"Given a population of $N = 500$ with $A = 200$ successes, a sample of $n = 100$ is taken, we would like to find the probability of any ratio of success within the smaller sample, say 55%."

This can be formulated into a hypergeometric distribution of the following formula:

$$P(X = x|n, N, A) = \frac{\binom{A}{x}\binom{N-A}{n-x}}{\binom{N}{n}}$$

With values substituted in, we have this statement:

$$P(X = 55|n = 100, N = 500, A = 200) = \frac{\binom{200}{55}\binom{500-200}{100-55}}{\binom{500}{100}}$$

Simplified as…

$$P(X = 55|n = 100, N = 500, A = 200) = \frac{\binom{200}{55}\binom{300}{45}}{\binom{500}{100}}$$

Expanding using the combinatorial function

$$\binom{n}{r} = \frac{n!}{r!\,(n-r)!}$$

This statement would be further complicated as

$$P(X = 55) = \frac{\left(\frac{200!}{55!\,145!}\right)\left(\frac{300!}{45!\,255!}\right)}{\left(\frac{500!}{100!\,400!}\right)} = \left(\frac{200!}{55!\,145!}\right)\left(\frac{300!}{45!\,255!}\right)\left(\frac{400!\,100!}{500!}\right)$$

Symbolically calculated using Mathematica as:

```
Binomial[200,55]*Binomial[300,45]/Binomial[500,100]
```

Gave a result of

$$P(X = 55) = 0.000288649$$

In the event of academic settings, where students are restricted from using tools designed prior to the past two decades, such a calculation would be unwieldy. I considered using Stirling's approximation as a means to find this ratio, but the losses of the approximation would likely lose the precision desired.

What I decided to do was determine the feasibility of designing a calculating algorithm for the TI-83. I first tested my algorithm in the C programming language. My method produces results regardless of the scale of the numbers given to the calculator. Using a dynamically growing linked list structure, I

designed a symbolic mathematics engine for reducing the ratio down to two very large relatively prime numbers. By minimizing the number of floating point operations, the floating point error is minimized, ensuring accurate results.

This provides accurate results, even for large population values. As designed, the final multiplications are only moderately optimized for minimizing floating point operations. For another 3 hours of effort, I could create a scheme for grouping the remaining prime terms into a linked-list of unsigned, long-integer data types by performing lossless integer multiplication up to the level where the architecture of the CPU is at it's limits. At this point, the long integers would by multiplied in a floating point arithmetic using the fewest possible operations in order to preserve as many digits of precision as possible.

The execution of this program finds the result in $O(n^2)$ time.

