Chapter 2

2.1 Write an algorithm that finds the largest numbers in a list of 'n' numbers.

Solution:

Algorithm largest($A[1\cdots n]$

Begin

$i = 0$

largest = $A[0]$

while $i \leq n$ do

if $A[i] >$ largest then

largest = $A[i]$

end if

$i = i + 1$

end while

End

2.2 Write an algorithm for finding whether a number is prime or not.

***Solution*:**

Algorithm prime(n)

%% Input: Integer n

%% Output: prime or not

Begin

flag = true

$i = 2$

while $i \leq \sqrt{n}$ do

if (n mod i)==0

flag = false

exit

else

i = i + 1

End while

If flag = True then

print(" n is a prime number")

else

print ("number is not prime number")

End

2.3 Write an algorithm to find the square root of a number.

***Solution*:**

Algorithm squareroot(Q)

%% Input : Q for which square root is desired

%% Output : $\sqrt{a}$

Begin

for i = 1 to n do

$$x_{\{i+1\}} = \frac{\left(x_i + \frac{Q}{x_i}\right)}{2}$$

End for

$return\ (x_{i+1})$

End

2.4 Write an algorithm to reverse the digits of a number.

***Solution*:**

Algorithm reverse(n)

%% Input: n

%% Output: reverse of n

Begin

$y = 0$

while (n≠0) do

$y = y \times 10 + (y \; mod \; 10)$

$n = \frac{n}{10}$

end while

return (*y*)

End.

2.5 Write an algorithm to convert a number in octal to hexadecimal system.

Solution:

Hint:

Algorithm convert(*n*)

%% Input : *n*-octal number

%% Output: *n* in Hexadecimal

Begin

Convert octal to binary number

Group the binary to group of four numbers

If the left-most group has less than 4 digits,

then add zero

Convert the four-bit group to hexadecimal

number.

End

2.6 Write an algorithm for finding whether a number is a perfect number or not.

***Solution*:**

Hint: If the sum of divisors equals the number, it is a perfect number.

```
Algorithm perfect (n)
%% Input: n
%% Output: whether n is perfect or not.
Begin
  for i = 1 to n do
   if [(n mode i)==0] then
        s = s + i
   end if
   end for
   if (s==n) then
      return('perfect number')
   end if
End.
```

2.7 Let *N* be the number of guests attend a party. If each guest shakes his hand with everyone else only once, how many handshakes will take place? Write a recursive definition and algorithm.

Solution:

The first person may shake hands with *n-1* other people. The next person with *n-2* other people. This chain would go as the closed solution:

$$\frac{n \times (n-1)}{2}$$

The recursive equation would be t(n) = t(n-1) + (n-1).

Therefore, the algorithm is given as

```
Algorithm(n)
%% Input: n
```

%% Output: number of handshakes

Begin

number $= \frac{n \times (n-1)}{2}$

return (number)

End

2.8 Use a recursive algorithm of factorial and write a recursive routine for finding nC_r and nP_r. Recollect that

$$nC_r = \frac{n!}{r!(n-r)!} \text{ and } nP_r = \frac{n!}{(n-r)!}$$

***Solution*:**

Both use factorial problem. The algorithm for finding factorial *NFACT* can be written as follows:

Algorithm *NFACT*(*n*)

%% Input: Integer *n*

%% Output: Factorial of *n*

Begin

for *i* = 1 to *n* do

result = $result \times i$

end for

End

Now the algorithm for nC_r and nP_r can be obtained as follows:

Algorithm *NCR*(*n*,*r*)

%% Input : Integer *n* and *r*

%% Output: *result*

Begin

$$result = \frac{NFACT(n)}{NFACT(r) \times NFACT(n-r)}$$

return (*result*)

End

Algorithm *NPR*(*n*,*r*)

%% Input : Integer *n* and *r*

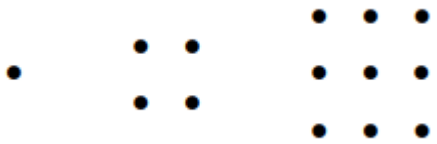
%% Output: *result*

Begin

return (*result*)

End

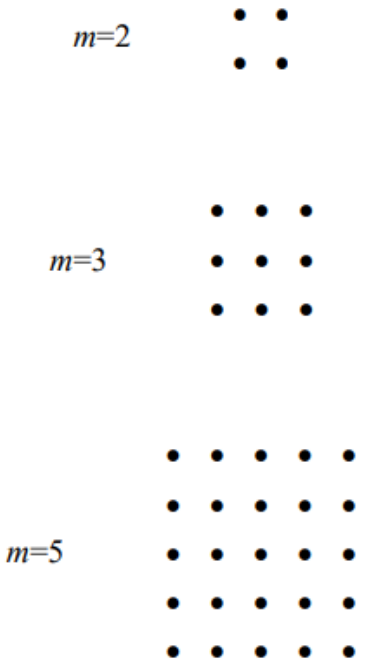
2.9 The following are square numbers, followed by dots.



