Notation:

DTM: Deterministic Turing Machine.

NDTM: Nondeterministic Turing Machine.

DEFINITION 1. The time required by a Turing machine M on an input x is the number of steps to halting. If the machine does not halt; i.e. $M(x) = \nearrow$, then the time is ∞.

DEFINITION 2. Let f be a function from the nonnegative integers to the nonnegative integers, then we say that M operates within time $f(n)$ if for any input x, the time required by M on x is less than or equal to $f(|x|)$.

DEFINITION 3. **TIME($f(n)$)**: Complexity class of languages that are decided by multistring $(k - string)$ $(k - tape)$ DTM's operating within time $f(n)$.

DEFINITION 4. **NTIME($f(n)$)**: Complexity class of languages that are decided by $k-$string NDTM's operating within time $f(n)$.

DEFINITION 5. Suppose that for a $k-$string Turing machine M_k and an input x,

$$(s, \triangleright, x, \triangleright, \epsilon, ..., \triangleright, \epsilon) \xrightarrow{M_k^*} (H, w_1, u_1, ..., w_k, u_k),$$

where H is one of the halting states. Then the space required by M_k on x is $\Sigma_{i=1}^{k}|w_i u_i|$. If M_k is with input and output, then the space required is: $\Sigma_{i=2}^{k-1}|w_i u_i|$.

DEFINITION 6. Let $f : \mathbb{N} \longrightarrow \mathbb{N}$ and let M be a Turing machine; we say M operates within space bound $f(n)$ if the space required for any input x is less than or equal to $f(|x|)$.

DEFINITION 7. SPACE($f(n)$): Languages decided by $k-$string deterministic Turing machines with input and output that operate within space bound $f(n)$.

DEFINITION 8. NSPACE($f(n)$): Languages decided by $k-$string nondeterministic Turing machines with input and output that operate within space bound $f(n)$.

DEFINITION 9. In the following, $k \in \mathbb{N}$.

1. P = TIME(n^k) = TIME($poly$) = $\cup_{j\in\mathbb{N}}$TIME(n^j).
2. NP = NTIME(n^k) = NTIME($poly$) = $\cup_{j\in\mathbb{N}}$NTIME(n^j).
3. EXP = TIME(2^{n^k}) = TIME(exp) = $\cup_{j\in\mathbb{N}}$TIME(2^{n^j}).
4. NEXP = NTIME(2^{n^k}) = NTIME(exp) = $\cup_{j\in\mathbb{N}}$NTIME(2^{n^j}).
5. PSPACE = SPACE(n^k) = SPACE($poly$) = $\cup_{j\in\mathbb{N}}$SPACE(n^j).
6. NPSPACE = NSPACE(n^k) = NSPACE($poly$) = $\cup_{j\in\mathbb{N}}$NSPACE(n^j).
7. L (or LOGSPACE) = SPACE($\log n$).
8. NL (or NLOGSPACE) = NSPACE($\log n$).

DEFINITION 10. P**:** set of all languages decidable by polynomial-time $k-$string DTM's (i.e. TIME(n^k), $k \geq 1$).

DEFINITION 11. P**:** set of all languages decidable by polynomial-time $k-$string NDTM's (i.e. NTIME(n^k), $k \geq 1$).

Complexity Classes

A complexity class is specified by

- Model of computation. In our case, it is the multi-string Turing machine.
- Mode of computation. In our case, it is deterministic and nondeterministic.
- A bound resource. In our case, it is space and time.
- A bound. In our case, it is a function from the nonnegative integers into the nonnegative integers.

In addition to the classes we have defined earlier, there are many other classes which are well-known in computational complexity. Before, we define some of these classes, let first define the following:

TIME(f): deterministic time.

SPACE(f): deterministic space.

NTIME(f): nondeterministic time.

NSPACE(f): nondeterministic space.

Where f is a function from the nonnegative integers into the nonnegative integers.

Now remember that we have defined P to be the class of languages decided by multi-string DTM's operating within polynomial time bound. (I.e. $P = \text{TIME}(n^k)$, $k \geq 1$.) Now we will prove that acceptance is good enough for determining this class.

PROPOSITION 12. *P: set of all languages accepted by polynomial-time $k-$string DTM's.*

PROOF. It suffices to show that if L is accepted by a $k-$string DTM, then L is decided by a $k-$string DTM. So, let L be a language accepted by a DTM, M. Then, $\exists k \in \mathbb{N}$ and a positive integer c, such that if $x \in L$, then L accepts x within $T = cn^k$ steps, where n is the size of the input. Now let M' be a DTM which simulates M in everything except in the output, which is as follows: $M'(x) =$ "*yes*" if M outputs "yes" within T steps, where x is the input string (of length n). And let $M'(x) =$ "*no*" if M does not accept x at the end of T steps. □

Now remember that we have defined NP to be the class of languages decided by multi-string NDTM's operating within polynomial time bound. (I.e. $NP = \text{NTIME}(n^k)$, $k \geq 1$.) The question is: what is the relationship between P and NP? A partial answer is in the following proposition.