For the given problem,

$$N = 500 \quad A = 200 \quad n = 100 \quad x = 55$$

The program returned:

```
Result is 2.8864853352e-04
```

in 0.009 seconds.

```
// This code symbolically calculates a Hypergeometric Probability with
// arbitrary population size
//
//  P(X=x|n,N,A)=Binomial(A,x)*Binomial(N-A,n-x)/Binomial(N,n)
//
// For public domain, by Timothy D. Legg

//There parameters defined here
#define N 500
#define A 200
#define n 100
#define x 55
//state that "From a population of N events, there are A known successes.
//from a sample of n items, what is the probability of x successes?"

//Method:  These numbers are used to generate integers in arrays if the
//         numerator and denominator.  Normally, these numbers are factorials
//         that are multiplied and then divided, but that may overflow the
//         capabilities of the hardware.  Instead, the factorials are prime
//         expanded into a linked list for each the numerator and denominator.
//         As a next step, terms are cancelled from each and remaining terms
//         then multiplied and divided.

#define SHOWCHAINS 0
#define SHOWFINALCHAINS 0

#include<stdio.h>
#include<stdlib.h>
struct node
      {
            int val;
            struct node *prev;
            struct node *next;
      };


//Expands number into a linked list of prime factors
//Warning: last element in linked list is value 1.
void expand(int a, struct node **end)
{
      int c=2;
      struct node *temp;
      while(c<=a)
      {
            //This case, a prime factor is found
            if(a%c==0)
            {
                  a=a/c;
                  (*end)->val = c;
                  (*end)->next = malloc( sizeof(struct node) );
                  temp = (*end);
                  (*end) = (*end)->next;
                  (*end)->prev=temp;
            }
```

```
            //This case, no prime factor is found
            if(a%c!=0)
            {
                  c++;
            }
      }
      (*end)->val=1;
      return;
}

void liminate(struct node **kablooie)
{
      struct node *previous;
      struct node *nextone;
      if( ((*kablooie)->prev==NULL)  &&  ((*kablooie)->next==NULL) )
      {
            (*kablooie)->val=1;
            printf("Warning: Something cancelled out to unity.\n");
            return;
      }
      if((*kablooie)->prev==NULL)
      {
            nextone = (*kablooie)->next;
            nextone->prev=NULL;
            free(*kablooie);
            (*kablooie)=nextone;
            return;
      }
      if((*kablooie)->next==NULL)
      {
            previous = (*kablooie)->prev;
            previous->next=NULL;
            free(*kablooie);
            (*kablooie)=previous;
            return;
      }
      previous = (*kablooie)->prev;
      nextone  = (*kablooie)->next;
      previous->next = (*kablooie)->next;
      nextone->prev = (*kablooie)->prev;
      free(*kablooie);
      (*kablooie)=previous;
      return;
}
void main()
{
      int i,j;
      int tops[4]={A,N-A,n,N-n};
      int bottoms[5]={x,A-x,n-x,N-A-n+x,N};
      struct node *numerator, *denominator, *traveler, *wanderer;
      double result;

      //
      //Create Numerator Chain
      //
      numerator = (struct node *) malloc( sizeof(struct node));
```

```
numerator->prev=NULL;
//I want to keep a copy of the start address, leaving it unchanged
traveler=numerator;
//expand that number into prime factors
for(j=0;j<4;j++)
{
      //printf("Array numerator val %d\n", tops[j]);
      for(i=tops[j];i>1;i--)
      {
            expand(i,&traveler);
      }
}
traveler->next = NULL;
//
//Numerator Chain Complete
//


//print the chain
if(SHOWCHAINS)
{
      traveler=numerator;
      printf("Printing the numerator chain\n");
      printf("Factors are");
      while(traveler->next!=NULL)
      {
            //numerator=numerator->next;
            printf(" %d ", traveler->val);
            traveler=traveler->next;
      }
      printf("\n");
}

  //
  //Create Denominator Chain
  //
  denominator = (struct node *) malloc( sizeof(struct node));
denominator->prev=NULL;
  //I want to keep a copy of the start address, leaving it unchanged
  traveler=denominator;
  //expand that number into prime factors
  for(j=0;j<5;j++)
  {
          //printf("Array denominator val %d\n", bottoms[j]);
          for(i=bottoms[j];i>1;i--)
          {
                  expand(i,&traveler);
          }
  }
  traveler->next = NULL;
  //
  //Denominator Chain Complete
  //

  //print the chain
  if(SHOWCHAINS)
```

```
{
      traveler=denominator;
      printf("Printing the denominator chain\n");
        printf("Factors are");
      while(traveler->next!=NULL)
        {
            printf(" %d ", traveler->val);
                traveler=traveler->next;
        }
        printf("\n");
}

//Cancellation of terms
traveler=numerator;
wanderer=denominator;
while(wanderer->next!=NULL)
{
      while(traveler->next!=NULL)
      {
            if(traveler->val==wanderer->val)
            {
                  liminate(&traveler);
                  liminate(&wanderer);
                  if(traveler->prev==NULL)
                  {
                        numerator=traveler;
                  }
                  else
                  {
                        traveler=numerator;
                  }
                  if(wanderer->prev==NULL)
                  {
                        denominator=wanderer;
                  }
            }
            else
            {
                  traveler=traveler->next;
            }
      }
      wanderer=wanderer->next;
      traveler=numerator;
}

//print simplified numerator chain
  //print the chain
  if(SHOWFINALCHAINS)
  {
      traveler=numerator;
          printf("Printing the simplified numerator chain\n");
          printf("Factors are\n");
          while(traveler->next!=NULL)
          {
                  //numerator=numerator->next;
                  printf("%d*", traveler->val);
```

```
                        traveler=traveler->next;
                }
                printf("\n");
        }
      //print the simplified denominator chain
        //print the chain
        if(SHOWFINALCHAINS)
        {
            traveler=denominator;
                printf("Printing the simplified denominator chain\n");
                printf("Factors are\n");
                while(traveler->next!=NULL)
                {
                        //denominator=denominator->next;
                        printf("%d*", traveler->val);
                        traveler=traveler->next;
                }
                printf("\n");
        }

      result=1.0;
      //calculate result

      while(numerator->next!=NULL)
      {
            result = result * (double)(numerator->val);
            numerator=numerator->next;
      }
      while(denominator->next!=NULL)
      {
            result = result / (double)(denominator->val);
            denominator=denominator->next;
      }

      printf("Result is %.10e\n",result);

      return;
}
```