1. Introduction

We are given a mathematical expression in **infix order** and our objective is to **calculate the gradient** in respective to variable in the expression. The gradient can be a combination of variables and literal values.

Auto gradient calculation is prevalent in Machine Learning cases. In packages like `PyTorch`, autograd() is used to back-propagate the gradients to optimize the weights and biases to minimize or maximize the objective function.

2. Algorithm Specification

We can handle this problem in a **Step-by-step operation**. First, we should **tokenize** the expression, separating variables, literal values and operators. After the tokenization, I constructed a **tokenList** to store the different elements with their types labeled. And then, I construct a **Expression Tree** with stacks to handle the problem of precedence of different calculations. **To be Continued...**

3. Testing Results

Table of test cases. Each test case usually consists of its purpose, the expected result, the program's actual behavior, the possible cause of a bug if your program does not function as expected, and the current status (*pass*, or *corrected*, or *pending*).

Table 1 shows some typical test cases for verifying the *Selection Sort* implementation and capturing potential bugs.

Table 1: Test cases for the *Selection Sort* implementation.

Test Cases	Design Purpose	result	status
[3]	Minimum array with a single element	[3]	*pass*
[1,2,4,5,9]	Array in ascending order	[1,2,4,5,9]	*pass*
[9,5,4,2,1]	Array in descending order	[1,2,4,5,9]	*pass*
[5,-9,2,-1,4]	Array in random order with negative values	[-9,-1,2,4,5]	*pass*
[1,1,1,1,1]	Array with repeated values	[1,1,1,1,1]	*pass*
...	...	...	...

Figures 1 shows the running time of the *Selection Sort* implementation. We

The running time of *Selection Sort*

Figure 1: The running time of *Selection Sort*

observe a quadratic-like curve from the figure, which implies an $O(n^2)$ algorithm (see analysis in the next section).

4. Analysis and Comments

Analysis of the time and space complexities of the algorithms. Comments on further possible improvements.

For the runtime of Selection Sort, there are two subtasks inside the *for*-loop. Specifically,

- finding the smallest integer between `list[i]` to `list[n-1]` takes $n - i$ time units;
- interchaning `list[i]` and `list[min]` takes a constant time c.

This is independent of any particular input. Therefore, both the average and worst time complexity can be computed as:

$$T(n) = \sum_{i=0}^{n-1} (n - i + c) = \Theta(n^2).$$

There is still considerable room for improvement. For example, we could adopt the divide-and-conquer strategy to achieve an $O(n \log n)$ algorithm.

For the space requirement, since we merely need an array to store the n integers, the space complexity is $\Theta(n)$ and should be optimal.

Appendix: Source Code (in C)

Please make sure that your code is sufficiently commented. Otherwise, it will *not* be evaluated.

Sample Code:

```c
/*
 * Find the smallest element in a range of an array
 * ------------------------------------------------
 *
 *   arr: an array of integers
 *   begin: the index to start the search
 *   end: the index to stop the search (non-inclusive)
 *
 *   returns: the index of the smallest element in the range
 */
int find_min(int arr[], int begin, int end) {
  int min_idx = begin;
  int min_val = arr[min_idx];

  for (int idx = min_idx+1; idx < end; ++idx) {
    if (arr[idx] < min_val) {
      min_idx = idx;
      min_val = arr[min_idx];
    }
  }

  return min_idx;
}

/*
 * Swap the values of two integers
 * -------------------------------
 *
 *   a: pointer to the 1st integer
 *   b: pointer to the 2nd integer
 *
 *   returns: none
 */
void swap(int *a, int *b) {
  int tmp = *a;
  *a = *b;
  *b = tmp;
}

/*
 * Rearrange an array of integers into increasing order in-place
 * -------------------------------------------------------------
 *
 *   arr: an unsorted array of integers
 *   n: number of elements in the array
 *
 *   returns: none
 */
void sort(int arr[], int n) {
  for (int i = 0; i < n; ++i) {
    int min_idx = find_min(arr, i, n);
    /*after the swap, arr[i] is the i-th smallest integer*/
    swap(&arr[i], &arr[min_idx]);
  }
}
```

Note: Feel free to use your own style of code or comments. Just make sure to be consistent with yourself.

Declaration

I hereby declare that all the work done in this project is of my independent effort.