```
####################################
# EXAMPLE:  Towers of Hanoi
####################################

def printMove(fr, to):
    print('move from ' + str(fr) + ' to ' + str(to))

def Towers(n, fr, to, spare):
    if n == 1:
        printMove(fr, to)
    else:
        Towers(n-1, fr, spare, to)
        Towers(1, fr, to, spare)
        Towers(n-1, spare, to, fr)

#print(Towers(4, 'P1', 'P2', 'P3'))

####################################
# EXAMPLE:  fibonacci
####################################

def fib(x):
    """assumes x an int >= 0
       returns Fibonacci of x"""
    if x == 0 or x == 1:
        return 1
    else:
        return fib(x-1) + fib(x-2)

####################################
# EXAMPLE:  testing for palindromes
####################################

def isPalindrome(s):

    def toChars(s):
        s = s.lower()
        ans = ''
        for c in s:
            if c in 'abcdefghijklmnopqrstuvwxyz':
                ans = ans + c
        return ans

    def isPal(s):
        if len(s) <= 1:
            return True
        else:
            return s[0] == s[-1] and isPal(s[1:-1])

    return isPal(toChars(s))

#print(isPalindrome('eve'))
#
#print(isPalindrome('Able was I, ere I saw Elba'))
```

```
#
#print(isPalindrome('Is this a palindrome'))


####################################
# EXAMPLE: comparing fibonacci using memoization
####################################
def fib(n):
    if n == 1:
        return 1
    elif n == 2:
        return 2
    else:
        return fib(n-1) + fib(n-2)

def fib_efficient(n, d):
    if n in d:
        return d[n]
    else:
        ans = fib_efficient(n-1, d)+fib_efficient(n-2, d)
        d[n] = ans
        return ans

d = {1:1, 2:2}

argToUse = 34
print("")
print('using fib')
print(fib(argToUse))
print("")
print('using fib_efficient')
print(fib_efficient(argToUse, d))

########################################
####search
########################################
def linear_search(L, e):
    found = False
    for i in range(len(L)):
        if e == L[i]:
            found = True
    return found

testList = [1, 3, 4, 5, 9, 18, 27]

def search(L, e):
    for i in range(len(L)):
        if L[i] == e:
            return True
        if L[i] > e:
            return False
    return False

########################################
```

```
#### subset
########################################
def isSubset(L1, L2):
    for e1 in L1:
        matched = False
        for e2 in L2:
            if e1 == e2:
                matched = True
                break
        if not matched:
            return False
    return True

testSet = [1, 2, 3, 4, 5]
testSet1 = [1, 5, 3]
testSet2 = [1, 6]

########################################
####get the intersection of two lists
########################################
def intersect(L1, L2):
    #get the intersection
    tmp = []
    for e1 in L1:
        for e2 in L2:
            if e1 == e2:
                tmp.append(e1)

    #remove the duplicate
    res = []
    for e in tmp:
        if not(e in res):
            res.append(e)
    return res

########################################################
####bisection in a sorted list, you have to sort first
########################################################
def bisect_search2(L, e):
    def bisect_search_helper(L, e, low, high):
        print('low: ' + str(low) + '; high: ' + str(high))  #added
to visualize
        if high == low:
            return L[low] == e
        mid = (low + high)//2
        if L[mid] == e:
            return True
        elif L[mid] > e:
            if low == mid: #nothing left to search
                return False
            else:
                return bisect_search_helper(L, e, low, mid - 1)
        else:
            return bisect_search_helper(L, e, mid + 1, high)
```

```
    if len(L) == 0:
        return False
    else:
        return bisect_search_helper(L, e, 0, len(L) - 1)

testList = []
for i in range(100):
    testList.append(i)
print(bisect_search2(testList, 76))

################################################
####generate subset using recursion
################################################
def genSubsets(L):
    if len(L) == 0:
        return [[]] #list of empty list
    smaller = genSubsets(L[:-1]) # all subsets without last element
    extra = L[-1:] # create a list of just last element
    new = []
    for small in smaller:
        new.append(small+extra)  # for all smaller solutions, add
one with last element
    return smaller+new  # combine those with last element and those
without

def generateSubSet(S):
    if S == []:
        return [[]]
    else:
        smaller = generateSubSet(S[1:]) # all subsets without first
element
        extra = S[:1] # create a list of just first element
        #extra = S[0] # cannot be this, since it's not a list
anymore,
                      #rather, it's a single  element of type int
for case here
        new = []
        for item in smaller:
            new.append(item + extra)
    return smaller + new

testSet = [1,2,3,4]
print(genSubsets(testSet))
print(generateSubSet(testSet))


################################################
####bubble sort of order O(n^2)
################################################
def bubble_sort(L):
    swap = False   # set up a flag
    while not swap:
        print('bubble sort: ' + str(L))
        swap = True   # put down the flag
```

```
        for j in range(1, len(L)):
            if L[j-1] > L[j]:
                swap = False   # under which condition, put up the
flag
                temp = L[j]
                L[j] = L[j-1]
                L[j-1] = temp

testList = [1,3,5,7,2,6,25,18,13]

print('')
print(bubble_sort(testList))
print(testList)

################################################
####selection sort of order O(n^2)
################################################
def selection_sort(L):
    suffixSt = 0
    while suffixSt != len(L):
        print('selection sort: ' + str(L))
        for i in range(suffixSt, len(L)):
            if L[i] < L[suffixSt]:
                L[suffixSt], L[i] = L[i], L[suffixSt]
        suffixSt += 1

testList = [1,3,5,7,2,6,25,18,13]

print('')
print(selection_sort(testList))
print(testList)

################################################
####merge sort of order O(nlog(n))
################################################
def merge(left, right):
    result = []
    i,j = 0,0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    while (i < len(left)):    # right side is empty, left side still
remain
        result.append(left[i])
        i += 1
    while (j < len(right)):   # left side is empty, right side still
remain
        result.append(right[j])
        j += 1
```

```
    print('merge: ' + str(left) + '&' + str(right) + ' to '
+str(result))
    return result

def merge_sort(L):
    print('merge sort: ' + str(L))

    if len(L) < 2:
        return L[:]

    else:
        middle = len(L)//2
        left = merge_sort(L[:middle])
        right = merge_sort(L[middle:]) # divide and conquer
        return merge(left, right)      # then merge

testList = [1,3,5,7,2,6,25,18,13]

#print('')
#print(merge_sort(testList))
```