Name: _____ *Student Number:* _____

CSE 417
Algorithms and Computational Complexity
Sample Final Exam

W. L. Ruzzo Winter 2019

Take "Sample Exam" with a big grain of salt. These questions are similar to ones I have used, but are not all from the same year, are not necessarily representative of questions I am likely to use, and it's probably twice as long as it should be..., *but* they do cover appropriate material.

The final exam is comprehensive: it will cover *pre*-midterm material, but I have not included any such questions here; see the "sample midterm."

1. Consider the following recursive procedure (which probably doesn't compute a useful function, but that doesn't really matter for purposes of this problem) which is defined for $n, k \geq 0$.

```
Function Weird(n, k)
    if (k = 0 or k = n) then
      return (1)
    else
      if (n = 0 or k > n) then
        return (42)
      else
        return (4 * Weird(n − 1, k) + 2 * Weird(n − 1, k − 1))
      end if
    end if
end
```

   (a) Draw a function "call tree" (as in the lecture slides for the Fibonacci example) for $\texttt{Weird}(4, 2)$, and highlight at least one instance in it where a function call is redundantly evaluated. Based on this, explain why you think the resulting runtime of the procedure above will not be polynomial in n and k.

   (b) Use *Dynamic Programming* to rewrite this function to be much more efficient.

   (c) What are the *Time* and *Storage Space* used by your new algorithm?

```
Function Weird(n,k)
for i = 0 to n:
for j = 0 to k:                                   runtime O(nk) storange O(nk)

if(k=0 or k=n) then
    W[ i ][ j ] =  (1)
else
  if(n=0 or k>n) then
       W[ i ][ j ] = (42)
else
     W[ i ][ j ] =  (4 * W[n – 1][ k] + 2 * W[n – 1][ k – 1])
end if end if

return W[ n ][ k ]
end
```

2. Something about knapsck/subset sum, perhaps similar to HW 6 #2 or #3.

Name: _____ Student Number: _____ 3

3. Given the RNA string A U G C G C A U, (and recalling that A pairs with U, C with G),

(a) Fill in the "Opt" matrix as calculated by the Nussinov-Jacobsen RNA folding algorithm:

i \ j	A 1	U 2	G 3	C 4	G 5	C 6	A 7	U 8
A 1	—	0	0	0	0	0	1	2
U 2	—	—	0	0	0	0	1	1
G 3	—	—	—	0	0	0	0	0
C 4	—	—	—	—	0	0	0	0
G 5	—	—	—	—	—	0	0	0
C 6	—	—	—	—	—	—	0	0
A 7	—	—	—	—	—	—	—	0
U 8	—	—	—	—	—	—	—	—

(b) How many pairs are present in the folding with the most pairs, and how is this reflected in the table above? 2; this is scene in entry i = 1 and j = 8, i.e. when the start of the sequence is the being of the RNA string and the end of the sequence is the end of the RNA string

(c) Show the optimal pairing (parens and dots):

A	U	G	C	G	C	A	U
(	(	.	.	.	.	)	)

(d) Suppose you are given the "Opt" matrix for the folding of an *unknown* RNA string x. There are many different strings that could have given this Opt matrix. Among all such strings, (a) it may be the case that all of them have a first letter that is paired with its last letter in every optimal pairing of that string. Alternatively, perhaps (b) none of them have this property, or perhaps (c) some do but others don't. I.e., even without knowing x, you may be able, from Opt, to deduce that x_1 and x_n necessarily are paired with each other, or necessarily not paired with each other, or perhaps Opt doesn't contain enough information to allow you to make this decision. Perhaps surprisingly, you can answer this by looking only at the first row of Opt, the last column of Opt, and Opt[2,n-1].

For the two examples below, say which of these cases (a, b or c) applies, and briefly explain why. The "…" parts of the matrices below are all zero's. *Hint: What would traceback do?*

Name: Student Number: 4

I:

0	⋯	0	1	2	3
				2	2
					1
					0
					⋮
					0

II:

0	0	0	0	0	1	1	1	1	1	1	1	2	3
												1	2
													2
													2
													2
													2
													2
													2
													1
													1
													0
													0
													0
													0

a)

If base b1 and bn are paired in the optimal pairing, than OPT[1][n] must have obtained its value from 1 + OPT[1][0] + OPT[2][n-1]

Since the first of these OPT[1][0] must clearly be zero, there cannot be a base pair between the bases starting at b1 and ending before b1…., so this yields OPT[1][n] = 1 + OPT[2][n -1]. As we can see in the matrix above, it is possible that 3 = 1 + 2 so it is possible this is the case.

