}$.

Let's look at the summation: This part of orig. $T(n) = 3T(\frac{n}{4}) + 9O(n)$

$$\sum_{i=0}^{k-1} \left(\frac{3}{4}\right)^i = \frac{1-\left(\frac{3}{4}\right)^k}{1-\frac{3}{4}} = \frac{1-\left(\frac{3}{4}\right)^k}{\frac{1}{4}} = 4\left(1-\left(\frac{3}{4}\right)^k\right).$$

We found out that this is a geometric series, and since $\left(\frac{3}{4}\right)^k$ decreases exponentially with k, for large n and k the summation converges to constants $\sum_{i=0}^{\infty}\left(\frac{3}{4}\right)^i \approx 4$

Therefore $T(n) = n^{\log_4 3} \cdot T(1) + O(n) \cdot 4$

? Dominant term

To determ. Dominant term: $\log_4 3 \approx 0.792$. Comparing $O(n^{\log_4 3})$ and $O(n)$.

$O(n^{0.792}) < O(n^1)$.

Therefore, $O(n)$ is a dominant term.

$\boxed{\text{We can conclude that } T(n) = \Theta(n).}$

√1 Using Master Theorem.

1) MT has 3 cases for the recurrence $T(n) = a \cdot T\left(\frac{n}{b}\right) + c \cdot n^k$.

2) We are given $T(n) = 3 \cdot T\left(\frac{n}{4}\right) + 4n$.

3) According to MT, $a = 3$, $b = 4$, $c = 4$, $k = 1$

4) Since $a < b^k$ $(3 < 4^1)$ →

MT 5) ⟺
$$\begin{cases} T(n) \in \Theta(n^k) & \text{if } a < b^k \\ T(n) \in \Theta(n^k \log n) & \text{if } a = b^k \\ T(n) \in \Theta(n^{\log_b a}) & \text{if } a > b^k \end{cases}$$

6) $T(n) \in \Theta(n^k)$ is our case.

7) $T(n) \in \Theta(n^1) \Leftrightarrow \boxed{\Theta(n)}$

2.

$\sqrt{2}$ (text)

a) $T(n) = 3T\left(\frac{n}{5}\right) + n^2$ $a=3, b=5, k=2$

Compare a and b^k, $3 < 25$

It's Case 1 of the Master Theorem.

$T(n) = \Theta(n^2)$

b) $T(n) = 4T\left(\frac{n}{3}\right) + 7n$, $a=4, b=3, k=1$

Compare a and b^k, $4 > 3^1$

It's case 3 of the MT.

$T(n) = \Theta(n^{\log_3 4})$

c) $T(n) = 5T\left(\frac{n}{4}\right) + 10$ $a=5, b=4, k=0$, since $f(n)=10$, $10 \cdot n^0$

Compare a and b^k, $5 > 1$

It's case 3.

$T(n) = \Theta(n^{\log_4 5})$

d) $T(n) = 9T\left(\frac{n}{3}\right) + n^4$

$a=9, b=3, k=4$

Compare a and b^k

$a = 9$, $b^k = 3^4 = 81$

$9 < 81$, we are in Case 1

$T(n) = \Theta(n^k) = \Theta(n^4)$

e) $T(n) = 6T\left(\frac{n}{8}\right) + n^3$

$a=6, b=8, k=3$

$a < b^k$, $\Rightarrow$ Case 1 $(6 < 512)$

$T(n) = \Theta(n^3)$.

Conclusion: Master Method applies everywhere (a–e).

