

Code for 1.24 part:


```
#include <stdio.h>
#include<stdlib.h>
#include <time.h> //for finding time
//code for transposing the matrix
void mult( int n, long int a[][n], long int b[][n],long int c[][n]){
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            c[i][j]=0;
            for(int k=1;k<=n;k++)
                c[i][j] +=a[i][k]*b[k][j];
        }
    }
}

int main()
{
    int n=1;
    int loop=0;
    double tim1[10], tim2[10];
    while (loop++ < 10) {
        long int a[n][n], b[n][n], c[n][n];
        for (int i = 0; i < n; i++) {
            for(int j=0;j<n;j++){
                long int no = rand() % n + 2;
                a[i][j] = no;
                b[i][j] = no;}}
        clock_t start, end;
        start = clock();//denoting the value of initial clock time
        mult(n,a,b,c);
        end = clock();//denoting the value of final clock time

        tim1[loop] = ((double)(end - start));
        printf("%d,%li \n",n,
            (long int)tim1[loop]);
        n=n+10;
    }

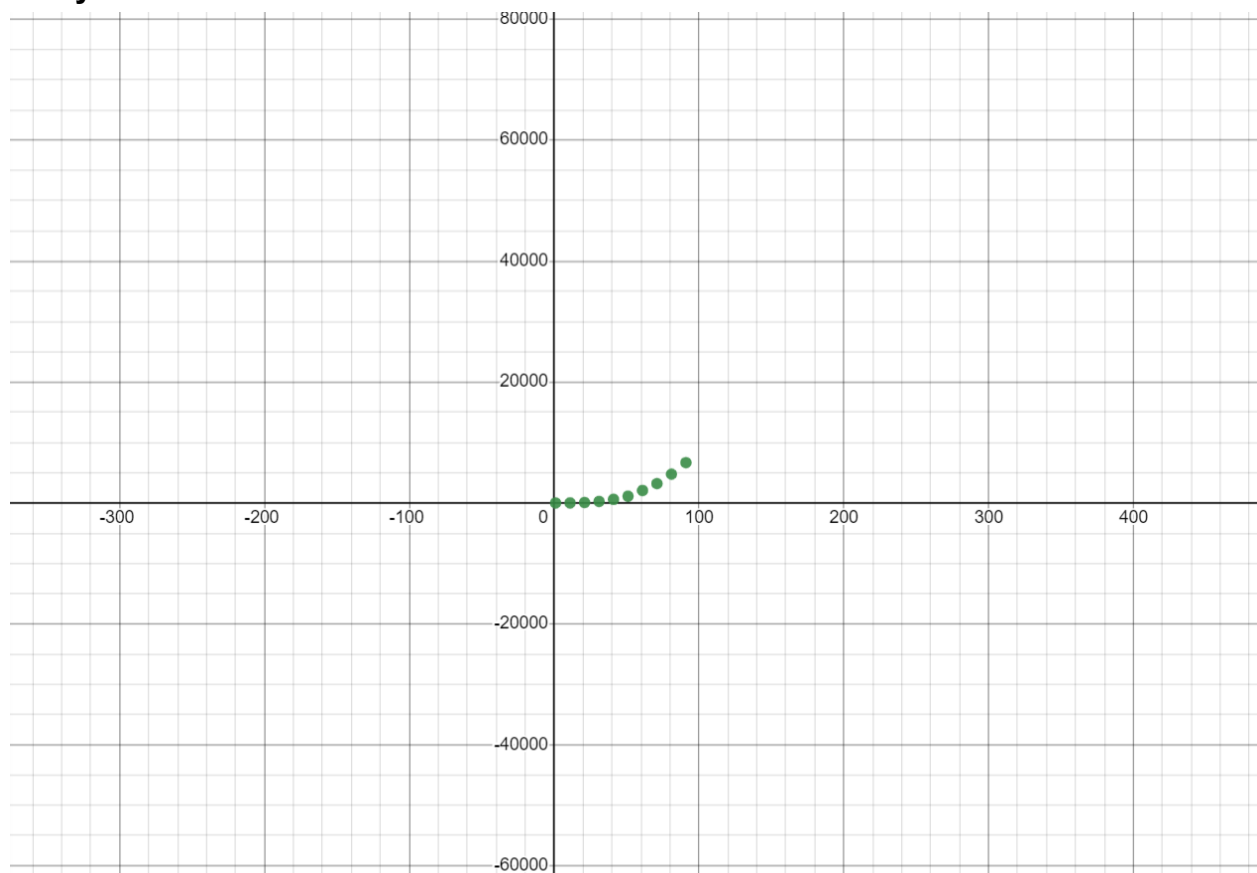
    return 0;
}
```

