Problem Set 6

If you come from mathematics, as I do, you realize that there are many problems, even classical problems, which cannot be solved by computation alone.
— *Roger Penrose*

- This problem set is due **at 9:00am** on **March 24, 2025** .
- This problem set comprises 3 problems.
- Submit early. Do **not** wait till the last moment.
- Each solution should start on a new page.
- We will give full credit **only** for correct solutions that are described clearly and convincingly.
- All violations of Academic Integrity Policy (including but not limited to plagiarism) will be reported to the Academic Integrity Committee and will result in an F grade for the entire course. Please familiarize yourself with the policies and sanctions.
- The use of generative AI is **strictly prohibited** and will be heavily penalised.

Problem 6-1. Integer Partitions [25 points]

A partition of a positive integer n is a way of writing n as a sum of positive integers, where the order of the summands (called parts) does not matter. Formally, a partition of n is a sequence of positive integers $\lambda = (\lambda_1, \lambda_2, \cdots, \lambda_k)$ such that:

$$= \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_k \qquad \text{and} \qquad n = \lambda_1 + \lambda_2 + \cdots + \lambda_k$$

The λ_i are called the parts of the partition. For example, the partitions of 5 are:

$$\begin{aligned}
5 &= 1+1+1+1+1 \\
5 &= 1+1+1+2 \\
5 &= 1+2+2 \\
5 &= 1+1+3 \\
5 &= 2+3 \\
5 &= 1+4 \\
5 &= 5
\end{aligned}$$

That is, there are 7 partitions of 5. The number of partitions of n is denoted by $p(n)$. For example, $p(5) = 7$. Design an implement a function:

```
partitions: int -> int list list
```

that takes an integer `n` and returns a list of all partitions of `n`, where each partition is represented as a list of integers. Formally prove that `partitions n` contains only and exactly all the unique partitions of `n`.

Problem 6-2. Counting Partitions [50 points]

Now instead of computing the partitions of a number, we will simply count them.

(a) [25 points] One way to count the number of partitions is to compose the `length` function with the `partitions` function from Problem 6-1. Write a function `p_count_1 : int -> int` that does this.

The method is slow for large values of `n` because it explicitly generates all partitions. Write a recursive function `p_count_2 : int -> int` that counts the number of partitions of `n` without explicitly generating all the partitions.

Calculate the time taken by the two methods `p_count_1` and `p_count_2` for values of `n` ranging from `1` to `100` and plot a graph to compare them. You may use the `Sys.time` function for computing the time.

(b) [25 points] One can write an even faster function to count the number of partitions using the Pentagonal Number Theorem. The theorem states that:

$$p(n) = \sum_{k \neq 0} (-1)^{k-1} p\left(n - \frac{3k^2 - k}{2}\right) \tag{1}$$

where the summation is over all nonzero integers k (positive and negative), $p(n) = 0$ for $n < 0$, and $p(0) = 1$. Use this summation to implement a function `p_count_3` that computes the number of partitions of `n` efficiently. Compare the performance of `p_count_3` with `p_count_1` and `p_count_2` for `n` ranging from `1` to `100` and plot the results.

Problem 6-3. Rogers-Ramanujan Identities [25 points]

One of the Rogers-Ramanujan identities states that the number of partitions of n satisfying the following two specific conditions are equal in number:

- The adjacent parts differ by at least 2, and
- Each adjacent part is congruent to either 1 or 4 modulo 5

are equal in number. For example, consider the number 7. The partitions of 7 where the adjacent parts differ by at least 2 are $\{[2,5],[1,6],[7]\}$. The partitions of 7 where each part is congruent to 1 or 4 modulo 5 are $\{[1,1,1,1,1,1,1],[1,1,1,4],[1,6]\}$. Notice that both sets of partitions have the same number of elements (3 in this case), illustrating the identity.

(a) [18 points] Design and implement the following two functions:

```
is_diff_at_least_2 : int list -> bool

is_1_or_4_mod_5 : int list -> bool
```

These functions check if a given partition satisfies the corresponding property:

- `is_diff_at_least_2` should return `true` if the difference between adjacent parts in the partition is at least 2.
- `is_1_or_4_mod_5` should return `true` if every part in the partition is congruent to either 1 or 4 modulo 5.

(b) [7 points] Implement a function `check_rr_id: int -> bool` such that:

- Generates all partitions of a number `n` using the `partitions` function from Problem 1.
- Filters these partitions using the predicates `is_diff_at_least_2` and `is_1_or_4_mod_5`, and counts them.
- Confirms that the number of partitions that satisfy the two conditions is equal, thus checking that the Rogers-Ramanujan identity holds for the given number `n`.