1. ABSOLUTE (x)

```
// returns |x|
if x < 0
      return -1*x
else
      return x
```

2. IS_ODD(x)

```
// returns true if x is an odd number,
// false otherwise
if x%2 == 1
      return true
else
      return false
```

3. MAX_IN_ARRAY(A)

```
// A is an array of n integers
// returns the largest element in A
max = A[1]
for i = 1 to n do
      if A[i] > max then
            max = A[i]
return max
```

4. SUM(A)

```
// A is an array of n integers
// returns the sum of all integers in A
i = 1
sum = 0
while i <= n
      sum = sum + A[i]
      i = i+1
return sum
```

5. MATRIX-ADDITION(A,B)

```
// A and B, both are n x n matrices
// returns C, n x n matrix. C = A + B
for i = 1 to n do
      for j = 1 to n do
            C[i][j] = A[i][j] +
B[i][j]
return C
```

6. EXPONENTATION_OF_2(n)

```
// prints all 2^i where 2^i <= n
i = 1
repeat
      print i
      i = i*2
until i >= n
```

7. EGYPT_MULTIPLICATION(x,y)

```
// computing x*y using egyptian method
result = 0
while x > 0
      if x%2 == 1 then
            result = result + y
      y = y*2
      x = x/2
return result
```

8. SEQUENTIAL_SEARCH(A, val)

```
// A is an array of n integers
// val is an integer that we want
// to search in A
// when found, returns the index
// returns 0 otherwise
for i = 1 to n do
      if A[i] == val then
            return i

return 0
```

9. MAX(A, idx)

```
// first call : MAX(A, 1)
if idx == n then
  return A[idx]
else
  return MAX2(A[idx], MAX(A,idx+1))
```

10. SUM(A, idx)

```
// first call : SUM(A, n)
if idx == 1 then
      return A[1]
else
      return A[idx] + SUM(A, idx-1)
```

11. FIBO(n)

```
// compute F(n) recursively
if n==0 then
      return 0
else if n==1 then
      return 1
else
      return FIBO(n-1) + FIBO(n-2)
```

12. Given two arrays: A of n elements and B of m elements

- Develop an algorithm to count how many (x,y) pairs, such that $x \in A$, $y \in B$, and $x = y$
- Compute the complexity of your algorithm !

13. Create a recursive function for these problems, then compute the complexity of your algorithm !

- multiply two integers, using only addition and recursion → MULT(a,b)
- compute the power of a number → POWER(a, n)