1.

a. The addition is computed once for each element of the matrix. Therefore, the total number of addiction is computed in this way:

$$n^2 = \mathrm{N}/2$$

b. Multiplication is a better choice as basic operation for the standard algorithm for matrix multiplication. The elements of the product of two matrices is compute as a scalar product in this way:

$$n \cdot n^2 = n^3 = (\mathrm{N}/2)^{3/2}$$

2.

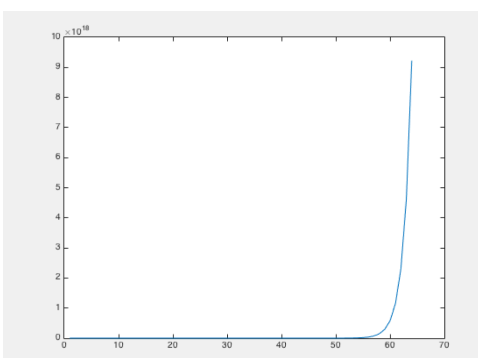
a. The best case is two, since you get the two gloves you need at the first 2 picks. The worst case is twelve, one more that the numbers of gloves.

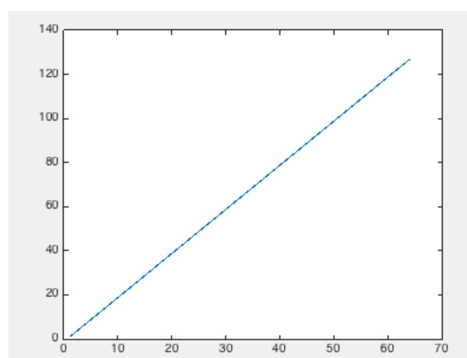
b. The possible results are two. The first one is that the two missing sock are paired, the second is that they are not paired.

The total number of outcomes is calculated as $\binom{10}{2}$ or $\frac{10!}{2!(10-2)!} = 45$

Therefore, the best case scenario is 5 and the probability is $p = \frac{5}{45} = \frac{1}{9}$ and the probability of the worst case scenario is $q = 1 - p = 1 - \frac{1}{9} = \frac{8}{9}$

Indeed, $E_{(x)} = 4 \cdot \frac{1}{9} + 3 \cdot \frac{8}{9} = \frac{28}{9}$





3.

a. As we can se from the first graph the result is $1.8 \cdot 10^{19}$ This can also be derived from the subsequent formula:

$$\sum_{i=1}^{64} 2^{i-1} = \sum_{j=0}^{63} 2^j = 2^{64} - 1 = 1.8 \cdot 10^{19}$$

b. As we can see from the second graph the result is 128 grains of beans.

4.

```
function [ sqrt ] = BabylonianSquare( numb )
strNumb = num2str(numb);
split = strsplit(strNumb, '.');

ArraySize = size(split);

if ArraySize(2) == 2
    leftDigits = split(1);
    rightDigits = split(2);
else
    leftDigits = split(1);
    rightDigits = '0';
end

if numb >= 1
    ArrayOfChar = char(leftDigits);
    ArrayOfCharSize = size(ArrayOfChar);
    D = ArrayOfCharSize(2);
elseif numb < 1
    ArrayOfChar = char(rightDigits);
    ArrayOfCharSize = size(ArrayOfChar);
    count = 0;
    for j=1:ArrayOfCharSize(2)
        if ArrayOfChar(j) == '0'
            count = count + 1;
        end
    end
    D = -count;
end

if mod(D,2) == 0
    D = ((2 * D) - 2)/2;
    xo = 6 * power(10,D/2);
else
    D = ((2 * D) - 1)/2;
    xo = 2 * power(10,D/2);
end

x = xo;
x1 = 0 * xo;
while ((power((x1 - xo), 2.0)) > 0.000001)
    xo = x1;
    x = 0.5 * (x + (numb / x));
    x1 = x;
end

sqrt = x;
end
```

```
numbers = [100, 23.25,98.89, 25, 10];
results = zeros(1,5);
for i = 1:5
    results(1, i) = BabylonianSquare(numbers(1,i));
end
disp(results);
```

```
nmand Window
> BabylonianScript
  10.0000     4.8218     9.9443     5.0000     3.1623

>
```

5.
```
function [ xo ] = HornerPolynomial( coefficients , x )

xo = 0;
for i = (numel(coefficients):-1:1)
    xo = (xo * x) + coefficients(i);
end

end
```

```
disp(HornerPolynomial([1, 5, 0, 3], 3));
```

```
mmand Window

>> HornerScript
   97

>>
```

6. The dominant operation of the algorithm is Multiplication.
The value of “Answer” at the end of its execution is

$$\sum_{x=1}^{n-1} x^n = \frac{x^n - x}{x - 1}$$

7.

$g(n) = n^4 \implies g(n) = O(n^4)$

therefore $F \in O(n^4)$

$2n^4 - 5n^3 + 10n^2 - 6n + 9 \leq 2n^4 + 10n^2 + 9$

$\leq 2n^4 + 10n^4 + 9n^4$

$\leq 21n^4$

We conclude that $F(n) = O(n^4) = g(n)$

8. The dominant operation in this algorithm is the ***WHILE loop***, because contains addiction and subtraction and a logical operation in itself.
The computational complexity of this algorithm is derived from:

$$\sum_{n=1}^{n-1} n = \frac{n(n-1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$$

Therefore the time complexity is $O(n^2)$