In [11]:

```
#PracticalNo1 {Non-Recursive}

nterms = int(input("Enter number of terms "))
n1, n2 = 0, 1
count = 0

if nterms <= 0:
    print("Please enter a positive integer")

elif nterms == 1:
    print("Fibonacci sequence upto", nterms,":")
    print(n1)

else:
    print("Fibonacci sequence:")
    while count < nterms:
        print(n1)
        nth = n1 + n2
        n1 = n2
        n2 = nth
        count += 1
```

```
Enter number of terms 10
Fibonacci sequence:
0
1
1
2
3
5
8
13
21
34
```

In [3]:

```
#PracticalNo1 {Recursive}

def fibonacci(n):
    if(n <= 1):
        return n
    else:
        return(fibonacci(n-1) + fibonacci(n-2))
n = int(input("Enter number of terms:"))
print("Fibonacci sequence:")
for i in range(n):
    print(fibonacci(i))
```

```
Enter number of terms:10
Fibonacci sequence:
0
1
1
2
3
5
8
13
21
34
```

In [12]:

```
#PracticalNo2 {Huffman}

import heapq
class node:
    def __init__(self,freq,symbol,left=None,right=None):
        self.freq=freq
        self.symbol=symbol
        self.left=left
        self.right=right
        self.huff= ''

    def __lt__(self,nxt):
        return self.freq<nxt.freq

def printnodes(node,val=''):
    newval=val+str(node.huff)
    if node.left:
        printnodes(node.left,newval)
    if node.right:
        printnodes(node.right,newval)

    if not node.left and not node.right:
        print("{} -> {}".format(node.symbol,newval))

if __name__=="__main__":
    chars = ['a', 'b', 'c', 'd', 'e', 'f']
    freq = [ 5, 9, 12, 13, 16, 45]
    nodes=[]

    for i in range(len(chars)):
        heapq.heappush(nodes, node(freq[i],chars[i]))

    while len(nodes)>1:
        left=heapq.heappop(nodes)
        right=heapq.heappop(nodes)

        left.huff = 0
        right.huff = 1
        newnode = node(left.freq + right.freq , left.symbol + right.symbol , l
        heapq.heappush(nodes, newnode)

    printnodes(nodes[0])
```

```
f -> 0
c -> 100
d -> 101
a -> 1100
b -> 1101
e -> 111
```

In [7]:

```
#PracticalNo3 {Knapsack}

def fractional_knapsack():
    weights=[10, 20, 30]
    values=[60,100,120]
    capacity=50
    res=0
    for pair in sorted(zip(weights,values), key= lambda x: x[1]/x[0], reverse=
        if capacity<=0:
            break
        if pair[0]>capacity:
            res+=int(capacity * (pair[1]/pair[0]))
            capacity=0
        elif pair[0]<=capacity:
            res+=pair[1]
            capacity-=pair[0]
    print(res)

if __name__=="__main__":
    fractional_knapsack()
```

```
240
```

In [10]:

```
#PracticalNo5 {Dynamic Knapsack}


def knapSack(W, wt, val, n):

    if n == 0 or W == 0:
        return 0

    if (wt[n-1] > W):
        return knapSack(W, wt, val, n-1)
    else:
        return max(
            val[n-1] + knapSack(
                W-wt[n-1], wt, val, n-1),
            knapSack(W, wt, val, n-1))

if __name__ == '__main__':
    profit = [60, 100, 120]
    weight = [10, 20, 30]
    W = 50
    n = len(profit)
    print (knapSack(W, weight, profit, n))
```

```
220
```