1 Polynomial Time

A reminder:

**Definition 1.0.1.** An algorithm runs in *polynomial time* if $T(n) = \mathcal{O}(n^k)$ for some $k \in \mathbb{N}$ where $n$ is the input size.

**Example 1.1.** MergeSort has $T(n) = \Theta(n \lg n) = \mathcal{O}(n^2)$ and hence MergeSort is polynomial time. ■

**Example 1.2.** Generating a list of all permutations of \{1, 2, ..., n\} has $T(n) = \Theta(n!)$ which is not polynomial time. Why not? Can you prove this? ■

Generally speaking we think of polynomial time as “fast” but depending on the coefficients, in reality polynomial time can be incredibly slow.
Definition 2.0.1. The set $P$ is the set of all decision problems such that a deterministic Turing machine can take any input and produce YES ∨ NO in polynomial time.

Example 2.1. YES ∨ NO: Given $x$, $y$ and $d$, is $d = \text{gcd}(x,y)$?

Observe that this is a YES ∨ NO question which can be solved by actually finding $\text{gcd}(x,y)$. This calculation can be done in polynomial time by the Euclidean Algorithm on a DTM. Thus we can solve the problem by simply finding the gcd and comparing it to $d$.

Thus this decision problem is (in) $P$. ■

Example 2.2. YES ∨ NO: Given two lists of integers $A$ and $B$ with the same length, do they contain the same values?

One way to solve this would be to sort both lists and then compare them value-by-value. All of these processes can be done in polynomial time on a DTM.

Thus this decision problem is $P$. ■

Example 2.3. YES ∨ NO: Given a graph $G$ on $n$ vertices, two specific vertices $s$ and $t$, and a distance $d$, is there a path of length less than or equal to $d$ from $s$ to $t$?

We can run Dijkstra’s algorithm on $G$ starting at $s$ to construct a shortest path tree from $s$, then we can check if the distance to $t$ is less than or equal to $k$. All of these processes are $O(n^2)$.

Thus this decision problem is $P$. ■

Example 2.4. YES ∨ NO: Given a partially filled $n^2 \times n^2$ Sudoku board, is there a solution?

It is unknown if this can be solved in polynomial time on a DTM.

Thus it is unknown if this decision problem is $P$. It is suspected that the answer is no. ■

Example 2.5. YES ∨ NO: Given a set of $n$ integers, is there a subset which adds to 0?

It is unknown if this can be solved in polynomial time on a DTM.

Thus it is unknown if this decision problem is $P$. It is suspected that the answer is no. ■

Example 2.6. YES ∨ NO: Given a graph $G$, does $G$ contain a Hamiltonian cycle? This is a cycle which contains each vertex exactly once.

It is unknown if this can be solved in polynomial time on a DTM.

Thus it is unknown if this decision problem is $P$. It is suspected that the answer is no. ■
In many cases if a decision version of an optimization problem is $P$ then the optimization problem itself can be solved in polynomial time.

**Example 2.7.** Let $G$ be a weighted graph and $s, t$ be connected vertices. Consider the optimization problem of finding the length of the shortest path from $s$ to $t$.

A decision version of this is to determine if there exists a path of length less than or equal to a given $k$ from $s$ to $t$.

Suppose we have a polynomial-time algorithm $\text{pathexists}(G, s, t, k)$ which answers the decision version in polynomial time. In other words within polynomial time it returns YES $\lor$ NO if a path of length less than or equal to $k$ from vertex $s$ to vertex $t$.

Since the graph is connected we know there is a path from $s$ to $t$ and the length of this path could at most be the total sum of all the edge weights. So what we can do is:

```python
function findshortestpathlength(G, s, t, k):
    max = sum of edge weights in G
    shortest = max
    for i = max down to 0
        if pathexists(G, s, t, i) then
            shortest = i
        end
    end
    return(shortest)
end
```

This function will return the length of the shortest path and will do so in polynomial time on a DTM.