Results:

1	x_1	 y_1	
	1	2	
	11	12	
	21	78	
	31	258	
	41	597	
	51	1121	
	61	2064	
	71	3205	
	81	4769	
	91	6671	

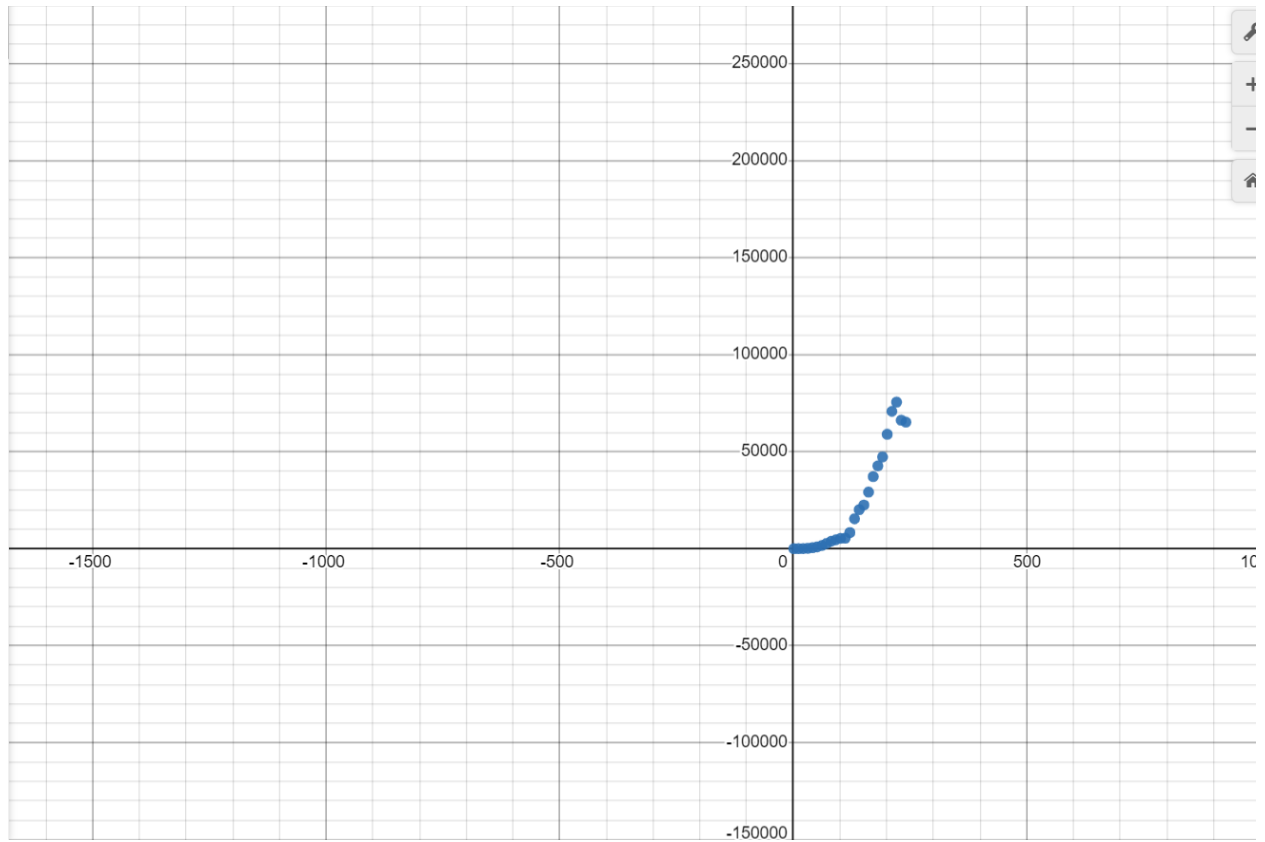
These results are obtained when I have changed the value of n by incrementing by 10 and for 10 iterations

