Week 3: Lab 2

Lab 2

A warning about GenAI: Many of the problems given in the first few labs are classical problems with well known solutions. They are well known because they are useful learning problems. This means tools like ChatGPT and Copilot are almost certain to have been trained on similar problems. This means they are easily able to solve many of these problems. While this is useful sometimes, using these solutions will prevent you from learning what the labs are trying to teach you. Even worse, later labs will build on prior labs and increase in novelty and difficulty. Generative AI tools will not be able to solve them, and neither will you if you didn't understand the prior labs. We strongly suggest solving these problems yourself and writing your own code. You are free to use Gen AI, but you should consider it to be the similar to Googling the answer or asking your lab facilitator--only use it if you are stuck and make sure you understand the solution it produces.

In this lab we will be looking at the *Maximum Sum Subarray* problem. This was mentioned in the Intro lecture. Formally, consider an array A of numbers. We want to find the subarray whose sum is the largest possible amoung all subarrays of A. Define a subarray as a contiguous subsequence of elements. For example, if A=[3,2,4,1,-4,2], then this is a subarray [3,**2,4,1,-4**,2], and so is this [**3,2**,4,1,-4,2]. However, this [**3**,2,**4,1**,-4,2] is not a subarray, because it is not contiguous. In this example, the maximum sum subarray is [**3,2,4,1**,-4,2]. Note that we define the empty subarray as having a sum of zero, so the maximum sum subarray of A=[-2,-3,-1] is the empty subarray and it has sum 0.

Maximum Sum Subarray in O(N^2)

Please revise lecture 1, the intro lecture. In this lecture the maximum sum subarray problem is mentioned. First, an O(n^3) algorithm is introduced. Here is the code from the lecture.

```
def max_sum_subarray_naive(xs: list[int]) -> int:
    best = 0   # Can always take empty subarray
    # Try every candidate xs[lwr:upr]
    for lwr in range(len(xs)):
        for upr in range(lwr + 1, len(xs) + 1):
            # Add up the candidate
            total = 0
            for i in range(lwr, upr):
                total += xs[i]
            # Keep track of the best
```