PROPOSITION 13. *$P \subseteq NP$.*

PROOF. The proof follows from the fact that every DTM is a NDTM. □

DEFINITION 14. A decision problem A *reduces* to a decision problem B if an instance α of A can be transformed to an instance β of B such that the answer to α is "yes" iff the answer to β is yes, and the answer to α is "no" iff the answer to β is no.

Thus, to solve A on an input x, solve B on $f(x)$, where $f(x)$ is the transformation which transforms α to β. We can state the definition of reduction as follows: Decision problem A

reduces to decision problem B if there is a function $f(x)$ such that the answer is "yes" to the instance x of A iff the answer is "yes" to the instance $f(x)$ of B and the answer is "no" to x iff the answer is "no" to $f(x)$. In terms of langauges, the definition can be stated as follows:

DEFINITION 15. Let L_1 and L_2 be two languages over an alphabet Σ. We say that L_1 is reduced in polynomial time to L_2 if there is a polynomial-time computable function $f : \Sigma^* \longrightarrow \Sigma^*$, such that $x \in L_1$ iff $f(x) \in L_2$, $\forall x \in \Sigma^*$.

REMARK 0.1. The following are the same:

- A reduces to B.
- A is reducible to B.
- There is a reduction from A to B.

REMARK 0.2. it Reduction is a very important technique in complexity theory, especially if it can be done in polynomial time. Notice that if P_1 is reducible to P_2, then P_2 is at least as hard as P_1. The following are two forms of reduction which we will use very often.

- To show that a problem P_1 can be solved in polynomial-time (i.e. $P_1 \in P$), reduce (if possible) in polynomial time P_1 to a problem P_2 which belongs to P. .
- To show that a problem $P_2 \in NP$ is $NP - complete$ (will be defined shortly), reduce (if possible) a problem P_1 which is $NP - complete$ to P_2 in polynomial time. .

Now the question is: is $NP \subseteq P$? If the answer is yes, then it will follow that $P = NP$. However, this is an open question, but people strongly believe that the answer is no. I.e. there are languages which are in NP, but not in P.

Let now introduce an important theorem to highlight a result related to this issue. But before that, we need an important definition.

DEFINITION 16. Let C be a complexity class and let $L \in C$. We say that L is $C - complete$ if all languages in C can be reduced to L. In other words, if $L' \in C$, then L' can be reduced in polynomial time to L.

DEFINITION 17. A language L is in $NP - complete$ if

(1) $L \in NP$.

(2) Every language L' in NP can be reduced in polynomial time to L.

If condition (2) holds in the last definition and we don't know whether (1) holds or not, then L is said to be $NP - hard$.

Notation: We will use NPC for $NP - complete$.

Notice that item (1) in the previous definition implies that $NPC \subseteq NP$.

REMARK 0.3. Notice that first condition of the previous definition implies that $NPC \subseteq NP$.

REMARK 0.4. All NPC problems are polynomial-time reducible to one another and so they are different faces of the same problem.

PROPOSITION 18. *If L_1 is reducible in polynomial time to L_2 and if $L_2 \in P$, then $L_1 \in P$.*

PROOF. In class. □

THEOREM 19. *Let $P_2 \in NP$. If P_1 is NPC and there is a polynomial-time reduction of P_1 to P_2, then P_2 is NPC.*

PROOF. Since $P_2 \in NP$, all we need to prove is that all languages of NP reduce in polynomial time to P_2. So, let $L \in NP$. Now since $P_1 \in NPC$, then there is a polynomial-time reduction of L to P_1. Thus, there exists a polynomial f such that if $winL$, then w is transformed to a string $x \in P_1$ within time $f(|w|)$ (notice that the length of $|x| \leq f(|w|)$. On the other hand, since there is a polynomial-time reduction of P_1 to P_2, then there exists a polynomial g such that x is transformed to a string $y \in P_2$ within time $g(f(|w|))$. Thus, the transformation of w to y is done within time $f(|w|) + g(f(|w|))$. Therefore, L reduces to P_2 in polynomial time. □

The above theorem is sometimes stated as follows:

THEOREM 20. *Let $L' \in NPC$. If L' is reducible in polynomial time to L, then*

(1) L is NP-hard.