Analysis:




This is the graph obtained for the initial part of the code, it resembles with the n^3 curve. Since for transposing, we have to do the iteration for loop for the 3 times and varying the value of n . Hence, time complexity value will depend on n for 3 times which gives n^3 .

The curve will be close to n^3 as the value of n increases. This is observed in curve below:

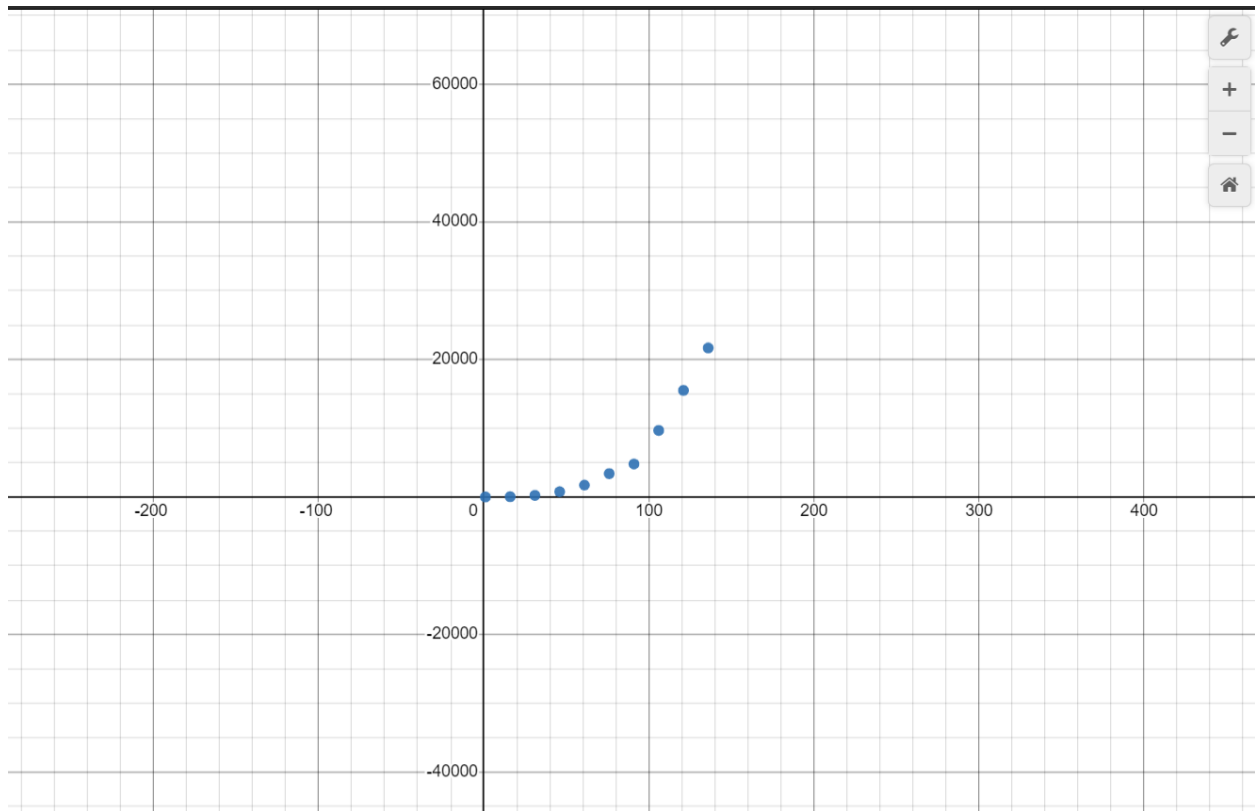


From the curve we can see that the curve will get close to n^3 as the value of n increases

x_3	 y_3
1	2
16	36
31	236
46	769
61	1731
76	3399
91	4793
106	9673
121	15496
136	21669

