INDIAN INSTITUTE OF TECHNOLOGY TIRUPATI
DEPARTMENT OF MATHEMATICS AND STATISTICS

MA635P-Scientific Programming Laboratory

Lab Exercise-1 (150 Marks) Deadline: 9 January 2025, 5:00 PM

Preliminaries

Definition 1 (Algorithm)**.** *An algorithm is a list of unambiguous rules that specify successive steps to solve a problem*

Definition 2 (Computer Program)**.** *The computer program is clearly specified sequence of computer instructions implementing algorithm*

Definition 3 (Elementary Operations)**.** *Most modern computers and languages build complex programs from ordinary arithmetic and logical operations such as standard unary and binary operations (negation, addition, subtraction, multiplication, division, modulo operations, assignment), boolean operations, binary comparisons ($=, >, <, \geq, \leq$), branching operations. These operations are called as elementary operations.*

Definition 4 (Running Time)**.** *The running time or computing time of an algorithm is the number of its elementary operations. It is denoted by $T(n)$.*

Example 1.1 (Sum of Elements of an array)**.** Let a denote an array or list of integers where the sum

$$s = \sum_{i=0}^{n-1} a[i]$$

is required. To get the sum s, we need to repeat n times the same elementary operations. Therefore, the running time $T(n)$ is proportional to or linear in n. That is $T(n) = cn$. This algorithm is called linear algorithm. The unknown factor c depends on a particular computer, programming language, compiler, OS etc.

In the above algorithm, suppose $T(1)$ is given to you, then you can compute $T(1000) = 1000T(1) = 10T(100)$. If per addition, it takes 1s, then $T(1) = 1$, then $T(1000) = 1000s$.

Example 1.2 (Sum of Elements of Subarrays)**.** Now, let us compute the sum of each subarray of some m. That is,

$$s_j = \sum_{k=0}^{m-1} a[j+k], j = 0, 1, 2, \cdots, n-m$$

How many subarrays? Prove that $T_2(n) = cm(n-m+1)$. Also, if If $m = \frac{n}{2}$, prove that $T_2(n) = 0.25cn^2 + 0.5cn = O(n^2)$.

Exercise 1: Linear Sum, Quadratic: Slow/Fast Sum

Algorithm 1: Linear Sum

Input: array, $a[0, 1, \cdots, n-1]$

Output: s

```
1 s ← 0
2 for i ← 0 to n − 1 do
3     s ← s + a[i]
```

Algorithm 2: Quadratic Algorithms: Slow Sum

Input: array, $a[0, 1, \cdots, 2m-1]$

Output: $s[0, 1, \cdots, m]$

```
1 s ← 0
2 for i ← 0 to m do
3     s[i] ← 0
4     for j ← 0 to m − 1 do
5         s[i] ← s[i] + a[i + j]
```

Algorithm 3: Quadratic algorithms: Fast Sum

Input: array, $a[0, 1, \cdots, 2m-1]$

Output: $s[0, 1, \cdots, m]$

```
1 s[0] ← 0
2 for j ← 0 to m − 1 do
3     s[0] ← s[0] + a[j]
4 for i ← 1 to m do
5     s[i] ← s[i − 1] + a[i + m − 1] − a[i − 1]
```

1. Write a Python/C++ function to compute the time taken to run a program in terms of minutes and hours (Internet usage allowed)

2. Extend this function to display the time in terms of hours or minutes or seconds or microseconds or milliseconds or nanoseconds or days or weeks or years depending the number (Internet usage allowed). Save this function for future exercises. We will use this function to get time taken by each program in the future (Internet usage allowed).

3. Write a Python/C++ program to implement algorithm 1, 2 and 3, generate random entries for given n as in Table 1. Let $T_1(n), T_2(n)$ and $T_3(n)$ be the running time for algorithm 1, 2 and 3 respectively. For T_2 and T_3, take, $m = n/2$.

4. For each elementary operations (in this case, additions/subtractions only), compute respective $T_1(n), T_2(n)$ and $T_3(n)$. Fill the table of minutes and hours.

5. Estimate the value of c from the table

6. Compute the $T_1(10^{12})$, $T_2(10^{12})$ and $T_3(10^{12})$ without doing the array sum operation.

$[3 + 3 + 3 \times 3 + 9 \times 5 + 4 + 6 = 70]$

n	$T_1(n)$	**Minutes**	**Hours**	$T_2(n)$	**Minutes**	**Hours**	$T_3(n)$	**Minutes**	**Hours**
100									
500									
1000									
5000									
50000									

Tab. 1: $T_1(n), T_2(n), T_3(n)$

Exercise 2 : Polynomials

Example 3.1 (Polynomials)**.** Let a denote an array or list of real number and

$$s(x) = \sum_{i=0}^{n-1} a_i x^i$$

is required. Let $T_4(n)$ be the running time to compute $s(x)$.

