ASSIGNMENT 4
SECTION 1

2020102043
VENKATA RAMANA M N

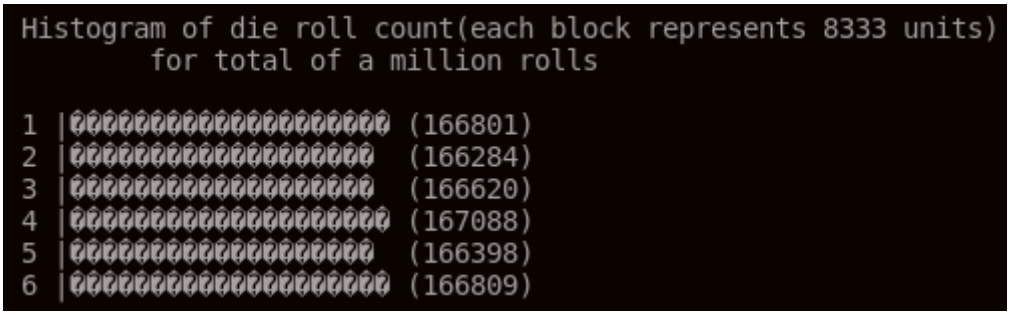
Q1

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define NUM_DIE 6
#define BLOCK_SIZE 8333

int main(){
    srand((unsigned)time(NULL));
    int die = rand() % NUM_DIE + 1, die_count[6] = {0};

    for (int i = 0; i < (int)1e6; i++){
        int die = rand() % NUM_DIE + 1;
        die_count[die - 1]++;
    }

    int count[6] = {0};
    printf ("\n");

    printf ("Histogram of die roll count(each block represents 8333 units)\n");
    printf ("\tfor total of a million rolls\n\n");
    for (int i = 0; i < 6; i++){
        count[i] = die_count[i]/BLOCK_SIZE;
        printf ("%d |", i + 1);
        for (int j = 0; j < count[i]; j++){
            printf ("%c", (char)254u);
        }
        printf ("\t(%d)\n", die_count[i]);
    }
    printf ("\n");
}
```

The program throws a die million times and plots count of die directly in the output of the program. No plotting tool was used. Just compile and run the program to see the graph for yourself :)

char(254u) is a filled square and is used here for plotting. When run in other systems, might show question mark instead of filled square.

E Here is the output for one trial of million throws:



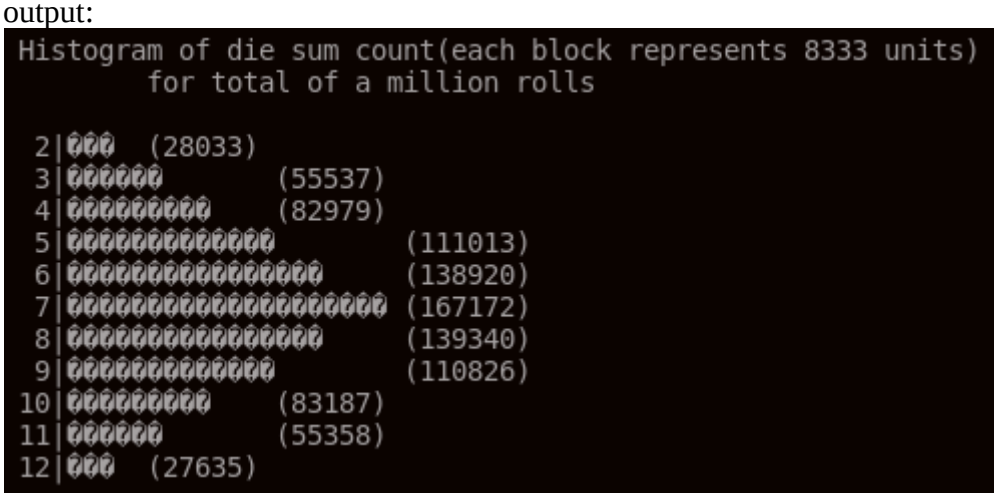
The above graph matches our expectations since the occurrence of each value of dice is uniform and equal to 1/6. The above graph is almost a uniform distribution.

Q2

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define NUM_DIE 6
#define BLOCK_SIZE 8333

int main(){
    srand((unsigned)time(NULL));
    int die_count[11] = {0};

    for (int i = 0; i < (int)1e6; i++){
        int die_1 = rand() % NUM_DIE + 1;
        int die_2 = rand() % NUM_DIE + 1;
        die_count [die_1 + die_2 - 2]++;
    }

    int count[11] = {0};
    printf ("\n");

    printf ("Histogram of die sum count(each block represents 8333 units)\n");
    printf ("\tfor total of a million rolls\n\n");
    for (int i = 0; i < 11; i++){
        count[i] = die_count[i]/BLOCK_SIZE;
        printf ("%2d\2|", i + 2);
        for (int j = 0; j < count[i]; j++){
            printf ("%c", (char)254u);
        }
        printf ("\t(%d)\n", die_count[i]);
    }
    printf ("\n");
}
```

Here the program calculates the sum of two dice for a trial of million times.