3. llustrate the operation of Radix-Sort on the following list of strings using lexicographic ordering.
CAP, COL, USD, SUN, JPY, VEE, ROW, JOB, COX, LOL, RAT, WOW, DOD, CAR, FIG, PIG, VIS, LOW, LOX, VEA, CAD, DOG, TSL

1. Group and stable sort in a bucket by the third letter to get:
VEA, JOB, USD, DOD, CAD, VEE, FIG, PIG, DOG, COL, LOL, TSL, SUN, CAP, CAR, VIS, RAT, ROW, WOW, LOW, COX, LOX, JPY

2. Group and stable sort in a bucket by the second letter (keeping the previous order for ties) to get:
CAD, CAP, CAR, RAT, VEA, VEE, FIG, PIG, VIS, JOB, DOD, DOG, COL, LOL, ROW, WOW, LOW, COX, LOX, JPY, USD, TSL, SUN

3. Finally, group and stable sort in a bucket by the first letter to yield the final lexicographically sorted list:
CAD, CAP, CAR, COL, COX, DOD, DOG, FIG, JOB, JPY, LOL, LOW, LOX, PIG, RAT, ROW, SUN, TSL, USD, VEA, VEE, VIS, WOW

4. (text)
we start at hash table size M =13 and resize when an unresolvable collision occurs.
[25, 14, 9, 7, 5, 3, 0, 21, 6, 33, 25, 42, 24, 107].
Initial Table (M = 13)
1. Key: 25
Home Slot (h1):

$$h1(25) = \left(\frac{(25+19)(25+11)}{15} + 25\right) \%13 = 0$$

Collisions: 0
Final Slot: 0

2. Key: 14
Home Slot (h1):

$$h1(14) = \left(\frac{(14+19)(14+11)}{15} + 14\right) \%13 = 4$$

Collisions: 0
Final Slot: 4

3. Key: 9
Home Slot (h1):

$$h1(9) = \left(\frac{(9+19)(9+11)}{15} + 9 \right) \%13 = 7$$

Collisions: 0
Final Slot: 7

4. Key: 7
 Home Slot (h1):

$$h1(7) = \left(\frac{(7+19)(7+11)}{15} + 7 \right) \%13 = 12$$

Collisions: 0
Final Slot: 12

5. Key: 5
 Home Slot (h1):

$$h1(5) = \left(\frac{(5+19)(5+11)}{15} + 5 \right) \%13 = 4$$

Collision: 1, Slot 4 is occupied.
Probe Sequence:
Step size = Reverse(5) = 5
Using general formula of double hashing:

$$\text{Next Slot} = (\text{Home Slot} + i \cdot \text{Step Size})\%M,$$

Next slot = (4+1*5)5%13 = 9
Collisions: 1
Final Slot 9

6. Key: 3
 Home Slot (h1):

$$h1(3) = \left(\frac{(3+19)(3+11)}{15} + 3 \right) \%13 = 10$$

Collisions: 0
Final Slot: 10

7. Key: 0
 Home Slot (h1):

$$h1(0) = \left(\frac{(0+19)(0+11)}{15} + 0 \right) \%13 = 0$$

Collision: Slot 0 is occupied.
Probe Sequence:
Step size = Reverse(0) = 0 (invalid step size).

Resize Triggered: New size M = 29 .

Resize and Rehash to M = 29

Hash Table (M = 13) Before Resizing

Slot	Key	Collisions	Probe Sequence
0	**25**	0	[0]
1	-	-	-
2	-	-	-
3	-	-	-
4	**14**	0	[4]
5	-	-	-
6	-	-	-
7	**9**	0	[7]
8	-	-	-
9	**5**	1	[4 → 9]
10	**3**	0	[10]
11	-	-	-
12	**7**	0	[12]

Rehash existing keys into the new table.

Insertion into Resized Table (M = 29)

8. Key: 21

Home Slot (h1):

$$h1(21) = \left(\frac{(21+19)(21+11)}{15} + 21\right) \%29 = 19$$

Collisions: 0

Final Slot: 19

9. Key: 6

Home Slot (h1):

$$h1(6) = \left(\frac{(6+19)(6+11)}{15} + 6\right) \%29 = 5$$

Collisions: 0
Final Slot: 5

10. Key: 33
Home Slot (h1):

$$h1(33) = \left(\frac{(33+19)(33+11)}{15} + 33 \right) \%29 = 11$$

Collision: Slot 11 is occupied.
Probe Sequence:
Step size = Reverse(33) = 33 → ((11 + 33) % 29 = 15).
Collisions: 1
Final Slot: 15