Clearly OPT[1][n] did not get its value from OPT[1][n-1] so there must be a pair between bn and some other base bt.
Again OPT[1][n] = 1 + OPT[1][t-1] + OPT[t+1][n-1]. Note that any entry must be at most the value of the entry to its right by the option OPT[i][j] = OPT[i][j-1] . SImilarly, any entry must be at most the value of the entry above it because moving down a column we only reduce the length of our interval without adding any new bases.

b)

For this same reason, as 3 =/= 1 + 1 there cannot be a pair between b1 and bn

4. Define the following:

   (a) P:

   (b) NP:

   (c) A is polynomial time reducible to B ($A \leq_p B$):

   (d) A is NP-complete:

   What is the practical significance of the statement "A is NP-complete"?

a) P is the set of decision problems that can be solved in polynomial time.

b) NP is the set of decisions problems that can be verified in polynomial time with a polynomial length hint

c)

There exists a polynomial algorithm for converting every instance of decision problem A to an instance of decision problem B such that an instance in A is a YES instance to decision problem A if and only if the decision problem B that it is reduced to is also a YES instance.

d)

decision problem A is in NP (see part b above) and every decision problem in NP is polynomially reducible to A.

If A is NP-complete, then a fast (polynomial) algorithm for A will provide a fast (polynomial) algorithm for solving all problems in NP by reduction to A, thus proving P = NP.

5. The *Partition Into Triangles Problem (PITP)* is, given a $3n$-vertex undirected graph $G = (V, E)$, to decide whether there is a partition of the vertices of G into n sets $V_1, \ldots, V_n$, each of size three, with $V = \bigcup_{i=1}^{n} V_i$, and such that for each $1 \leq i \leq n$ the three edges joining each pair of vertices in V_i are present in E.

   (a) Carefully explain why PITP is in NP.

   (b) As usual, let *SAT* be the satisfiability problem for Boolean formulas. We know that SAT is NP-complete. Say which of the following two statements is known to be true, and explain why:

      i. Suppose there is a polynomial time procedure that solves PITP, using a hypothetical polynomial time subroutine for SAT. Then PITP is NP-complete.

      ii. Suppose there is a polynomial time procedure that solves SAT, using a hypothetical polynomial time subroutine for PITP. Then PITP is NP-complete.

a) Define a hint h to the PITP problem to be a sequence of numbers within the range 1,2,…,n. Since for any positive integer n, n has less than n digits we can see this hint has less than n^2 digits and is thus polynomial in length. Then let the ith number in this sequence represent the partition that vertex v_i is placed in. Using the following verifier, we can check if a given hint provides a solution in polynomial time. As PITP is polynomially verifiable, it is in NP

```
# x is the problem instance, i.e. a graph G = (V,E)
v(x, h):
    check if x is a well-formed representation of a graph
    check that  h is well-formed (i.e. includes numbers 1 thru n and each number appears exactly 3 time)

    Generate an array where each index is a set where the set in array entry i represents the current vertices in partition i.
    for vertex = 0 to length(h) - 1:
        for other_vertex in partition h[ i ]:
            if (vertex, other_vertex) NOT in E: return N
        add vertex to partition h[i]

    return YES
```

We loop over the entire hint in O(n) time. For each entry in the hint there are at most two other vertices already in the partition and we will traverse over the entire edge set to check if the edge is present. THere are at most |E| = |V|^2 = N^2 edges so overall the verifier takes at most O(N^3) time and is thus polynomial

The second one is true. A problem is NP complete if all other problems in NP are reducivle to it polynomially. Thus, all problems are reducible to SAT in poly time. If we can reduce SAT to PITP in poly time, as ii is implying, we could reduce any NP problem A into SAT and then into PITP in polynomial time. Thus, since PITP is in NP and any NP problem reduces to it, PITP is NP complete.