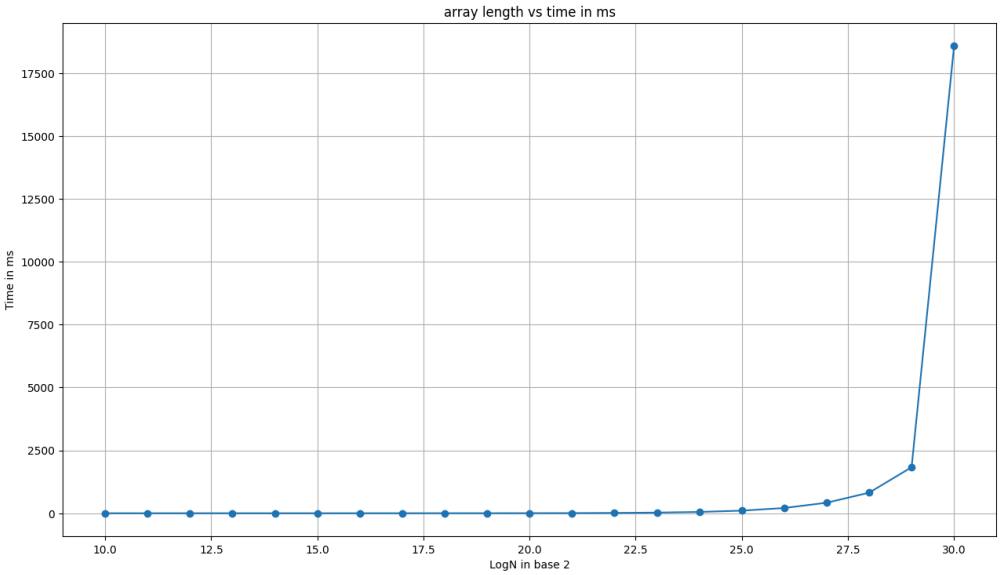
HW02

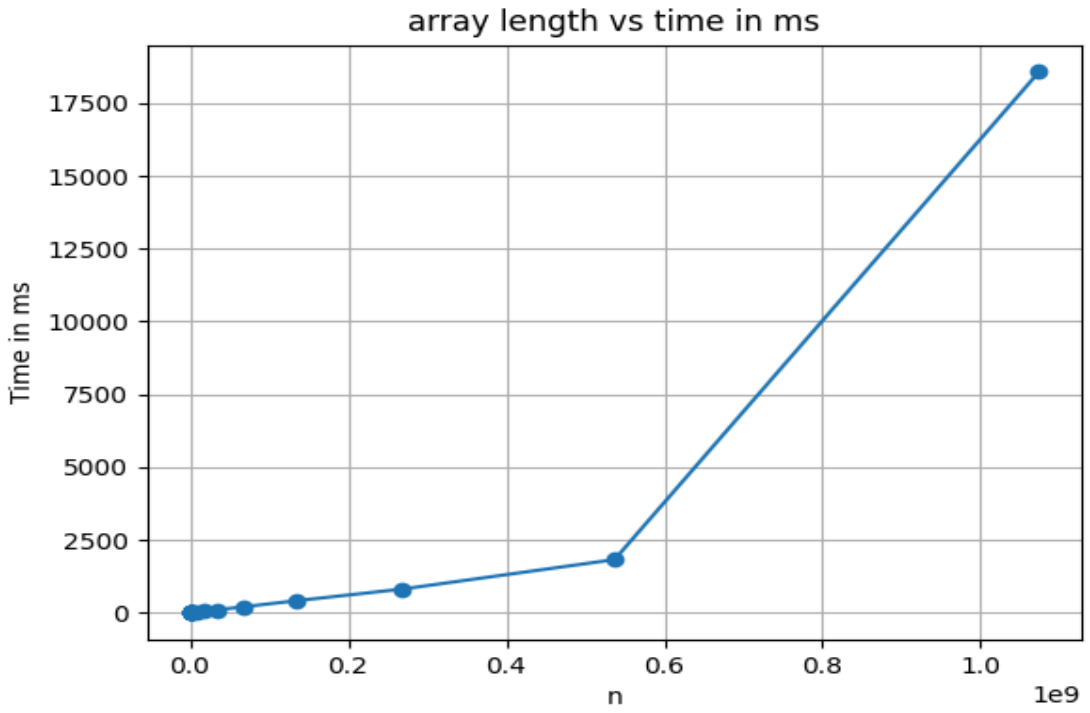
Git-link:
https://github.com/surendra-UW/repo759/tree/main/HW02

1.

a. Code in the repo
b. Code in the repo
   Sample output:
   0.028708
   0.531451
   -107.667



c.
   Code submitted in the repo
   X-axis LogN_2



2. Code submitted in git
Output:
0.00135
-8.25175
-6.11961

3. a-d code submitted in git
e. Output:
1000
1634.82
-4.69027
734.832
-4.69027
8243.46
-4.69027
1623.76
-4.69027
f. Explanation:
Matmul1: From the code we can understand that every multiplication in the inner loop the matrix B has to skip n locations to fetch the next number from B matrix while C and A has space locality. So, it gives the second best result of 1634.82ms.

```
    for(unsigned int i=0;i<n;i++) {
        for(unsigned int j=0;j<n;j++) {
            for (unsigned int k=0;k<n;k++){
                C[i*n+j] += A[i*n+k]*B[k*n+j];
            }
        }
```

```
    }
```

Matmul2: Compared with matmul1 we have better spatial access of all three matrices A,B and C as both C and B matrices are accessing consecutive elements and A matrix the same element for the innermost loop.

```
for(unsigned int i=0;i<n;i++) {
        for (unsigned int k=0;k<n;k++){
            for(unsigned int j=0;j<n;j++) {
                C[i*n+j] += A[i*n+k]*B[k*n+j];
            }
        }
    }
```

Matmul3: In this code both C and A matrices jump n elements for each access which explains the reason for the highest latency of 8243ms.

```
for(unsigned int j=0;j<n;j++) {
        for (unsigned int k=0;k<n;k++){
            for(unsigned int i=0;i<n;i++) {
                C[i*n+j] += A[i*n+k]*B[k*n+j];
            }
        }
    }
```

Matmul4:
Using vectors produced almost the same result as matmul1. There is a small difference of 10ms which might be due to the fact that A and B matrices are cached because of other 3 multiplications. This can be proved by switching the matmul4 code to the first operation followed by matmul1 which gives matmul1 as 25ms better than matmul4.