Exercise Sheet 2

April 13, 2016

0.1 Exercise 1

In [1]:
```
def f(n):
    if(n%2 == 0):
        return n/2
    else:
        return 3*n + 1

def fi(i,n):
    if i == 0:
        return n
    else:
        return f(fi(i-1, n))

def g(n):
    smallestI = 0
    i=0
    while smallestI != 1:
        smallestI = fi(i, n)
        i = i + 1
    return i - 1

n = int(input("Number: "))
print("The smallest number i for fi(n) is: ")
print(g(n))
```

```
Number: 1000
The smallest number i for fi(n) is:
111
```

0.2 Exercise 2

iterative function

In [24]:
```
def is_palindrome(string):
        for i in range(0,len(string)//2):
                if string[i] == string[-(i+1)]:
                        print(string[i], string[-(i+1)])

                else:
                        return False
        return True

print(is_palindrom_rec('anna'))
```

```
        print(is_palindrom_rec('AlgorithmssmhtiroglA'))
        print(is_palindrom_rec('Fail'))
```

```
(True, '1 recursive steps needed')
(True, '9 recursive steps needed')
(False, 'Stopped at recursive step #0')
```

Recursive function

```
In [23]: def is_palindrom_rec(string, step=-1):
                 if string:
                         step += 1
                         if (string[0] == string[-1]):
                                 return is_palindrom_rec(string[1:-1], step)
                         else:
                                 return False, 'Stopped at recursive step #{}'.format(step)
                 else:
                         return True, '{} recursive steps needed'.format(step)

        print(is_palindrom_rec('anna'))
        print(is_palindrom_rec('AlgorithmssmhtiroglA'))
        print(is_palindrom_rec('Fail'))
```

```
(True, '1 recursive steps needed')
(True, '9 recursive steps needed')
(False, 'Stopped at recursive step #0')
```

0.3 Exercise 3

```
In [1]: def bisection(array, search):

            half = None
            #copy array to avoid changes to original array (Python works with references)
            temp = array.copy()
            min_element = temp[0]

            #Check some obvious cases, before looping
            if array[0] >= search:
                return "Found. i=",1
            elif array[-1] == search:
                return "Found. i=",len(array)
            elif array[-1] < search:
                 return "Not found. i=",len(array)+1


            #loop trhough array now
            while temp:
                half = len(temp)//2   #compute middle of array
                #print(temp, half, temp[half])

                if temp[half] < search:
                    #slice array; kick those values which aren't needed anymore.
                    #add one to 'half' to exclude that element as well
                    temp = temp[half+1: ]
```

```
            elif temp[half] > search:
                if min_element >= search:
                    return "Found. i=", array.index(min_element)+1

                else:
                    # slice array from min_element on up to middle element, excluding both
                    temp = temp[1:half]

            #finally update min element before entering next loop step
            min_element = temp[0]

    print(bisection([50,250,500,1000,1500,2500,5000,10000], 10))
    print(bisection([50,250,500,1000,1500,2500,5000,10000], 100))
    print(bisection([50,250,500,1000,1500,2500,5000,10000], 1501))
    print(bisection([50,250,500,1000,1500,2500,5000,10000], 999))
    print(bisection([50,250,500,1000,1500,2500,5000,10000], 2500))
    print(bisection([50,250,500,1000,1500,2500,5000,10000], 10001))
    print(bisection([50,250,500,1000,1500,2500,5000,10000], 10000))
    print(bisection([50,250,500,1000,1500,2500,5000,10000], 50))
```

```
('Found. i=', 1)
('Found. i=', 2)
('Found. i=', 6)
('Found. i=', 4)
('Found. i=', 6)
('Not found. i=', 9)
('Found. i=', 8)
('Found. i=', 1)
```

Explanatory code

copy a into b, and c. once via .copy(), and another time via assignment

```
In [35]: b = a.copy()
         c = a
         #change 1st element in b, see what happens
         b[0] = 255
         a,b,c
```

```
Out[35]: ([1, 2, 3, 4], [255, 2, 3, 4], [1, 2, 3, 4])
```

array a and c left unchanged

now change 1st element in c, see what happens

```
In [36]: c[0] = 666
         a,b,c
```

```
Out[36]: ([666, 2, 3, 4], [255, 2, 3, 4], [666, 2, 3, 4])
```

both a and c got changed, while b is unchanged. this demonstrates that Python works with references