(2) If $L \in NP$, then $L \in NPC$.

THEOREM 21. *Let $L \in NPC$. If $L \in P$, then $P = NP$. And if there exists $T \in NP - P$, then $NPC \cap P = \phi$.*

PROOF. Let $L \in NPC \cap P$. We have to prove that $NP = P$. But, since $P \subseteq NP$, then all we have to prove is that $NP \subseteq P$. So, let $H \in NP$. Now since $L \in NPC$, then there is a polynomial-time reduction of H to L. But also $L \in P$. So, by proposition 18, $H \in P$. Thus, $NP \subseteq P$.

Now assume that there is a language $T \in NP - P$. We have to prove that $NPC \cap P = \phi$. The proof is by contradiction. So, assume that $NPC \cap P \neq \phi$. This means that there exists $S \in NPC \cap P$. Therefore, buy the first part, we get that $P = NP$. Now since $T \in NP$, and $P = NP$, it follows that $T \in P$. This contradicts our assumption that $T \in NP - P$. □

The above theorem is sometimes stated in the following three forms:

THEOREM 22. *Let $L \in NPC$. Then $P = NP$ iff $L \in P$.*

THEOREM 23. *$P \neq NP \Longrightarrow NPC \cap P = \phi$.*

THEOREM 24. *If some $NP-complete$ problem is in P, then $P = NP$.*

Now we recall that if L is a language over an alphabet Σ, then the complement of L, denoted by $\overline{L}$ is defined by: $L = \Sigma^* - L$.

Next, we define complements of problems.

DEFINITION 25. Let A be a decision problem, the complement of A, denoted by A *COMPLEMENT*, is the decision problem whose answer is "yes" if the answer of A is "no" and vice versa.

Example: Recall that HAM-CYCLE is the problem: given an undirected graph G does G have a Hamiltonian cycle? The complement of HAM-CYCLE, denoted by HAM-CYCLE COMPLEMENT is the following problem:

Given a graph G, is it true that G does not have a Hamiltonian cycle.

Now we introduce a new complexity class:

DEFINITION 26. Let C be a complexity class. We define the complexity class coC to be:

$$coC = \{\overline{L} \mid L \in C\}.$$

Notice that we can say that

$$coC = \{L \mid \overline{L} \in C\}.$$

Thus, coC is the set of all languages whose complements are in C. The above definition is equivalent to the previous one.

Notation: Sometime we will write $co - C$ instead of coC.

REMARK 0.5. Notice that coC is different than $\overline{C}$. To explain this point, let C be a complexity class over an alphabet Σ, then $\overline{C} = \{H \ H \notin C\}$. Thus, it's the class of all languages over Σ that are not in C. On the other hand, coC is the complexity class of the complements of all languages that are in C. Notice also that $C \cap \overline{C} = \phi$, but $C \cap coC$ is not necessarily empty. As a matter of fact, C could be equal to coC as it is the case when $C = P$.

REMARK 0.6. Let C be a complexity class.

- If $L \in C$, then $\overline{L} \in coC$.
- If $L \in coC$, then $\overline{L} \in C$.
- If $\overline{L} \in C$, then $L \in coC$.
- If $\overline{L} \in coC$, then $L \in C$.

DEFINITION 27. A complexity class C is closed under complement if $\forall L \in C$, $\overline{L} \in C$.

Next we address the question: what is the relationship between P and coP, $PSPACE$ and $coPSPACE$, NP and $coNP$, and finally, $NPSPACE$ and $coNPSPACE$? A partial answer is in the following propositions. Before, we state these propositions, let emphasize that coC is different than $\overline{C}$ for any complexity class C.

PROPOSITION 28. *If C is a deterministic time or space complexity class, then $C = coC$.*

PROOF. Let C be a deterministic time or space complexity class, and let $L \in C$, then there exists a DTM, M, that decides L within the space/time bound of that class. Now let M' be the DTM that has the same structure as M except that the "yes" and "no" states are switched. It is easy to see that M' decides $\overline{L}$ within the same space/time bound as that of M. Hence, $\overline{L} \in C$. □

Notice that the above proposition states that all deterministic time and space complexity classes are closed under complementation (complements of languages which are memebers of these classes.) Notice also that the above proposition states that if $L \in P$, then $\overline{L} \in P$.

Thus, $P = coP$. This highlights the fact that coP is different than $\overline{P}$. Recall here that there are problems which are not in P. The following is one of them: given a TM M and an input x, does M accept x within $2^{|x|}$ steps?

The next question is: do the results which we have just stated apply to nondeterministic space/time complexity classes? A partial answer is in the following.