1. Write a Python/C++ program to compute $s(x)$ for given x and n
2. Generate the list of real numbers for a and choose $x = 0.1$ and fill the below table
3. Compute the $T_4(10^{12})$ without doing the array sum operation.

$[3 + 5 \times 3 + 2 = 20]$

n	$T_4(n)$	**Minutes**	**Hours**
5			
10			
20			
25			
50			

Tab. 2: $T_4(n)$

Exercise 3 : Taylor Series

Example 4.1 (Taylor Series)**.**

$$f(x) = \sum_{i=0}^{\infty} (x - x_0)^i \frac{f^{(i)}(x_0)}{i!}$$

- You can relate that this problem is similar to the above problem, where $a[i]$ is given by some mathematical expression. In order to compute the infinite sum, you require enormous amount of sum and enormous amount of accuracy. Let us not do infinite sum, instead, let us do a finite sum, for sufficiently large n, say $f_n(x)$.

Example 4.2 (Taylor Series - Truncated)**.**

$$f_n(x) = \sum_{i=0}^{n-1} (x - x_0)^i \frac{f^{(i)}(x_0)}{i!}$$

Example 4.3 (Taylor Series - $\sin x$ Truncated)**.**

$$S_1(x) = \sin x = \sum_{i=0}^{n-1} (-1)^i \frac{x^{2i+1}}{(2i+1)!}$$

1. Write a Python/C++ function create a factorial of any natural number. That is, $i!$
2. Write a Python/C++ function create a exponential power of any real number (x) for a given i, that is, x^i.
3. For given n and x, use the above two functions to compute the truncated sine series.
4. Compute the value of $\sin x$ using S_1 and numpy library and compare it
5. Let $T_5(n, x)$ be running time for $S_1(x)$. Fill out the following table

x	n	$S_1(x)$	np.sin(x)	$T_5(n, x)$	Seconds
$\pi/2$	20				
$\pi/4$	20				
$\pi/6$	20				
$\pi/3$	20				
$\pi/3$	50				

Tab. 3: $\sin(x)$

$[3 + 3 + 3 + 3 + 7 \times 4 = 40]$

Exercise 4 : Norm of a Vector

Let

$$x = (x_1, x_2, \cdots, x_n)$$

be a vector in $\mathbb{R}^n$.

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2}$$

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|$$

The $p-$ norm or ℓ_p norm of a vector is defined by

$$\|\mathbf{x}\|_p^p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

1. Compute each of the norms using Python/C++ function. Get the input n from the user, generate a random vector between 0 and 1 of size n. Display the norm for all cases.
2. Use $p = 3, p = 5$ and $p = 1/2$ for testing. Compute the time requirement $\|x\|_1, \|x\|_2^2, \|x\|_p^p$ when $n = 10^{12}$.

$[4 \times 5 = 20]$

Bonus : Taylor Series

1. Repeat the above exercise for the following series

$$S_2(x) = \cos x = \sum_{i=0}^{n-1} (-1)^i \frac{x^{2i}}{(2i)!}$$

$$S_3(x) = e^x = \sum_{i=0}^{n-1} \frac{x^i}{i!}$$

$$S_4(x) = \tan^{-1} x = \sum_{i=0}^{n-1} (-1)^{i-1} \frac{x^{2i-1}}{2i-1}$$

1. Compute the value of $\cos x$, e^x and $\tan^{-1}(x)$ using S_2, S_3 and S_4. Also use numpy library and compare it
2. Fill out tables 4, 5 and 6.
3. Use $\tan^{-1}(x)$ to compute the value of π. Hint: $4 \times \tan^{-1}(1)$.

$[2 + 7 \times 4 + 2 + 7 \times 4 + 2 + 6 \times 4 + 4 = 90]$

x	n	$S_2(x)$	np.cos(x)	$T_6(n,x)$	Seconds
$\pi/3$	20				
$\pi/4$	20				
$\pi/6$	20				
$\pi/2$	20				

Tab. 4: $\cos(x)$

x	n	$S_3(x)$	np.exp(x)	$T_7(n,x)$	Seconds
0	20				
1	20				
-1	20				
0.1	20				

Tab. 5: $\exp(x)$

x	n	$S_4(x)$	np.atan(x)	$T_8(n,x)$	Seconds
3	20				
30	20				
500	20				
0.5	20				

Tab. 6: $\tan^{-1}(x)$

Bonus : FLOPS for T_1 to T_8 (Paper Work)

1. Convert the table entries to years. How many years will take for $T_i(n), i = 1, 2, \cdots, 8$ if $n = 10^6$? $[1 + 1 = 2]$

2. Suppose your computer can perform 10^3 [kFLOPS] operations (additions/subtraction) in 1s, how many years will it take for finding the sum using algorithm 2 and algorithm 3, when $n = 10^9$. $[1 + 1 = 2]$

3. Answer the above question 2, for 10^6 operations [MFLOPS], 10^9 operations [GFLOPS] and 10^{12} operations [TFLOPS]. $[2 + 2 + 2 = 6]$