

# Assignment 2

---

## Github Link

- **HW02:** <https://github.com/hemanth-nakshatri/repo759/tree/main/HW02>
- **Parent Repo:** <https://github.com/hemanth-nakshatri/repo759>

## Question 1

A. `scan.h` and `scan.cpp` files in the `HW02` directory.

B. `task1.cpp` creates a random array of size `n` between `-1` and `1`. It calls `scan` function and times the execution time.

C. `task1.pdf` contains the graph of the execution time of the `scan` function for different values of `n`. The scales are logarithmic to help understand the result easily.

### ChatGPT generated graph:

- **Prompt:** Write a python code to plot a graph of execution times stored in a file called `scaling_results.txt`. The values in the file is the y-axis and the values from  $2^{10}$  to  $2^{30}$  are the x-axis.
- **GPT Provided code:**

```
import numpy as np
import matplotlib.pyplot as plt

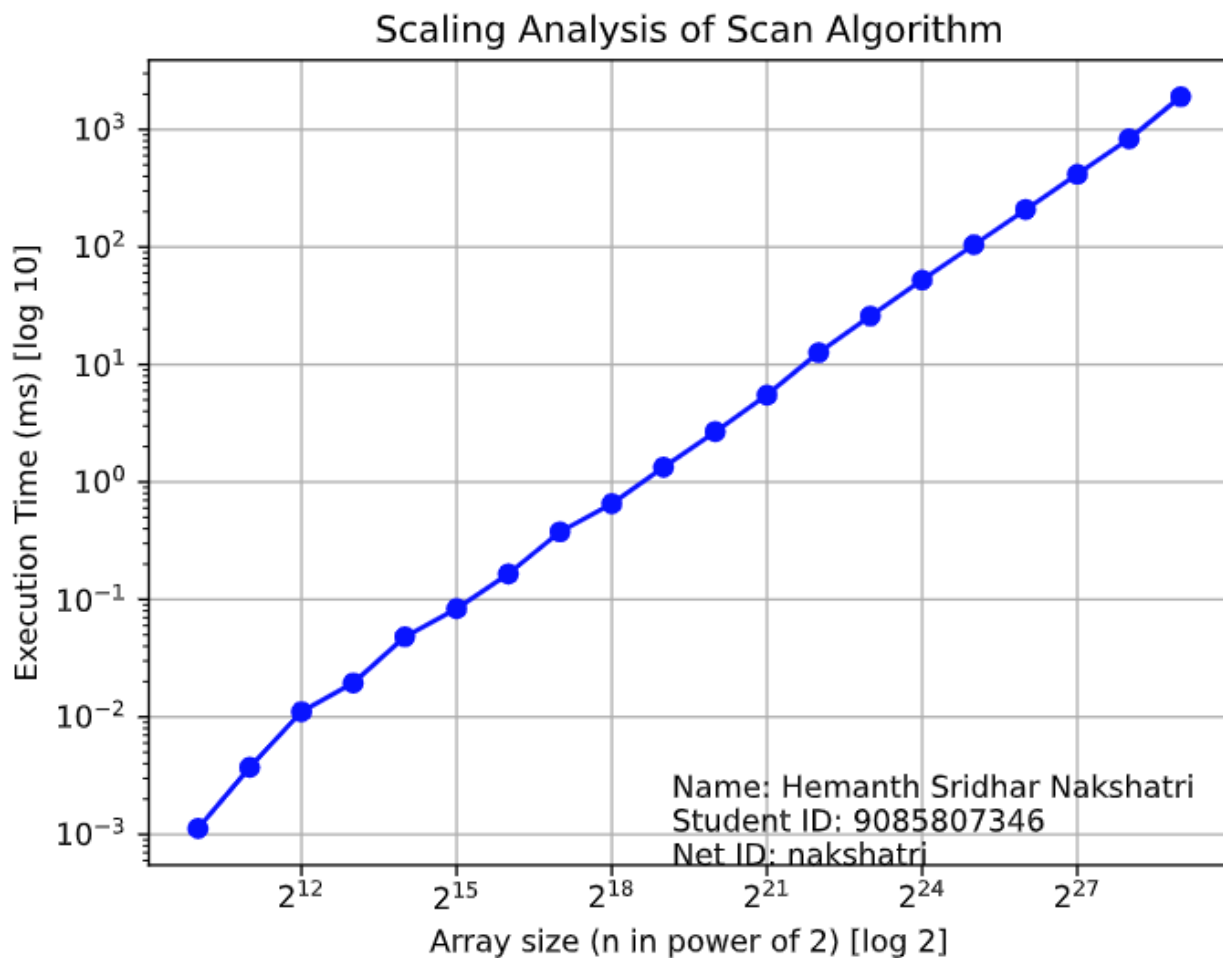
# Step 1: Read the execution times from the file
with open('scaling_results.txt', 'r') as file:
    y_values = [float(line.strip()) for line in file.readlines()]

# Step 2: Generate x-axis values ( $2^{10}$  to  $2^{30}$ )
x_values = [2**i for i in range(10, 31)]

# Step 3: Plot the graph
plt.figure(figsize=(10, 6))
plt.plot(x_values, y_values, marker='o')

# Step 4: Add labels and title
plt.xlabel('Input Size ( $2^n$ )')
plt.ylabel('Execution Time (seconds)')
plt.title('Execution Time vs Input Size ( $2^{10}$  to  $2^{30}$ )')
plt.xscale('log', base=2) # Use logarithmic scale for x-axis
plt.grid(True)

# Step 5: Show the plot
plt.show()
```



## Question 2

- Check the Github repository for the code.

## Question 3

- It can be seen that different mmul functions have different execution times.
- **mmul1:** (i,j,k) iteration order. Thus summed along k.
  - A has spatial locality. B has none and C has temporal locality. Uses cache but not optimal in my opinion.
- **mmul2:** (i,k,j) iteration order. Thus summed along j.
  - A has spatial locality. B and C have temporal locality. Uses cache optimally as compared to other functions.
- **mmul3:** (j,i,k) iteration order. Thus summed along i.
  - A and C have no locality. B has temporal activity. Not optimal again.
- **mmul4:** Approach is similar to mmul1 but with <vector> operations.
  - Has similar run time.

- But it might have a bit of overhead due to vector memory allocations.

- **Execution times for  $n = 1024$ :**

Method	Time (ms)
mmul1	6037
mmul2	991
mmul3	10683
mmul4	6549

***Approach for MMUL problem was partly discussed with a friend of mine who isn't enrolled in the course.***