0.4 Exercise 4

a) For each function call, the memory complexity is O(1). Since there are n function calls using recursion, the complexity should be O(n)

b) Compiler should be able to transform the recursive to an iterative function. According to our lecture it is possible to automate such a transformation for endrecursion at least. So a compiler should tranform the complexity to O(1)

0.5 Exercise 5

this is basically a solution for nxn. set `nrows, ncols` to `8,8` and you get the solution to b)

Complexity

For every possible branch – with max z branches – of each partial solution and a solution tree with max depth N, the worst case scenario is 1 + z + zˆ2 + zˆ3 + . . . + zˆN. So O(zˆN) should be expected to be the worst case.

In [9]:
```
# Create a list with solutions on a ROWxCOL board
# Each solution is a list representing the columns
# of the board, where the number itself represents the row.
def queens(nrows, ncols):
    if nrows <= 0:
        return [[]] # keine Dame zu setzen; leeres Brett ist Lösung
    else:
        return add_queen(nrows - 1, ncols, queens(nrows - 1, ncols))



# Check all columns, where for a given partial solution
# a queen can be place within "new_row"
# If no conflict with the partial solution,
# a new solution for the expanded board has been found
def add_queen(new_row, ncols, prev_solution):
    new_solution = []
    for solution in prev_solution:
        # Try to insert queen in each column of new_row
        for new_col in range(ncols):
            #print('Trial: %s in row %s' % (new_col, new_row))
            if no_conflict(new_row, new_col, solution):
                # no conflict => solution found
                new_solution.append(solution + [new_col])
    return new_solution

# Check whether it's possible to place a new queen at "new_row"/"new_col",
# without being able to attack another queen
def no_conflict(new_row, new_col, solution):
    # Make sure new queen does not share same row, col or
    # diagonal with another queen
    for row in range(new_row):
        if (solution[row]         == new_col              or  # same col
            solution[row] + row == new_col + new_row or  # same diagonal
            solution[row] - row == new_col - new_row):    # same diagonal
                return False
```

```python
            return True

        count = 0
        for solution in queens(8, 8):
            count +=1