Thus this optimization process can be solved in polynomial time. ■
3 NP

Before talking about \( NP \), here is \( P \) again so we can compare:

**Definition 3.0.1.** The set \( P \) is the set of all decision problems such that a DTM can take an input and produce \( \text{YES} \lor \text{NO} \) in polynomial time.

And now \( NP \):

**Definition 3.0.2.** The set \( NP \) is the set of all decision problems such that a NTM can take an input and produce \( \text{YES} \lor \text{NO} \) in polynomial time.

Now then, let’s stop to point out that this was the original definition of \( NP \) but there’s an equivalent and more modern definition of \( NP \) which is based upon verification of solutions.

**Definition 3.0.3.** The set \( NP \) is the set of all decision problems having the property that if we are given a potential witness we can verify that witness within polynomial time on a DTM.

In this case we say we have a polynomial time verifier.

Note that now we can refrain from mentioning a DTM.

**Example 3.1.** \( \text{YES} \lor \text{NO} \): Given a set \( S \) of integers can we partition \( S \) into two subsets whose sums are equal?

Suppose you give me \( S = \{1, 3, 5, 7\} \).

- If I provide you with the two sets \( \{3, 5\} \) and \( \{1, 7\} \) you can immediately say “We have a witness!”.
- If I provide you with the two sets \( \{1, 3\} \) and \( \{5, 7\} \) all you can say is “We do not have a witness”.

We see that this decision problem is \( NP \). ■

Some other problems which can be seen to be \( NP \) in a similar way:

**Example 3.2.** \( \text{YES} \lor \text{NO} \): Does a given partially filled Suduko board have a solution? ■

**Example 3.3.** \( \text{YES} \lor \text{NO} \): Given a set of \( n \) integers, is there a subset which adds to 0? ■

**Example 3.4.** \( \text{YES} \lor \text{NO} \): Given a graph \( G \) does it contain a cycle? ■

**Note 3.0.1.** I read somewhere once that some people believe that we should just use \( VP \) to mean verifiable in polynomial time on a DTM instead of using \( NP \). That way the machine is always a DTM.

Or even better, \( SP \) (for solvable in polynomial time) and \( VP \) (verifiable in polynomial time). But there we go.
4 \textbf{P v NP}

\textbf{Note 4.0.1.} First, observe that \( P \subseteq NP \). The best argument for this is to return to our original definition of \( NP \) and note that if we can solve it in polynomial time on a DTM then we can solve it in polynomial time on a NTM.

\textbf{Definition 4.0.1.} The \textit{P v NP Problem} asks whether \( P = NP \) or not. In other words is it the case that when we can verify any potential witness in polynomial time that we can also solve the decision problem in polynomial time?

This is perhaps the greatest unsolved problem in computer science. There is overwhelming evidence that \( P \neq NP \) in the sense that there are many important problems for which potential witnesses can be verified in polynomial time but no polynomial-time solution has been found. However note that this does not mean that such solutions don’t exist.
5 Problem Reduction and Equivalence

Suppose I told you the following two things:

- I have \(n^2\) motorcycles.
- I can fix one motorcycle.

You might rightfully conclude that I can fix all \(n^2\) motorcycles “in polynomial time as a function of fixing one”.

Here is a somewhat more formal way of saying this:

**Definition 5.0.1.** We say that \(A\) is polynomially reducible to \(B\) if we may, given a way to decide \(B\), construct a way to decide \(A\) using a “polynomial time wrapper”.

We’ll write \(A \leq_P B\).

**Note 5.0.1.** It’s not important how long \(B\) takes, really we’re interested in how we get from \(B\) to \(A\).

**Example 5.1.** Suppose a function \(B(n)\) decides something and returns YES or NO. Consider the following algorithm for \(A(n)\):

```python
function A(n)
    for i = 1 to n
        if B(i) = YES
            return(YES)
        end
    end
    return(NO)
end
```

Observe that \(A(n)\) is polynomially reducible to \(B(n)\). We would thus write \(A(n) \leq_P B(n)\).

Note that there might be other algorithms that do whatever \(A(n)\) does and they may do it faster, but we don’t know. What we do know, however, is that this algorithm for \(A(n)\) reduces the problem to \(B(n)\) in polynomial time so it’s essentially “easier than” \(B(n)\).