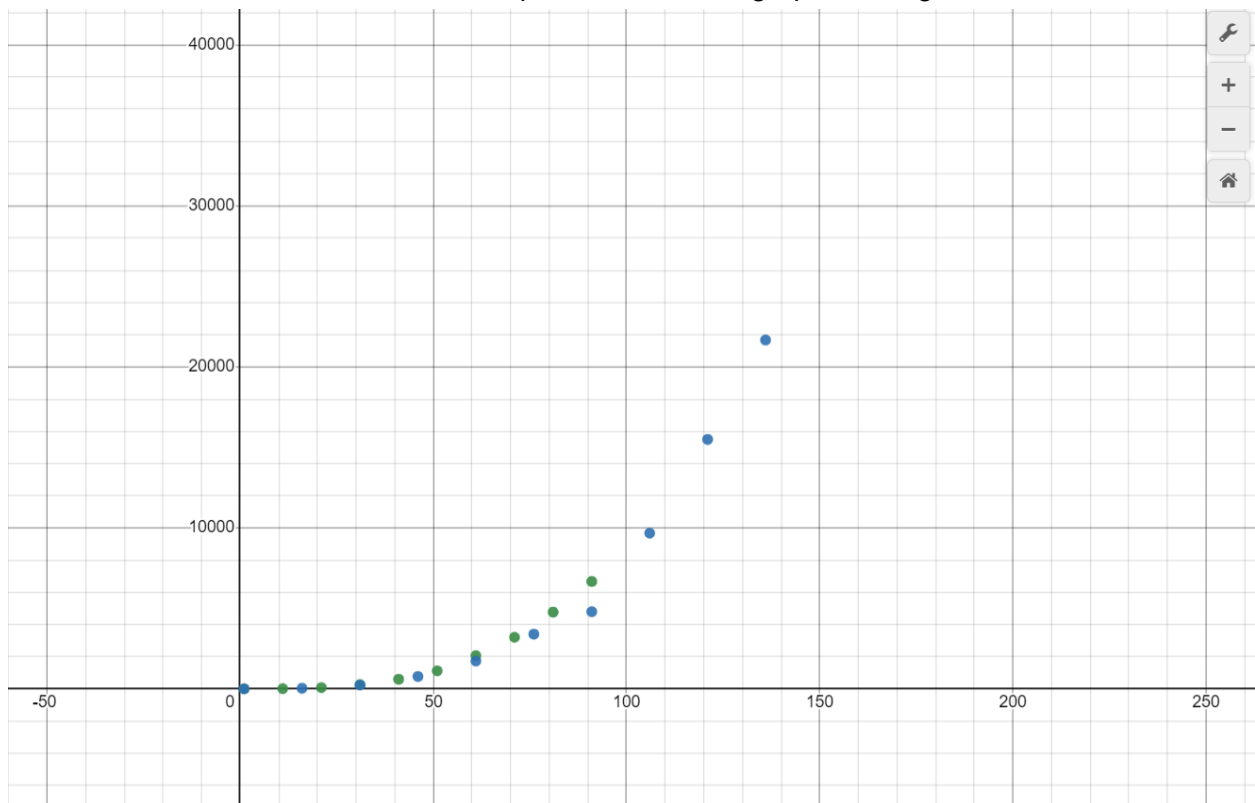


The increment value is altered from 10→15 the observations are noted hereby



The results from the above observation are noted in the graph above.

The two cases with different curves are plotted in a same graph which gives results as:



Both curves will have different values at different instants but when the expected curve is estimated using limits, it will give the same curve of form n^3 .

