These problems relate to RNA secondary structure prediction (aka “RNA folding”), described in lecture and section 6.5 of the text. In short, you will implement Nussinov’s algorithm and its associated “traceback” routine.

**Core Algorithm:** You should have a subroutine named “Nussinov” that has a single parameter which is a string of letters $x_1 x_2 \ldots x_n$ from the 4 letter alphabet \{A,G,C,U\} (all uppercase), as in line [1] below. This routine should calculate the Nussinov OPT table, call your traceback routine, and print the outputs specified below, including the time it took to do all this.

**Traceback:** Also provide a suitable traceback routine that will generate one of the optimal structures corresponding to your OPT table. I say “one of” since there may be different structures with equal numbers of pairs, often slight variants of each other. Giving any one of them is OK. The structure will be represented by a string of parens and dots, of the same length as the RNA string. A dot in the structure line means that the corresponding position in the RNA is unpaired; a left paren means it is paired with a position to its right, marked by a right paren. Furthermore, parens must be properly balanced/nested, (a consequence of the “no pseudoknots” rule) so specific paired positions are marked by “matching” left/right parens.

**Output:** For each “Nussinov” call, print to standard out one line containing the input, like [1] below, a second line containing (one of) its optimal structure(s) (i.e., output of your traceback), formatted as in [2] below and vertically aligned with the RNA sequence, plus a third line giving (i) the length of the input, (ii) the total number of pairs in that structure, and (iii) the time, in seconds (or fractions thereof) as in [3] (note that the structure/count shown are not optimal for this sequence). Additionally, for a length $n$ input, if $n < 25$, print the $n \times n$ OPT matrix calculated by Nussinov’s algorithm; print one line per row with $n$ white-space-separated integer values per line, preferably keeping columns vertically aligned. (v) Follow all of this by one blank line. E.g.:

```
GCCCACCUCGAAAAGACUGGAUGACCAUGGGCCAUGAUU [1]
((((....((......)).(((....))).))))........ [2]
Length = 40, Pairs = 9, Time = 0.00006 sec [3]
```

The figure at right reflects the structure above (not the optimal one for this sequence).

**Input:** Your main program should read a sequence of lines from “standard input” each containing one such string, and call “Nussinov” on each. Different lines may be of different lengths (different “$n$”). For timing purposes, you should also generate random sequences of length, say, $n = 2^k$ for $4 \leq k \leq 12$ or more, with A,C,G,U independently equally likely in each position. Generate at least one sequence of each length, preferably several. Call “Nussinov” on each. Generating and processing these random sequences should be optional, and by default this option should be “off” so that our hard-working TAs don’t have to endure the wait time, but it should be obvious how to enable it. E.g., your main program might look like:

```c
while(!end_of_file(STDIN)){
    seq = read_a_line();
    Nussinov(seq);
}
if(FALSE){
    for(k=4; k<=12; k++){
        Nussinov(random_seq(2^k))
        Nussinov(random_seq(2^k))
        Nussinov(random_seq(2^k))
    }
}
```
What You Need To Do:

1. [2 points] The conventions used to index the OPT table differ between the book and the slides; clearly state which you are using.

2. [28 points] Implement the Nussinov algorithm for calculating \( OPT[i,j] \).

3. [20 points] Devise and implement an algorithm to construct and print the structure (i.e., the string of parens and dots). This is a “traceback,” similar to ones we’ve seen with other dynamic programming algorithms. I strongly recommend that you look for a recursive algorithm to do this, but it is not required. If you’d like, you may create auxiliary data structures while you’re building OPT to facilitate the traceback, but I recommend against this approach.

   As stated above, print the input, with the structure aligned vertically below it, and also print the number of pairs. Additionally, print the OPT matrix if \( n \leq 25 \). All output should go to standard out.

4. [20 points] Write a description of your traceback algorithm, explaining how it works/why it is correct.

5. [10 points] Analyze (separately and collectively) the (big-O) run time of the algorithms in problems 2 and 3.

6. [10 points] Measure the actual run time of your algorithm (total time for both parts) on random RNA sequences of length 16–4096, say, plot them on a graph (e.g., Excel might be convenient, but is not required), and discuss how this compares to the theoretical performance predicted in problem 5. For some tips on how to do the timing, see the FAQ page.

Test Cases: TBD

Language: You may use C, C++, C#, Haskell, Java, Lisp, ML, Perl, Python, R, or Ruby; talk to me before beginning if you prefer something else.

What/How To Turn In: TBD