**PROBLEM 1**

**a)**

[ .3(100) + .1(300) + .1(600) + .5(160)] [10]-9 = 200 [10]-9 Seconds

→ 109 / 200 (operations / second)

→ 103 / 200 MIPS

= 5 MIPS

**b)**

[ .3(50) + .1(240) + .1(480) + .5(128)] [10]-9 = 151 [10]-9 Seconds

→ 109 / 151 (operations / second)

→ 103 / 151 MIPS

= 6.62 MIPS

**c)**

[ .3(100) + .1(100) + .1(300) + .5(160)] [10]-9 = 140 [10]-9 Seconds

→ 109 / 140 (operations / second)

→ 103 / 140 MIPS

= 7.14 MIPS

Buying the coprocessor would be a larger performance increase than the memory. If I could and had to choose only one for free that would be it.

Realistically, I would compare the prices, weigh the benefits, and see if I could afford both or either of the two.

**PROBLEM 2**

**Cylinders = 900**

**Capacity = 300MB**

**RPM = 7000**

**t(x) = x1/2**

**a)**

Trd = 30 / 7000

Tseek = x1/2

Cylinders used by file = 900/(300/100) = 300

Rotational delay = 1/ [7200 RPM / (60/2 seconds)] = 1/[240] seconds = 1000/[240] ms

Access time = 3001/2 + 1000/240 = 21.49 ms

**b)**

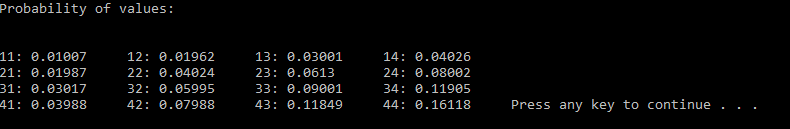
**c)**

Tcache = 1ms

Scache = 50MB

Access time = 1 + (100-50)/100 \* [ 3001/2 + 1000/240 ]= 11.74ms

**PROBLEM 3**



**N = 100,000**

#include <math.h>

#include <cstdlib>

#include <iostream>

#include <string>

using namespace std;

static double result3[45] = { 0 }; //array to store results

//generate number between 1 and 4 with given distributions

static int generateSingleNumber3()

{

double v = ( (double)rand() / (double)RAND\_MAX);

if (v < .1) return 1;

else if (v < .3) return 2;

else if (v < .6) return 3;

else return 4;

}

//generate the two digit number

static int generateNumber3()

{

return generateSingleNumber3() + 10 \* generateSingleNumber3();

}

//fills result array with probability

static void results3()

{

const int N = 10000;

for (int i = 0; i < N; i++) { result3[generateNumber3()] ++; }

for (int i = 0; i < 45; i++) { result3[i] = result3[i] / (double)N; }

}

//outputs solution

static void solve3()

{

results3();

cout << "Probability of values: \n";

for (int i = 1; i < 45; i++)

{

if ((i - 1) % 10 == 0) cout << "\n";

if ( result3[i] != 0 ) cout << i << ": " << result3[i] << "\t"; //im too lazy to program skipping

//impossible values in a better way

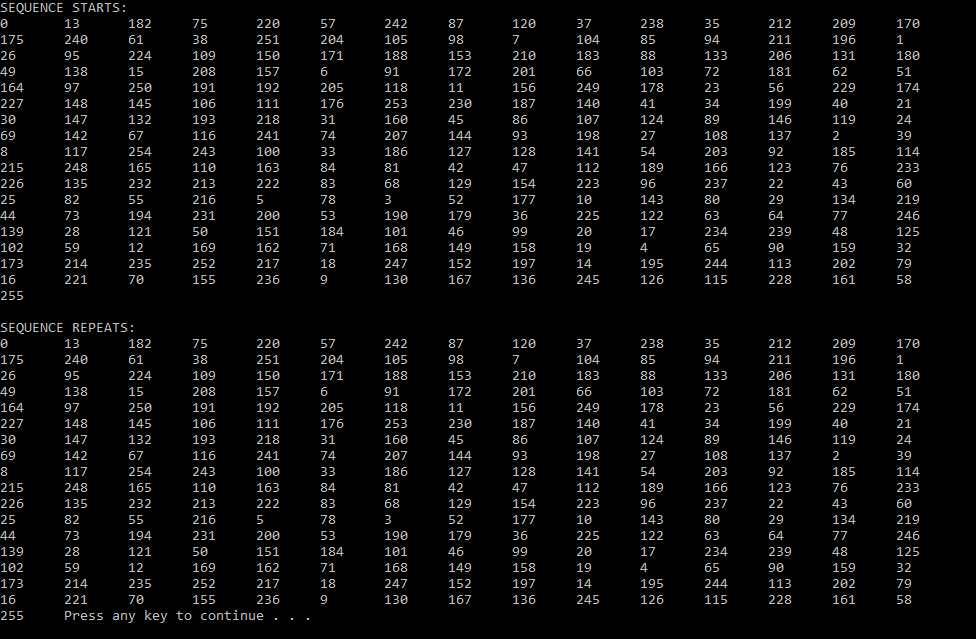
}

}

**PROBLEM 4**

**PROBLEM 5**

The maximum sequence size possible is 256, achieved with a = 15, c = 15.



#include <iostream>

static int lehmerGenerator( int x )

{

static int a = 13;

static int c = 13;

static int m = 256;

return ( a \* x + c ) % m;

}

static void solve5()

{

static const int N = 256;

static int value = 0;

std::cout << "SEQUENCE STARTS: \n";

for (int i = 0; i < N; i++)

{

std::cout << value << "\t";

value = lehmerGenerator(value);

}

std::cout << "\n\nSEQUENCE REPEATS: \n";

for (int i = 0; i < N; i++)

{

std::cout << value << "\t";

value = lehmerGenerator(value);

}

}

**PROBLEM 6**



Number points = 100000

Number trials = 10

#include <cstdlib>

#include <time.h>

#include <iostream>

static double f(double x) //the function being tested

{

return exp(x);

}

static double monteCarlo( double upperBound )

{

static const int N = 100000; //number points

static int n = 0; //number points within region

static double x, y, result;

srand(time(NULL));

static double maxFunctionHeight = f(upperBound); //height of rectangle

static double rectangleArea = upperBound \* f(upperBound); //area rectangle

n = 0;

for (int i = 0; i < N; i++) //plot N points

{

x = (double)rand() / (double)RAND\_MAX;

y = maxFunctionHeight \* (double)rand() / (double)RAND\_MAX;

if (f(x) >= y) n++;

}

result = rectangleArea \* (double)n / double(N);

return result;

}

static void solve7(double x)

{

static double sumTrials = 0;

static const int NUMBER\_TRIALS = 10;

for (int i = 0; i < NUMBER\_TRIALS; i++)

{

sumTrials += monteCarlo(x);

}

std::cout << "Area under function exp(x) from 0 to x: " << sumTrials / NUMBER\_TRIALS << "\n";

}

**PROBLEM 7**

xav = 10( 1/ sec )

sav = 50[10]-3

**PROBLEM 8**

Sav = 20[10]-3 seconds

U = .4

**a)**

R = S/(1-U2) = 20(10)-3 / ( 1 - .16 )

= .0238 seconds

= 23.8 ms

**b)**

U = .8

R = S/(1-U) = 20(10)-3 / ( 1 - .8 )

= 1 second

= 100 ms

**PROBLEM 9**