Here is another example:

**Example 5.2.** Consider these two decision problems:

- \(A\): Given an undirected graph, is there a Hamiltonian cycle in the graph?
- \(B\): Given a directed graph, is there a Hamiltonian cycle in the graph?

Note: The adjacency matrix for a directed graph has a 1 in the \(ij\) position iff there is a directed edge from vertex \(i\) to vertex \(j\).

We claim \(A \leq_P B\).
To see this, suppose we have an algorithm `exists_directed(G)` which returns TRUE if there is a Hamiltonian cycle in a directed graph `G` and FALSE if there isn’t.

Given a undirected graph `G` the adjacency matrix for `G` also represents the adjacency matrix for `G'` where `G'` is obtained from `G` by replacing each (undirected) edge with two edges, one in each direction. This takes no time. We can then apply `exists_directed(G)` to find a Hamiltonian cycle in `G'` which is also a Hamiltonian Cycle in `G`.

It follows that `A \leq_P B`.

Now then, it is widely believed that `A \notin P` and so if this is true, then `B \notin P` also.

Now we can formalize some facts:

- If `A \leq_P B` and `B \in P` then `A \in P`.
- If `A \leq_P B` and `A \notin P` then `B \notin P`.

For the mathematicians here, a nice way of thinking about this is to imagine three functions `a(x)`, `b(x)`, and `p(x)`. Suppose we know for sure that `p(x)` is a polynomial and we know that `a(x) = p(b(x))`.

Then we can say:

- If `b(x)` is a polynomial then `a(x)` is a polynomial.
- If `a(x)` is not a polynomial then `b(x)` is not a polynomial.

Note that if `p(x)` is not a polynomial we can say nothing.
6 Thoughts, Problems, Ideas

1. Explain how you know that the following decision problems are in \( P \). You don’t need to provide pseudocode, a basic explanation will suffice.

   (a) \( Y \lor N \): Given a list with \( n \) elements, is it unsorted?

   (b) \( Y \lor N \): Given the adjacency matrix for a graph with \( n \) vertices, is there one vertex which is connected to all the others?

   (c) \( Y \lor N \): Given base-10 list representations of two \( n \) digit numbers \( A \) and \( B \), is \( AB \geq 5 \cdot 10^{2n-2} \)?
   
   For example is \( 84 \cdot 23 \geq 58 \cdot 10^2 = 500 \) ?

   (d) \( Y \lor N \): Given a list with \( n \) elements, is the maximum to the left of the minimum?

2. For each of the following you are given a problem \( PROB \) and an associated decision problem \( DEC \). For each, write pseudocode to show that if \( DEC \in P \) then \( PROB \) can be solved in polynomial time.

   Note: Don’t worry about whether or not it’s true in the real world that \( DEC \in P \), just assume it is and base your pseudocode on it.

   (a) Given a simple connected unweighted graph with \( n \) vertices.

      \( PROB \): Find length of the longest path.

      \( DEC \): For any given \( k \), is there a path of length \( k \)?

   (b) Given a list \( A \) of \( n \) integers, a subset \( S \subset A \), and a target \( t \).

      \( PROB \): Assuming there is a subset of \( A \) containing \( S \) which sums to \( t \), find it.

      \( DEC \): Is there a subset of \( A \) containing \( S \) which sums to \( t \)?

   (c) Given an integer \( n \geq 2 \).

      \( PROB \): Find the smallest prime factor of \( n \).

      \( PROB \): For any given \( k \), is \( k \) prime?

3. Explain why reverse-sorting a list is polynomially reducible to sorting a list.

4. Suppose a garage contains \( n \) motorcycles each of which has 1, 2 or 4 cylinders. Explain why fixing all cylinders on all motorcycles is polynomially reducible to fixing one cylinder.

5. Suppose \( G \) is a simple graph with \( n \) vertices. Explain why counting the edges in the graph is polynomially reducible to calculating the degree of a vertex.