Code for 1.25:

```
#include <stdio.h>
#include<stdlib.h>
#include <time.h>
int m=2;
int p=2;
void mult(int n, long int a[2][n], long int b[n][2],long int c[2][2]){
    for(int i=1;i<=2;i++){
        for(int j=1;j<=2;j++){
            c[i][j]=0;
            for(int k=1;k<=n;k++)
                c[i][j] +=a[i][k]*b[k][j];
        }
    }
}


int main()
{
    int n=1;
    int loop=0;
    double tim1[10];
    while (loop++ < 100) {
        long int a[m][n], b[n][p], c[m][p];
        for (int i = 0; i < m; i++) {
            for(int j=0;j<n;j++){
                long int no = rand() % n + 1;
                a[i][j] = no;}}
        for (int i = 0; i < n; i++) {
            for(int j=0;j<p;j++){
                long int no = rand() % n + 2;
                b[i][j] = no;}}
        clock_t start, end;
        start = clock();
        mult(n,a,b,c);
        end = clock();

        tim1[loop] = ((double)(end - start));
        printf("%d,%li \n",n,
            (long int)tim1[loop]);
        n=n+10;
    }
```

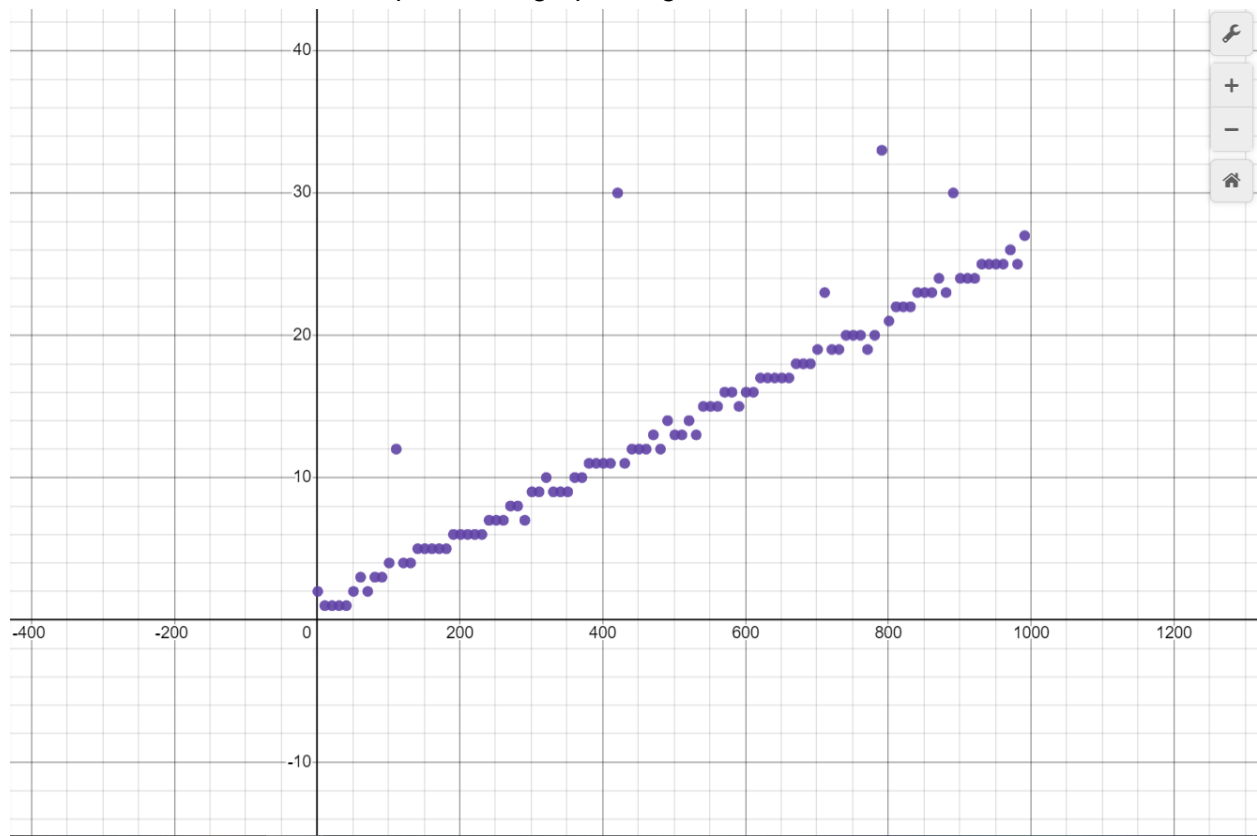
```
}  
  
return 0;  
}
```

In the question it has been mentioned that we have to obtain time complexity values for varying 'n'. Hence, I have equated values of m, p to constant or else the time complexity will also depend on m, p and final time complexity will tend to $O(n*m*p)$. Since we are only changing with respect to 'n', time complexity will come as $O(n)$

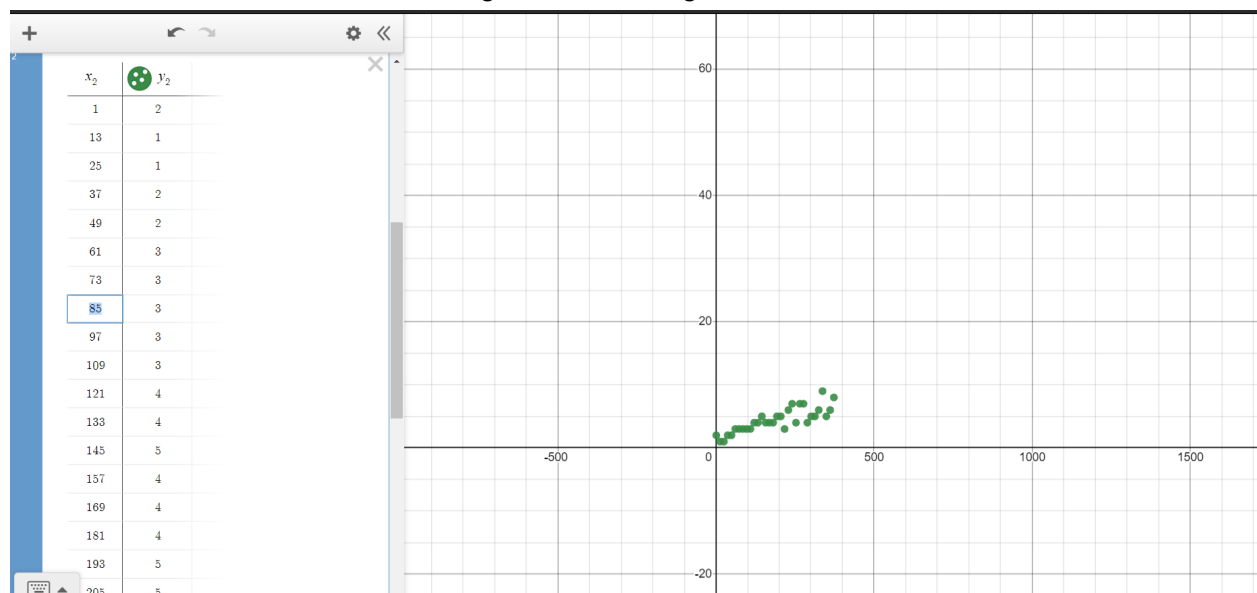
Results:

1	x_1	 y_1	
	1	2	
	11	1	
	21	1	
	31	1	
	41	1	
	51	2	
	61	3	
	71	2	
	81	3	
	91	3	
	101	4	
	111	12	
	121	4	
	131	4	
	141	5	
	151	5	
	161	5	
	171	5	