What is the recurrence equation for constructing higher order square numbers?

***Solution*:**

m=1



The formula is

$$a_n = a_{n-1} + 2n - 1$$

One can check that

$$a_1 = 1$$
$$a_2 = 1 + 4 - 1 = 4$$
$$a_3 = a_2 + 6 - 1 = 4 + 6 - 1 = 9$$

Hence the algorithm would be

Algorithm square(*n*)

%% Input = *n* , *howmany*

%% Output = *square*

Begin

index = 1

while *index<howmany*
k = 1
A[k] = 1
for *k* = 2 to *n*

*A[k]= A[k-1] + 2 * k - 1*

end for

return (A[*result])*

end while

End

2.10 The Lucas number is same as Fibonacci number. It is given by the following recurrence equation.

$$L_1 = 2$$
$$L_2 = 1$$
$$L_n = L_{n-1} + L_{n-2}$$

Write a recurrence algorithm.

***Solution*:**

Algorithm Lucas (*limit*)

%% Input : *limit* for which Lucas numbers to be generated

%% Output: Lucas numbers

Begin

x = 2

y= 1

while (*limit*>*y*)

result = *x* + *y*

x = *y*

y = *z*

End while

return (*result*)

end for

End

$$GCD(n,m) = \begin{cases} m & if\ n = 0 \\ GCD(n, m \bmod n) & if\ n \geq m \end{cases}$$

2.11 Write a recursive program for computing the GCD of two numbers. The recursivedefinition is given as follows:

***Solution*:**

Algorithm GCD(*n*,*m*)

%% Input : *m* and *n*

%% Output: GCD(*m*,*n*)

Begin

if (n mod *m*) == 0 then

return *m*

else

return GCD(*n, m* mod *n*)

$$L(N) = \begin{cases} 1 & if\ N = 1 \\ 1 + L(N-1) & otherwise \end{cases}$$

end if

End

2.12 The length *L* of a list of *N* number *L*(*N*) calculated as follows :\

Write a recursive algorithm

Solution:

Algorithm *length*(*L*,*N*)

%% Input: List *L* and length *N*

%% Output: *length*

Begin

if (*L* == Null) then %% empty list

return 0

else

return (1+*length*(*n*-1))

end if

End

2.13 Find Ulam length for the Collatz series. The following is the definition of the Ulam length.

$$Ulamlength\ h(x) = \begin{cases} 1 & if\ x = 0 \\ 1 + ulamlength(3x_{n-1}) & if\ x > 1\ and\ even \\ 1 + ulamlength\left(\frac{x_{n-1}}{2}\right) & if\ x > 1\ and\ odd \end{cases}$$

Solution:

Algorithm *ulam*(*x*)

%% Input : *x*

%% Output: *length*

Begin

If (*x*<= 1) thenreturn (1)

Else

if (x mod 2! = 0) then

return(1 + *ulam*(3*x*+1))

else

return(1+*ulam*($\frac{x}{2}$))

End if

End if

End.

2.14 Prove that the following algorithm segment with a loop invariant $i + j = 20$.

Solution:

The main segment of the algorithm is

Begin

$\left.\begin{matrix} j = 20 \\ i = 0 \end{matrix}\right\}$ *Take sum* $i + j = 20$ The invariant is true

for *s* = 1 to 20 do

j = *j* – 1
i = i + 1

end for

End

Hint : Let *i´* and *j´* be the condition before execution and

i´´ and *j´´* be the condition after the execution. Then,

j´´ = *j´* - 1 (as *j´* = 20 – *i*)

= 20 – *i´* - 1

i´´ = *i´* + 1

$\therefore i'' + j'' = 20 - i' - 1 + i' + 1$

$= 20$

$\therefore i' + j' \Rightarrow i'' + j''$

$\therefore$ The algorithm is right.

2.15 Write a recursive program for returning triangular numbers and provide proof of correctness.

Solution:

$triang(n) = \frac{n(n+1)}{2}$ (closed solution)

The algorithm can written as

Algorithm *triang*(*n*)

%% Input : Integer *n*

%% Output : resultBegin

if (*n*==1) then return(*triang*(*n*))

else

return(*triang*(*n*-1)**n*)

endif

End

Proof:

The recursive equation is $T(n) = n + T(n+1)$

$T(1) = 1$

When $n = 1$ $t_1 = 1$

$n = 2 \quad t_2 = 2 \times t_1 = 3$

$t_3 = 3 \times t_2 = 6$

$t_4 = 4 \times t_3 = 10$

Hence, the sequence is 1, 3, 6, 10

So the closed formula is $T_n = \frac{n(n+1)}{2}$

One can prove this by mathematical Induction as

$$T(n) = \frac{n(n+1)}{2}$$

Replace n by n+1

$$T(n+1) = \frac{\big((n+1)+1\big)}{2}\ (n+1+1)$$

$$= \frac{(n+1)+1}{2}((n+1)+1)$$

∴ It is proved