11. Key: 25
Home Slot (h1):

$$h1(25) = \left(\frac{(25+19)(25+11)}{15} + 25 \right) \%29 = 14$$

Collision: Slot 14 is occupied.
Probe Sequence:
Step size = Reverse(25) = 52 → ((14 + 52) % 29 = 8).
Collisions: 1
Final Slot: 8

12. Key: 42
Home Slot (h1):

$$h1(42) = \left(\frac{(42+19)(42+11)}{15} + 42 \right) \%29 = 25$$

Collisions: 0
Final Slot: 25

13. Key: 24
Home Slot (h1):

$$h1(24) = \left(\frac{(24+19)(24+11)}{15} + 24 \right) \%29 = 8$$

Collision: Slot 8 is occupied.
Probe Sequence:
Step size = Reverse(24) = 42 → ((8 + 42) % 29 = 21).
Collisions: 1
Final Slot: 21

14. Key: 107
Home Slot (h1):

$$h1(107) = \left(\frac{(107+19)(107+11)}{15} + 107 \right) \%29 = 25$$

Collision: Slot 25 is occupied.
Probe Sequence:
Step size = Reverse(107) = 701 → ((25 + 701) % 29 = 6).
Collision at slot 6 → Next slot: ((6 + 701) % 29 = 1).
Collision at slot 1 → Next slot: ((1 + 701) % 29 = 6).
Collisions: 2
Final Slot: 6

Final Hash table:

Slot	Key	Slot	Key	Slot	Key
0	-	10	-	20	-
1	**5**	11	**14**	21	**24**
2	-	12	-	22	-
3	-	13	**0**	23	**3**
4	-	14	**25**	24	-
5	**6**	15	**33**	25	**42**
6	**107**	16	-	26	-
7	-	17	**9**	27	-
8	**25**	18	-	28	-
9	**7**	19	**21**		

Collision counter is independent for each key.

7. (text) Complexity analysis.

4. (text) Double hashing.

Time complexity:

Since hash table uses hash code for each look up, usually time complexity for each operation besides resizing is O(1). However, if collisions are present, the worst-case complexity for each element collision

occurs, which would require $O(n^2)$. where n is the size of the input array.
So, while double hashing improves the average case by reducing clustering, the theoretical worst-case time complexity remains $O(n^2)$ due to the possibility of needing to probe all slots for each insertion.
Space complexity:

The table size M is growing proportional to the input array size, so even after resizing its $O(2*M) = O(M)$. All calculations and collision resolutions are $O(1)$ time.

5.(code)
Time complexity:
$O(n*m)$, where n = number of strings in the array and m is the maximum length of the longest string in the array.
counting sort $O(m)$ per each character position:
Counting Occurrences: $O(n)$, where
- Cumulative Sum Calculation: $O(1)$
- Building Output Array: $O(n)$.
- Copying Output Back: $O(n)$.
- Radix sort loop runs $O(m)$

Total for all iterations: $O(maxLen \times n)$.

Space Complexity

Worst-Case Space:
$O(n*m)$, where n is a number of strings in array and m is the maximum length of the longest string in the array.
Input array stores n strings, each of length up to max length.
Output Array: $O(n)$ (used in countingSort).
countMap: $O(1)$ (fixed 53 entries, independent of n).
Auxiliary Space: $O(1)$ (no recursion or other significant storage).

6. Word pattern (code)
Time complexity: the worst time complexity is $O(n^2)$ where n is the is the length of the pattern input string which is equal to the word count in the s. The worst case is when all words or charahters collide in the hashtables, or each character is mapped to the same bucket. The hash Table lookups / insertions cost $O(n)$, and in the worst case for each operation from n, collision would cost another $O(n)$ probes, resulting in $O(n^2)$ time complexity.

Space complexity: HashTables store up to n unique character-word pairs $O(n) + O(n)$ in the worst case. Storing the split words requires $O(n)$ space.