PROPOSITION 29. *Let C be a nondetreministic space complexity class; then $C = coC$.*

Notice that the above proposition implies that $NPSPACE = coNPSPACE$. The question whether $C = coC$, where C is a nondeterministic time complexity class, is an open question. For example, it is not known whether NP is closed under complement or not. The answer is expected to be negative. In fact, it is expected that L in $NP - Complete \Longrightarrow \overline{L} \notin NP$. This leads us to the following theorem.

THEOREM 30. *$NP = coNP$ iff there is some NPC problem whose complement is in NP.*

PROOF. ($\Longrightarrow$) : Assume that $NP = coNP$ and let $L \in NPC$. Now since $NPC \subseteq NP$, then $L \in NP$. On the other hand, since $NP = coNP$, then $L \in coNP$. Thus, $\overline{L} \in NP$.

($\Longleftarrow$) : Assume that there is $H \in NPC$ such that $\overline{H} \in NP$. We have to prove that $NP = coNP$. This can be done by proving that $NP \subseteq coNP$ and $coNP \subseteq NP$.

First we will prove that $NP \subseteq coNP$. So, let $L \in NP$. Then, since $H \in NPC$, there is a polynomial-time reduction of L to H. The same reduction is also a reduction of $\overline{L}$ to $\overline{H}$. Call the machine which performs this reduction Y_1. On the other hand, since $\overline{H} \in NP$, then there is a polynomial-time NDTM which decides $\overline{H}$. Call this machine Y_2. Now combine Y_1 and Y_2 to get a polynomial-time NDTM which decides $\overline{L}$. Thus, $\overline{L} \in NP$. Hence, $L \in coNP$.

Now we will prove that $coNP \subseteq NP$. So, let $L \in coNP$. Then, $\overline{L} \in NP$. Now since $H \in NPC$, there is a polynomial-time reduction of $\overline{L}$ to H. The same reduction is also a reduction of L to $\overline{H}$. Call the machine which performs this reduction Y_1. On the other hand, since $\overline{H} \in NP$, then there is a polynomial-time NDTM which decides $\overline{H}$. Call this machine Y_2. Now combine Y_1 and Y_2 to get a polynomial-time NDTM which decides L. Hence, $L \in NP$. □

Now what are the possible relationships between P, NP, and $coNP$? And is $NP \cap coNP = \phi$? The answer to the second is no, because we know that $P \neq \phi$ and I claim that $P \subseteq NP \cap coNP$. We proved earlier that $P \subseteq NP$. Now it remains to prove that $P \subseteq coNP$. But, if $L \in P$, then $\overline{L} \in P$. This implies that $\overline{L} \in NP$. Thus, by definition, $L \in coNP$. Notice also that $coNPC \subseteq coNP$, because if $L \in coNPC$, the $\overline{L} \in NPC$. This implies $\overline{L} \in NP$. Hence, $L \in coNP$. Now let answer the first question. We have the following 4 possibilities:

- $P = NP = coNP$.
- $P \subset NP$ and $NP = coNP$.
- $P = NP \cap coNP$ and $NP \neq coNP$.
- $P \subset NP \cap coNP$ and $NP \neq coNP$.

The last possibility is what people expect. Also they expect $NPC \cap P = NPC \cap coNP = \phi$ and $NPC \cap coNPC = NP \cap coNPC = \phi$. Notice that this means that scientists believe

that no NPC has its complement in NP, or equivelently, no NPC problem is in $coNP$. Notice also that $coNPC \subseteq coNP$. This is trivial, because if $L \in coNPC$, then $\overline{L} \in NPC$ and so $\overline{L} \in NP$ (because $NPC \subseteq NP$). But, $\overline{L} \in NP$ implies that $L \in coNP$.

PROPOSITION 31. *If $NP \neq coNP$, then $P \neq NP$.*

PROOF. The proof is by contradiction. So, assume that $P = NP$. Now since $NP \neq coNP$, then either

1. $\exists L \in NP$ and $L \notin coNP$,
   or
2. $\exists H \in coNP$ and $H \notin NP$.

Case 1: $\exists L \in NP$ and $L \notin coNP$. Now since $L \in NP$, then $L \in P$. This implies $\overline{L} \in P$, which implies $\overline{L} \in NP$. Thus, $L \in coNP$. This contradicts our assumption that $L \notin coNP$.

Case 2: $\exists H \in coNP$ and $H \notin NP$. Now since $H \in coNP$, then $\overline{H} \in NP$. This implies $\overline{H} \in P$, which implies $H \in P$. Thus, $H \in NP$. This contradicts our assumption that $H \notin NP$.

□