Worth: 5%

1. **[16 marks]**

   (a) **[5 marks]** Give a *detailed* argument that for all decision problems D and E, if $D \leq_p E$ and $E \in NP$, then $D \in NP$.

   (b) **[5 marks]** By analogy with the definition of NP-hardness, give a *precise* definition of what it means for a decision problem D to be "$coNP$-hard."

   (c) **[6 marks]** Show that if decision problem D is $coNP$-hard, then $D \in NP$ implies $NP = coNP$.

2. **[24 marks]**

   For each decision problem D below, state whether $D \in P$ or $D \in NP$, then justify your claim.

   - For decision problems in P, describe an algorithm that decides the problem in polytime (including a brief argument that your decider is correct and runs in polytime).
   - For decision problems in NP, describe an algorithm that verifies the problem in polytime (including a brief argument that your verifier is correct and runs in polytime), and give a detailed reduction to show that the decision problem is NP-hard — *for your reduction(s), you **must** use one of the problems shown to be NP-hard during lectures or tutorials.*

   For each of the following decision problems, recall that a cycle an an undirected graph is *simple* if it contains no repeated vertex or edge.

   (a) **[8 marks]** ExactCycle:

   Input: An undirected graph G and a non-negative integer k.

   Question: Does G contain some simple cycle on **exactly** k vertices?

   (b) **[8 marks]** SmallCycle:

   Input: An undirected graph G and a non-negative integer k.

   Question: Does G contain some simple cycle on **at most** k vertices?

   (c) **[8 marks]** LargeCycle:

   Input: An undirected graph G and a non-negative integer k.

   Question: Does G contain some simple cycle on **at least** k vertices?

3. **[20 marks]**

   Consider the following "Partition" search problem.

   Input: A set of integers $S = \{x_1, x_2, \ldots, x_n\}$ — each integer can be positive, negative, or zero.

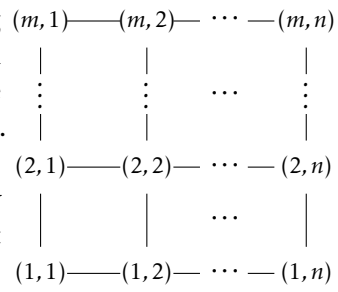
   Output: A partition of S into subsets S_1, S_2 with equal sum, if such a partition is possible; otherwise, return the special value nil. (S_1, S_2 is a "partition" of S if every element of S belongs to one of S_1 or S_2, but not to both.)

   (a) **[5 marks]** Give a *precise* definition for a decision problem "Part" related to the Partition search problem.

   (b) **[15 marks]** Give a *detailed* argument to show that the Partition search problem is polynomial-time self-reducible. (Warning: Remember that the input to the decision problem does **not** contain any information about the partition — if it even exists.)

4. **[20 marks]**

Your friends want to break into the lucrative coffee shop market by opening a new chain called *The Coffee Pot*. They have a map of the street corners in a neighbourhood of Toronto (shown on the right), and estimates $p_{i,j}$ of the profits they can make if they open a shop on corner (i,j), for each corner. However, municipal regulations forbid them from opening shops on corners $(i-1,j)$, $(i+1,j)$, $(i,j-1)$, and $(i,j+1)$ (for those corners that exist) if they open a shop on corner (i,j). As you can guess, they would like to select street corners where to open shops in order to maximize their profits!



(a) **[5 marks]** Consider the following greedy algorithm to try and select street corners.

```
C := {(i, j) : 1 ≤ i ≤ m, 1 ≤ j ≤ n}    # C is the set of every available corner
S := ∅    # S is the current selection of corners
while C ≠ ∅:
    pick (i, j) ∈ C with the maximum value of p_{i,j}
    # Add (i, j) to the selection and remove it (as well as all corners adjacent to it) from C.
    S := S ∪ {(i, j)}
    C := C − {(i, j), (i − 1, j), (i + 1, j), (i, j − 1), (i, j + 1)}
return S
```

Give a precise counter-example to show that this greedy algorithm does not always find an optimal solution. State clearly the solution found by the greedy algorithm, and show that it is not optimal by giving another selection with larger profit.

(b) **[15 marks]** Prove that the greedy algorithm from part (a) has an approximation ratio of 4. (Hint: Let S be the selection returned by the greedy algorithm and let T be any other valid selection of street corners. Show that for all $(i,j) \in T$, either $(i,j) \in S$ or there is an adjacent $(i',j') \in S$ with $p_{i',j'} \geq p_{i,j}$. What does this means for all $(i,j) \in S$ and their adjacent corners?)

5. **[20 marks]**

You are using a multi-processor system to process a set of jobs coming in to the system each day. For each job $i = 1, 2, \ldots, n$, you know v_i, the time it takes one processor to complete the job.

Each job can be assigned to one and only one of m processors — no parallel processing here! The processors are labelled $\{A_1, A_2, \ldots, A_m\}$. Your job as a computer scientist is to assign jobs to processors so that each processor processes a set of jobs with a reasonably large total running time. Thus given an assignment of each job to one processor, we can define the *spread* of this assignment to be the minimum over $j = 1, 2, \ldots, m$ of the total running time of the jobs on processor A_j.

Example: Suppose there are 6 jobs with running times $3, 4, 12, 2, 4, 6$, and there are $m = 3$ processors. Then, in this instance, one could achieve a spread of 9 by assigning jobs $\{1, 2, 4\}$ to the first processor (A_1), job 3 to the second processor (A_2), and jobs 5 and 6 to the third processor (A_3).

The ultimate goal is find an assignment of jobs to processors that **maximizes** the spread. Unfortunately, this optimization problem is *NP*-Hard (you do not need to prove this). So instead, we will try to approximate it.

(a) **[15 marks]**

Give a polynomial-time algorithm that approximates the maximum spread to within a factor of 2. That is, if the maximum spread is s, then your algorithm should produce a selection of processors for each job that has spread at least $s/2$. In your algorithm you may assume that no

single job has a running time that is significantly above the average; specifically, if $V = \sum_{i=1}^{n} v_i$ denotes the total running time of all jobs, then you may assume that no single job has a running time exceeding $\frac{V}{2m}$.

Show that your algorithm achieves the required approximation ratio.

(b) **[5 marks]**

Give an example of an instance on which the algorithm you designed in part (5a) does not find an optimal solution (that is, one of maximum spread). Say what the optimal solution is in your instance, and what your algorithm finds.