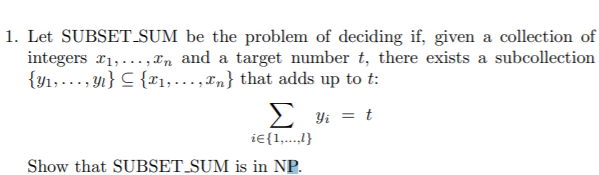
# Exercise 1



Firstly, It is trivial to see that adding a set of n numbers and comparing it with another number takes O(n).

We can create a non-deterministic Turing machine that decides the SUBSET\_SUM in the following way:

* On input (N,t): //N is a set of numbers
  + Generate all possible subsets of N
  + Check each of them:
    - If one of the subets adds up to t then accept, otherwise reject

The time complexity of the above non-deterministic Turing machine is linear O(n) where n is the size of the input set because the longest branch of the Turing machine will be O(n).

# Exercise 2

For each of the following questions, assume that A is an NP problem. In each case, the question is ”What would you tell the world?”. In other words, what is the strongest or most interesting conclusion you can draw from this information?

(a) Suppose you prove a theorem showing that a lower bound for the running time of any algorithm to solve A is Θ(2n).

(b) Suppose you find a deterministic algorithm that solves A in polynomial time.

(c) Suppose A is NP-complete and you find a deterministic algorithm that solves A in polynomial time.

(d) Suppose you prove that the satisfiability problem is reducible to A in polynomial time.

(a) NP ≠ P

(b) A is in P

(c) P == NP

(d) A is in NP

# Exercise 3

Here, suppose that A and B are NP problems, and that there is a polynomial time reduction from A to B. Again, what would you tell the world?

(a) Suppose you find a deterministic algorithm that solves B in time O(nlogn), and you know that A is NP-complete

(b) Suppose that the complexity of A is Θ(n!). (That is, we can solve A with an algorithm which runs in Θ(n!), and we know that no algorithm can do better than this.)

(a) P = NP because all problems in NP can be solved in polynomial time

(b) Then Θ(n!) is a lower bound for B. In other words B is Ω(n!). Then P ≠ NP

# Exercise 4

Show that the class NP is closed under union and concatenation.

## Closed under union

Let A be a language in NP. Let B be a language in NP. Thus, there exists a non-deterministic polynomial time Turing machine M such that L(M) = A and a non-deterministic polynomial time Turing machine M’ such that L(M’) = B.

We can create a non-deterministic Turing machine N which behaves in the following way:

* On input x:
  + Run M with input x
  + Run M’ with input x
  + If either M accepts or M’ then accept, otherwise reject

The running time of N is trivially non-deterministic polynomial and L(N) = L(M) ∪ L(M’).

## Closed under concatenation

Let A be a language in NP. Let B be a language in NP. Thus, there exists a non-deterministic polynomial time Turing machine M such that L(M) = A and a non-deterministic polynomial time Turing machine M’ such that L(M’) = B.

We can create a non-deterministic Turing machine N which behaves in the following way:

* On input x:
  + Non-deterministically split x into two substrings vw such that vw = x
  + Run machine M with string v
  + Run machine M’ with string w
  + If both machines accept then accept otherwise reject

# Exercise 5

Let DOUBLE SAT be the problem of deciding whether a boolean expression has at least two satisfying assignments. Show that DOUBLE SAT is NP-complete.

In order to show that DOUBLE SAT is NP-complete I need to show that :

* DOUBLE SAT is in NP
* Reduce an NP-complete problem to DOUBLE SAT

Proof DOUBLE SAT is NP:

On input x:

* Run SAT with x
* If SAT accepts then store the satisfying truth values P, otherwise reject
* Run a modified version of SAT which will check all possible truths values except P
* If the latter accept then accept otherwise reject

It is possible to reduce SAT to DOUBLE SAT in the following way:

Let μ be a boolean expression. Let f be a reduction which bevhaves in the following way:

f(μ) = μ ∨ xm where xm is a variable not in μ.

Clearly if μ has a solution then DOUBLE SAT will accept because there are at least 2 solutions:

* xm = True
* the solution of μ

Clearly, the reduction f is deterministic polynomial.

Therefore, DOUBLE SAT is NP-Complete