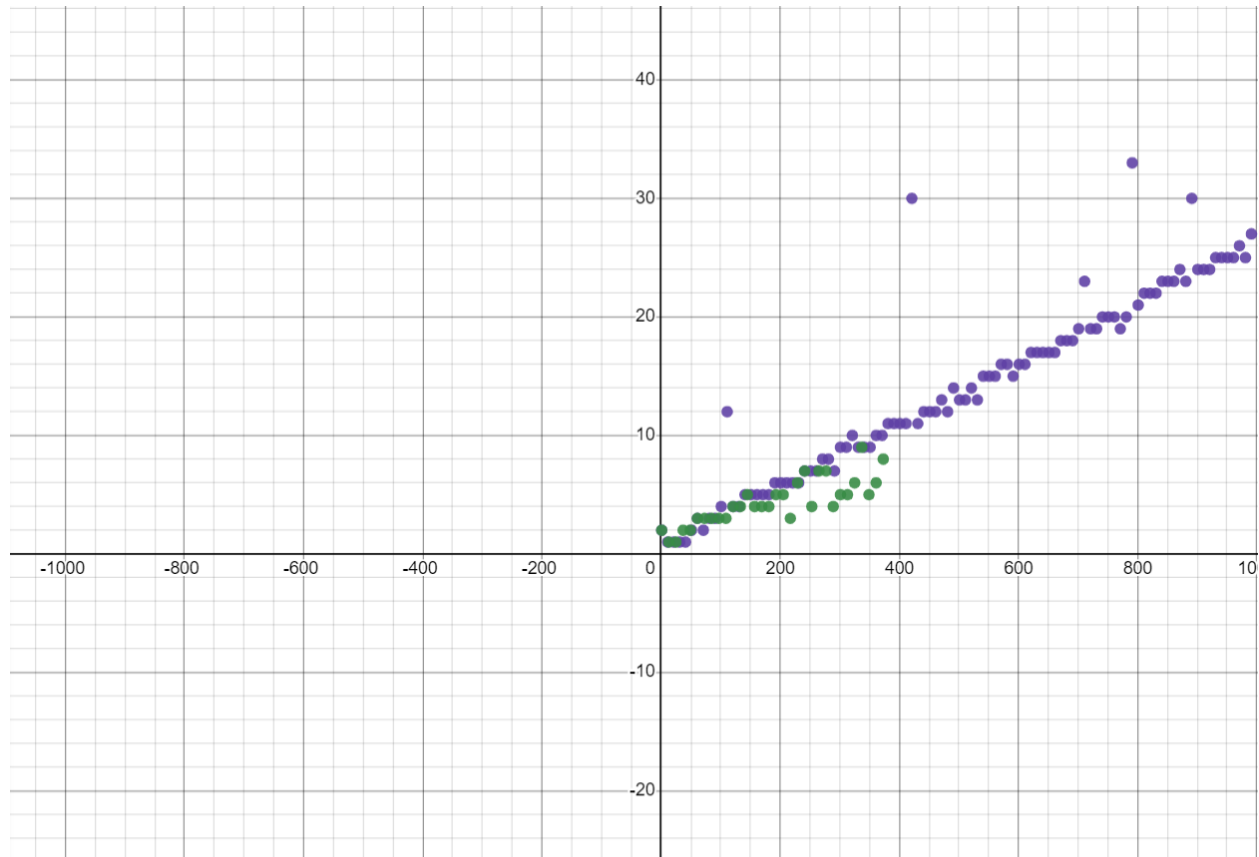
When these observations are plotted on graph,we get



We can observe that the curve will get close to linear graph as the time complexity tends to $O(n)$
When the value of increment is changed to 12,it will get as



The variation in the curve has been increased with increased increment. This is due to output matrix multiplication of same size but different instructions hence, varied n will result in more variance. We can observe this in comparing both curves:



We can see that, the final distribution of curves are tending to linear distribution but with increased variance with varied increments.

Conclusion:

1. For the matrix transpose application in code 1.24 with varying n , we get $O(n^3)$ as time complexity. The values are tending to curve of n^3 with increased observations.
2. For the matrix multiplication application in the code 1.25 with varying n and constant m, p for observing change in time complexity with respect to n , we get time complexity as $O(n)$ since m, p are constant it will only be proportional to linear curve. The values will tend to linear curve n with increased observations.