```
//-------------------------------------------------------------------------
Search( query ) {
1.  hits = SearchTopStrand( query )

2.  if( doubleStranded && !Palindromic(query) )
3.    reverseHits = SearchTopStrand( ReverseComplement(query) )
4.  else
5.    reverseHits = [ ]

6.  return [ hits, reverseHits ]
7.  }
//-------------------------------------------------------------------------
SearchTopStrand( query ) {
1.  length = l query l

    //Pad query to length K with N's if necessary.
2.  if( length < K ) {
3.    query = PadQuery( query, K )
4.    length = K
5.    kmers = [ query ]
6.  }

    //Otherwise divide query into K-mers.
7.  else {
8.    kmers = DivideQuery( query, K )
9.    //Sort by increasing frequency in subject DNA sequence.
10.   kmers = SortKMers( kmers, kmerFrequencies )
11. }

12. if( VeryDegenerate( kmers[1] ) )
13.   return BruteForceSearch( query )  //Examine DNA sequence directly.

    //Fetch first K-mer intra-chunk position values from index and
    //convert to 4-byte absolute position values.
14. newList = Hits( kmers[1] );
15. workingList = AbsolutePositions ( newList );

    //Cull working list using subsequent K-mers.
16. for( i = 2 : l kmers l ) {
17.   if( VeryDegenerate( kmers[i] ) ) {
18.     CheckHits( workingList, query )  //Examine DNA sequence directly.
19.     break
20.   }
21.   else {
22.     newList = Hits( kmers[i] );
23.     workingList = Intersect( newList, workingList );
24.     if( l workingList l == 0 ) break
25.   }
26. }

    //If subject sequence is circular, search for hits spanning the
    //origin and append these hits to the working list.
27. if( circular )
28.   workingList = AppendWrapAroundHits( workingList, query )

29. return workingList;
30. }
//-------------------------------------------------------------------------
```