            print(solution)
        print("{} solutions found.".format(count))
```

```
[0, 4, 7, 5, 2, 6, 1, 3]
[0, 5, 7, 2, 6, 3, 1, 4]
[0, 6, 3, 5, 7, 1, 4, 2]
[0, 6, 4, 7, 1, 3, 5, 2]
[1, 3, 5, 7, 2, 0, 6, 4]
[1, 4, 6, 0, 2, 7, 5, 3]
[1, 4, 6, 3, 0, 7, 5, 2]
[1, 5, 0, 6, 3, 7, 2, 4]
[1, 5, 7, 2, 0, 3, 6, 4]
[1, 6, 2, 5, 7, 4, 0, 3]
[1, 6, 4, 7, 0, 3, 5, 2]
[1, 7, 5, 0, 2, 4, 6, 3]
[2, 0, 6, 4, 7, 1, 3, 5]
[2, 4, 1, 7, 0, 6, 3, 5]
[2, 4, 1, 7, 5, 3, 6, 0]
[2, 4, 6, 0, 3, 1, 7, 5]
[2, 4, 7, 3, 0, 6, 1, 5]
[2, 5, 1, 4, 7, 0, 6, 3]
[2, 5, 1, 6, 0, 3, 7, 4]
[2, 5, 1, 6, 4, 0, 7, 3]
[2, 5, 3, 0, 7, 4, 6, 1]
[2, 5, 3, 1, 7, 4, 6, 0]
[2, 5, 7, 0, 3, 6, 4, 1]
[2, 5, 7, 0, 4, 6, 1, 3]
[2, 5, 7, 1, 3, 0, 6, 4]
[2, 6, 1, 7, 4, 0, 3, 5]
[2, 6, 1, 7, 5, 3, 0, 4]
[2, 7, 3, 6, 0, 5, 1, 4]
[3, 0, 4, 7, 1, 6, 2, 5]
[3, 0, 4, 7, 5, 2, 6, 1]
[3, 1, 4, 7, 5, 0, 2, 6]
[3, 1, 6, 2, 5, 7, 0, 4]
[3, 1, 6, 2, 5, 7, 4, 0]
[3, 1, 6, 4, 0, 7, 5, 2]
[3, 1, 7, 4, 6, 0, 2, 5]
[3, 1, 7, 5, 0, 2, 4, 6]
[3, 5, 0, 4, 1, 7, 2, 6]
[3, 5, 7, 1, 6, 0, 2, 4]
[3, 5, 7, 2, 0, 6, 4, 1]
[3, 6, 0, 7, 4, 1, 5, 2]
[3, 6, 2, 7, 1, 4, 0, 5]
[3, 6, 4, 1, 5, 0, 2, 7]
[3, 6, 4, 2, 0, 5, 7, 1]
[3, 7, 0, 2, 5, 1, 6, 4]
[3, 7, 0, 4, 6, 1, 5, 2]
[3, 7, 4, 2, 0, 6, 1, 5]
```

```
[4, 0, 3, 5, 7, 1, 6, 2]
[4, 0, 7, 3, 1, 6, 2, 5]
[4, 0, 7, 5, 2, 6, 1, 3]
[4, 1, 3, 5, 7, 2, 0, 6]
[4, 1, 3, 6, 2, 7, 5, 0]
[4, 1, 5, 0, 6, 3, 7, 2]
[4, 1, 7, 0, 3, 6, 2, 5]
[4, 2, 0, 5, 7, 1, 3, 6]
[4, 2, 0, 6, 1, 7, 5, 3]
[4, 2, 7, 3, 6, 0, 5, 1]
[4, 6, 0, 2, 7, 5, 3, 1]
[4, 6, 0, 3, 1, 7, 5, 2]
[4, 6, 1, 3, 7, 0, 2, 5]
[4, 6, 1, 5, 2, 0, 3, 7]
[4, 6, 1, 5, 2, 0, 7, 3]
[4, 6, 3, 0, 2, 7, 5, 1]
[4, 7, 3, 0, 2, 5, 1, 6]
[4, 7, 3, 0, 6, 1, 5, 2]
[5, 0, 4, 1, 7, 2, 6, 3]
[5, 1, 6, 0, 2, 4, 7, 3]
[5, 1, 6, 0, 3, 7, 4, 2]
[5, 2, 0, 6, 4, 7, 1, 3]
[5, 2, 0, 7, 3, 1, 6, 4]
[5, 2, 0, 7, 4, 1, 3, 6]
[5, 2, 4, 6, 0, 3, 1, 7]
[5, 2, 4, 7, 0, 3, 1, 6]
[5, 2, 6, 1, 3, 7, 0, 4]
[5, 2, 6, 1, 7, 4, 0, 3]
[5, 2, 6, 3, 0, 7, 1, 4]
[5, 3, 0, 4, 7, 1, 6, 2]
[5, 3, 1, 7, 4, 6, 0, 2]
[5, 3, 6, 0, 2, 4, 1, 7]
[5, 3, 6, 0, 7, 1, 4, 2]
[5, 7, 1, 3, 0, 6, 4, 2]
[6, 0, 2, 7, 5, 3, 1, 4]
[6, 1, 3, 0, 7, 4, 2, 5]
[6, 1, 5, 2, 0, 3, 7, 4]
[6, 2, 0, 5, 7, 4, 1, 3]
[6, 2, 7, 1, 4, 0, 5, 3]
[6, 3, 1, 4, 7, 0, 2, 5]
[6, 3, 1, 7, 5, 0, 2, 4]
[6, 4, 2, 0, 5, 7, 1, 3]
[7, 1, 3, 0, 6, 4, 2, 5]
[7, 1, 4, 2, 0, 6, 3, 5]
[7, 2, 0, 5, 1, 4, 6, 3]
[7, 3, 0, 2, 5, 1, 6, 4]
92 solutions found.
```

In [ ]: