1.

Code in Java

```
public boolean decideSuper(String w) {
	String y;
	for (int i = 0; i < w.length+1; i++) {
		for (int j = 0; j < w.length+1; j++) {
			y = w.substring( i , j ); // iterate over all possible substrings
			if ( DecideL(y) == true ) { // if DecideL returns true for a substring, the
			    return true;		// string w is a superstring
			}
		}
	}
	return false;
}
```

This algorithm works by iterating over all possible substrings y of the input string w. For each substring, it calls the DecideL subroutine to check if y is in L. If DecideL(y) returns True for any substring y, then w is a superstring of L, and the algorithm returns True. If no such y is found after checking all substrings, the algorithm returns False, indicating that w is not a superstring of L.

It is correct because it iterates over all possible substrings and therefore exhausts all possibilities.

2.

Code in Java

```
public boolean recognizeSUB(String w) {
   for (String x : Σ* ) { // iterate over all x in Σ*
      for (String z : Σ* ) { // iterate over all z in Σ*
         if (recognizeL(x + w + z)) {
            return true;
         }
      }
   }
   return false;
}
```

This algorithm works by iterating over all possible strings x and z in Σ*. For each pair of strings, it concatenates x, w, and z and calls the RecognizeL subroutine. If RecognizeL returns True for any such concatenation, then the algorithm returns True, indicating that w is a substring of a string in L. If no such concatenation of w with x and z belongs to L, the algorithm returns False.

This algorithm is correct because it checks every possible way that w could be a substring of a string in L and correctly identifies whether such a string exists.