Here is the output:



The graph resembles the values expected from theory.
This is because sum is 2 if the values on dice are (1,1). And the sum is 12 if values on dice are (6, 6). Whereas the sum is 7 if values on dice are (1, 6), (6, 1), (2, 5), (5, 2), (3, 4), (4, 3).

The probabilities of each element are:

2 – 1 /36
3 – 2 /36
4 – 3 /36
5 – 4 /36
6 – 5 /36
7 – 6 /36
8 – 5 /36
9 – 4 /36
10 – 3 /36
11 – 2 /36
12 – 1 /36

Q3

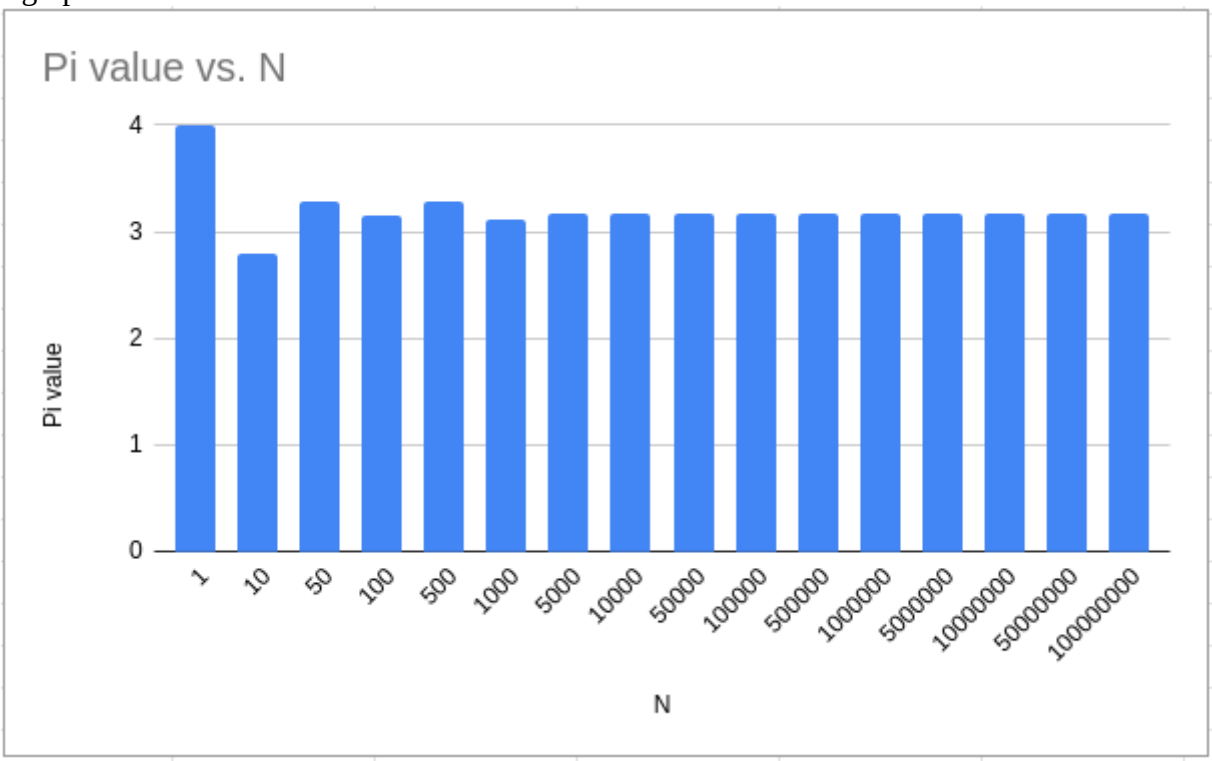
```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(){

    long long N;
    scanf("%lld",&N);
    long long DIV = 1e8;
    double x, y, d, pi;     //x, y coordinates, distance from centre, pi value
    int p = 0, q = 0;           //points inside circle, square

    srand((unsigned) time(NULL));        //seed random numbers

    for (int i = 0; i < N; i++){
        x = (double)((rand() % (2*DIV)) - DIV)/ DIV;       //gives a random point between -1 and 1
        y = (double)((rand() % (2*DIV))- DIV)/ DIV;       //gives a random point between -1 and 1

        if((x*x + y*y) < 1) p++;

        q++;
    }

    pi = (double) 4 * p / q;

    printf("%lf\n", pi);
}
```

In the above code, we seed random numbers and take x and y values representing coordinates. X and y are random values between -1 and 1. N random coordinates are taken and tested if they fall in the circle. This probability is finally equated to Pi/4 and value of pi is found consequently.

Values of pi found from above code:

N	Pi value
1	4
10	2.8
50	3.28
100	3.16
500	3.28
1000	3.124
5000	3.168
10000	3.1724
50000	3.16696
100000	3.18016
500000	3.170264
1000000	3.176452
5000000	3.173644
10000000	3.172405
50000000	3.172211
100000000	3.172479

In a graph:



We see here that after 5000, the value remains fairly the same and thus it requires about ~5000 points for a good value of pi. Later the accuracy keeps increasing.