Certainly! Let's break down the explanation:

\*\*Given Recurrence Relation:\*\*

\[ T(n) = T(n-1) + n\sqrt{n} \]

\*\*Expansion:\*\*

We expanded the recurrence relation iteratively:

\[ T(n) = T(n-1) + n\sqrt{n} \]

\[ T(n) = T(n-2) + (n-1)\sqrt{n-1} + n\sqrt{n} \]

\[ \vdots \]

\[ T(n) = T(2) + \sum\_{i=2}^{n} i\sqrt{i} \]

\*\*Approximation:\*\*

We approximated the sum in the expression. The sum \(\sum\_{i=2}^{n} i\sqrt{i}\) can be approximated as \(\frac{2}{3}(n^{\frac{3}{2}} - 2^{\frac{3}{2}})\) using mathematical techniques.

\*\*Final Result:\*\*

\[ T(n) \approx T(2) + \frac{2}{3}(n^{\frac{3}{2}} - 2^{\frac{3}{2}}) \]

\*\*Asymptotic Bounds:\*\*

Finally, after simplification and approximation, we found that the asymptotic upper and lower bounds for the given recurrence relation are \(T(n) = \Theta(n^{\frac{3}{2}})\).

In simpler terms, the time complexity of the algorithm described by the recurrence relation is proportional to \(n^{\frac{3}{2}}\) in the big-theta notation.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Let's analyze both expressions:

1. \*\*For \(5^{n+1} = O(5^n)\):\*\*

To check if \(5^{n+1} = O(5^n)\), we need to see if there exist constants \(c\) and \(n\_0\) such that \(5^{n+1} \leq c \cdot 5^n\) for all \(n \geq n\_0\).

Simplifying the expression:

\[ 5^{n+1} = 5 \cdot 5^n \]

Now, let's choose \(c = 5\) and \(n\_0 = 1\). For all \(n \geq 1\), we have:

\[ 5^{n+1} = 5 \cdot 5^n \leq 5 \cdot 5^n \]

Therefore, \(5^{n+1} = O(5^n)\).

2. \*\*For \(7^{2n} = O(7^n)\):\*\*

To check if \(7^{2n} = O(7^n)\), we need to see if there exist constants \(c\) and \(n\_0\) such that \(7^{2n} \leq c \cdot 7^n\) for all \(n \geq n\_0\).

Simplifying the expression:

\[ 7^{2n} = (7^n)^2 \]

Now, let's choose \(c = 1\) and \(n\_0 = 1\). For all \(n \geq 1\), we have:

\[ 7^{2n} = (7^n)^2 \leq 7^n \cdot 7^n = 7^n \cdot 7^n \]

Therefore, \(7^{2n} = O(7^n)\).

In both cases, the given expressions satisfy the conditions for Big O notation.

The recurrence relation for the recursive version of insertion sort can be expressed as follows:

\[ T(n) = T(n-1) + \Theta(n) \]

Here, \( T(n) \) represents the time complexity of sorting an array of size \( n \). The term \( T(n-1) \) represents the time required to sort the array of size \( n-1 \) recursively, and \( \Theta(n) \) represents the time taken to insert the \( n \)-th element into the already sorted array of size \( n-1 \).

To solve this recurrence, we can use the method of iteration or unwinding:

\[ T(n) = T(n-1) + \Theta(n) \]

\[ = T(n-2) + \Theta(n-1) + \Theta(n) \]

\[ = T(n-3) + \Theta(n-2) + \Theta(n-1) + \Theta(n) \]

\[ \vdots \]

\[ = T(1) + \Theta(2) + \Theta(3) + \ldots + \Theta(n-1) + \Theta(n) \]

Now, we can simplify the summation term:

\[ \Theta(2) + \Theta(3) + \ldots + \Theta(n-1) + \Theta(n) = \Theta\left(\frac{n(n+1)}{2}\right) \]

Therefore, the final expression for the time complexity of the recursive version of insertion sort is:

\[ T(n) = \Theta\left(\frac{n(n+1)}{2}\right) \]

Hence, the time complexity is \( \Theta(n^2) \) for the recursive version of insertion sort.

