Some notes on $\mathcal{P}$ **and** $\mathcal{NP}$
CSC-024, Dr. Ostheimer

1. Definitions

Definition 1. A *problem specification* is a precise description of the desired input and output for a particular problem.

Definition 2. A problem is a *decision problem* if its output is a single boolean, **True** or **False**.

Definition 3. An *algorithm description* for a problem is a precise step-by-step description of how to solve a particular problem on all valid inputs.

Definition 4. *Algorithm analysis* is the process of coming up with a worst-case time-complexity function for a particular algorithm.

Definition 5. The *worst-case time complexity function*, $f(n)$, for a particular algorithm is a function that takes as input a positive integer n and returns a real number $f(n)$ which is the time it takes, in the worst case, for the algorithm to run on any input of size n.

Definition 6. An algorithm has *worst-case polynomial time complexity* if there exists a positive integer k such that the algorithm's worst-case time complexity function $f(n)$ is $O(n^k)$.

Definition 7. A decision problem is in the class $\mathcal{P}$ if there exists an algorithm to solve it with worst-case polynomial time complexity. Such problems are called *tractable*.

Definition 8. A decision problem is in the class $\mathcal{NP}$ if there exists an algorithm to check a certificatee of **True** with worst-case polynomial time complexity.

Definition 9. A decision problem R is $\mathcal{NP}$-hard if $R \in \mathcal{P}$ implies that $\mathcal{P} = \mathcal{NP}$.

Definition 10. A decision problem R is $\mathcal{NP}$–complete if R is in $\mathcal{NP}$ and R is $\mathcal{NP}$-hard.

Remarks:

(1) Be sure that you are clear on the difference between a problem specification, and algorithm description and algorithm analysis. Students have a strong tendency to blend these three things. A problem specification answers the question "What problem are we trying to solve?". An algorithm description answers the question "How are we going to solve it?". Algorithm analysis answers the question "How long will it take to run the algorithm?".

(2) Since an algorithm will typically take longer to run on larger inputs, algorithm analysis really answers the question "How long will it take the algorithm to run on inputs of a given size?". That's why this process produces a function $f(n)$ to describe the time it takes to run the algorithm on an input of size n.

(3) Sometimes, an algorithm might take longer to run on one input of a given size than on another of the same size. Consider the decision version of the Traveling Salesman Problem, for example, and the exhaustive algorithm we came up with in class for solving this problem. For one input with 100 cities, the first tour that is tried might be under the given bound; in this case the length of just one tour will be calculated. On the other hand, for another input also with 100 cities, there might be no tours under the given bound; in this case the lengths of all $n!$ tours will be calculated.

(4) Notice that an "algorithm" (in order to be called an algorithm) must always terminate in a finite amount of time on all legal inputs. However, the time is typically unbounded: as the size n of the input approaches infinity, the time it takes to run the algorithm approaches infinity as well. Nonetheless, for any given n, the time it takes the algorithm to terminate on an input of size n is finite.

(5) Focus carefully on the difference between $\mathcal{P}$ and $\mathcal{NP}$. Notice that the $\mathcal{P}$ in both $\mathcal{P}$ and $\mathcal{NP}$ stands for *polynomial*, but notice also that the $\mathcal{N}$ in $\mathcal{NP}$ does *not* stand for "not": in both cases we are asserting the *existence* of a worst-case *polynomial* time algorithm. In the case of $\mathcal{P}$, it is an algorithm to *solve* the problem, and in the case of $\mathcal{NP}$, it is an algorithm to *check a certificate of* **True** for the problem.

(6) We have not defined what it means to "check a certificate of **True**" to a problem. I will have to define this for you on a problem-by-problem basis. This is because you are not ready to understand the true definition of the class $\mathcal{NP}$, so this intuitive definition is the best we can do at this point.

(7) Although we are focusing on worst-case time complexity, there are lots of other kinds of complexity: average-case time complexity (where you look at the average time it takes the algorithm to terminate on inputs of size n); worst-case or average space complexity (where you look at the amount of memory needed by the algorithm for inputs of size n); circuit complexity (where you look at various measures of the complexity of a circuit to solve the problem); and many more!

(8) It follows from the definition of $\mathcal{P}$ and $\mathcal{NP}$ that $\mathcal{P} \subseteq \mathcal{NP}$. We can't prove this since we don't have a rigorous definition of $\mathcal{NP}$, but I hope it is intuitively clear that *solving* a problem is harder than *checking a solution to a problem*, and hence that this relationship between $\mathcal{P}$ and $\mathcal{NP}$ is plausible.

(9) It is an open problem as to whether, in fact $\mathcal{P} = \mathcal{NP}$. This is one of the most famous open problems in mathematics and theoretical computer science today, and it has stumped the very best of our researchers for decades.

2. TSP – DECISION VERSION

Problem Specification

- **Input:**
  - a list of cities
  - a table with the distance between every pair of cities
  - a positive integer B
- **Output:** If there is a tour through the cities of length less than or equal to B, return **True**; otherwise, return **False**.

Algorithm Description
Try every possible tour, calculating the length of each as you go. If you encounter a tour of length less than or equal to B, stop and return **True**. If there is no such tour, return **False**.

Algorithm Analysis
Let $f(n)$ be the worst-case time-complexity function for the above algorithm, where n is the number of cities. We proved in class that $f(n)$ is $O(n(n!))$.

What it means to "check"
To check a certificate of **True** means to check that the length of a *given* tour is less than or equal to B, We saw in class that this can be done in time $O(n)$. Therefore, the Traveling Salesman Problem is in the class $\mathcal{NP}$.

Complexity
Not only is the Traveling Saleman Problem known to be in $\mathcal{NP}$, it is also known to be $\mathcal{NP}$-hard. The proof of this (big!) theorem is beyond the scope of our class.