1. linear

the computational complexity of fact0 is where n is the value of the input(i). This is because in the recursive implementation the number of calls is directly proportional to the input since it calls itself n times.

2. linear

the computational complexity of fact1 is where n is the value of the input(i). This is because the iterative implementation iterates over all values of the input, and therefore is proportional to the input.

3.

the computational complexity of makeset is where n is the length of the input(s). This is because the for-loop has to iterate over the entire length of the input to determine that no character is repeated in res, and therefore the amount of calculations is linearly proportional to the length of the input.

4.

The computational complexity of intersect is where n is the combined length of the inputs(s1,s2). This is because first the entire length of the strings are iterated by makeset, and then afterwards, at worst, if there is no repeated c’s in s1 or s2, it has to iterate over the entire length of the combined strings again to determine if e in s1 is in s2.

5.

def swap0(s1, s2):

assert type(s1) == list and type(s2) == list

tmp = s1[:]

s1 = s2[:]

s2 = tmp

return

s1 = [1]

s2 = [2]

swap0(s1, s2)

print s1, s2

main():

s1🡪val(s2),s2🡪val(s1)

swap0():

tmp🡪val(s1)

s1🡪val(s2)

s2🡪tmp

6.

def swap1(s1, s2):

assert type(s1) == list and type(s2) == list

return s2, s1

s1 = [1]

s2 = [2]

s1, s2 = swap1(s1, s2)

print s1, s2

Global:

s1🡪[1]

s2🡪[2]

swap

s1🡪[1]

s2🡪[2]

7.

def rev(s):

assert type(s) == list

for i in range(len(s)/2):

tmp = s[i]

s[i] = s[-(i+1)]

s[-(i+1)] = tmp

s = [1,2,3]

rev(s)

print s

global:

s 🡪 [1,2,3]

rev(s)

s🡪[3,2,1]

local:

i🡪0

tmp🡪s[0]🡪1

s[0]🡪s[-1]🡪3 [3,2,3]

s[-1]🡪tmp🡪1 [3,2,1]