Let's analyze each statement one by one:

### (1) \( (n+7)^2 \in O(n^2) \)

To prove or disprove this statement, we need to find constants \( c \) and \( n\_0 \) such that \( (n+7)^2 \leq c \cdot n^2 \) for all \( n \geq n\_0 \).

\[ (n+7)^2 = n^2 + 14n + 49 \]

We can see that for any \( c \) and \( n\_0 \), there exist \( n \) values where \( (n+7)^2 > c \cdot n^2 \). Therefore, the statement is \*\*disproved\*\*.

### (2) \( n \in O(n) \)

This statement is trivially true since \( n \) is \( O(n) \) by definition. There exist constants \( c \) and \( n\_0 \) such that \( n \leq c \cdot n \) for all \( n \geq n\_0 \).

### (3) \( n! \in O((n+1)!) \)

Let's analyze \( \frac{(n+1)!}{n!} \):

\[ \frac{(n+1)!}{n!} = n+1 \]

To prove this statement, we need to find constants \( c \) and \( n\_0 \) such that \( n! \leq c \cdot (n+1)! \) for all \( n \geq n\_0 \).

\[ n! \leq c \cdot (n+1)! \]

\[ 1 \leq c \cdot (n+1) \]

Now, we can choose \( c = 1 \) and \( n\_0 = 0 \). For all \( n \geq 0 \), the inequality holds.

Therefore, the statement is \*\*proved\*\*.

In conclusion:

- (1) is \*\*disproved\*\*.

- (2) is \*\*true\*\*.

- (3) is \*\*proved\*\*.

The provided algorithm is a simple linear search in an array. The running time can be analyzed in terms of the Big Theta (\(\Theta\)) notation for best, average, and worst cases.

### Best Case:

The best-case scenario occurs when the element 'x' is found at the beginning of the array, i.e., at index 0.

- \*\*Running Time:\*\* \( \Theta(1) \) (constant time)

### Average Case:

In the average case, we assume that the element 'x' is equally likely to be found at any position in the array.

- \*\*Running Time:\*\* \( \Theta\left(\frac{n}{2}\right) \) (linear time)

### Worst Case:

The worst-case scenario occurs when the element 'x' is not present in the array, and we have to traverse the entire array.

- \*\*Running Time:\*\* \( \Theta(n) \) (linear time)

In summary:

- \*\*Best Case:\*\* \( \Theta(1) \)

- \*\*Average Case:\*\* \( \Theta\left(\frac{n}{2}\right) \)

- \*\*Worst Case:\*\* \( \Theta(n) \)

The given recurrence relation is \( T(n) = T\left(\frac{9n}{10}\right) + n^2 \sqrt{n} \).

We can analyze this recurrence relation using the Master Theorem.

The standard form of the Master Theorem is \( T(n) = aT\left(\frac{n}{b}\right) + f(n) \), where \( a \) is the number of recursive subproblems, \( b \) is the factor by which the problem size is reduced, and \( f(n) \) is the cost of dividing and combining the solutions.

In the given recurrence, we have:

- \( a = 1 \) (one recursive subproblem)

- \( b = \frac{10}{9} \) (problem size is reduced by \( \frac{10}{9} \))

- \( f(n) = n^2 \sqrt{n} \)

Now, let's compare \( f(n) \) with \( n^{\log\_b a} \):

\[ n^{\log\_b a} = n^{\log\_{\frac{10}{9}} 1} = n^0 = 1 \]

Comparing \( f(n) \) with \( n^{\log\_b a} \):

\[ f(n) = n^2 \sqrt{n} \]

Since \( f(n) \) is polynomially larger than \( n^{\log\_b a} \), we are in case 3 of the Master Theorem.

The Master Theorem case 3 states that if \( f(n) \) is polynomially larger than \( n^{\log\_b a} \), then the solution has the form \( \Theta(f(n)) \).

Therefore, the running time of the given recurrence relation is \( \Theta(n^2 \sqrt{n}) \).

To determine the time complexity of the given function \( f(n) = \frac{n^2}{2} - 9n \), let's analyze its dominant term as \( n \) approaches infinity.

The given function is a quadratic function with \( n^2 \) term being the dominant term. Therefore, the time complexity is \( O(n^2) \).

The necessary values for the computation are:

- Dominant term: \( n^2 \)

- Coefficient for the dominant term: \( \frac{1}{2} \)

So, the time complexity is \( O(n^2) \) with a coefficient of \( \frac{1}{2} \).