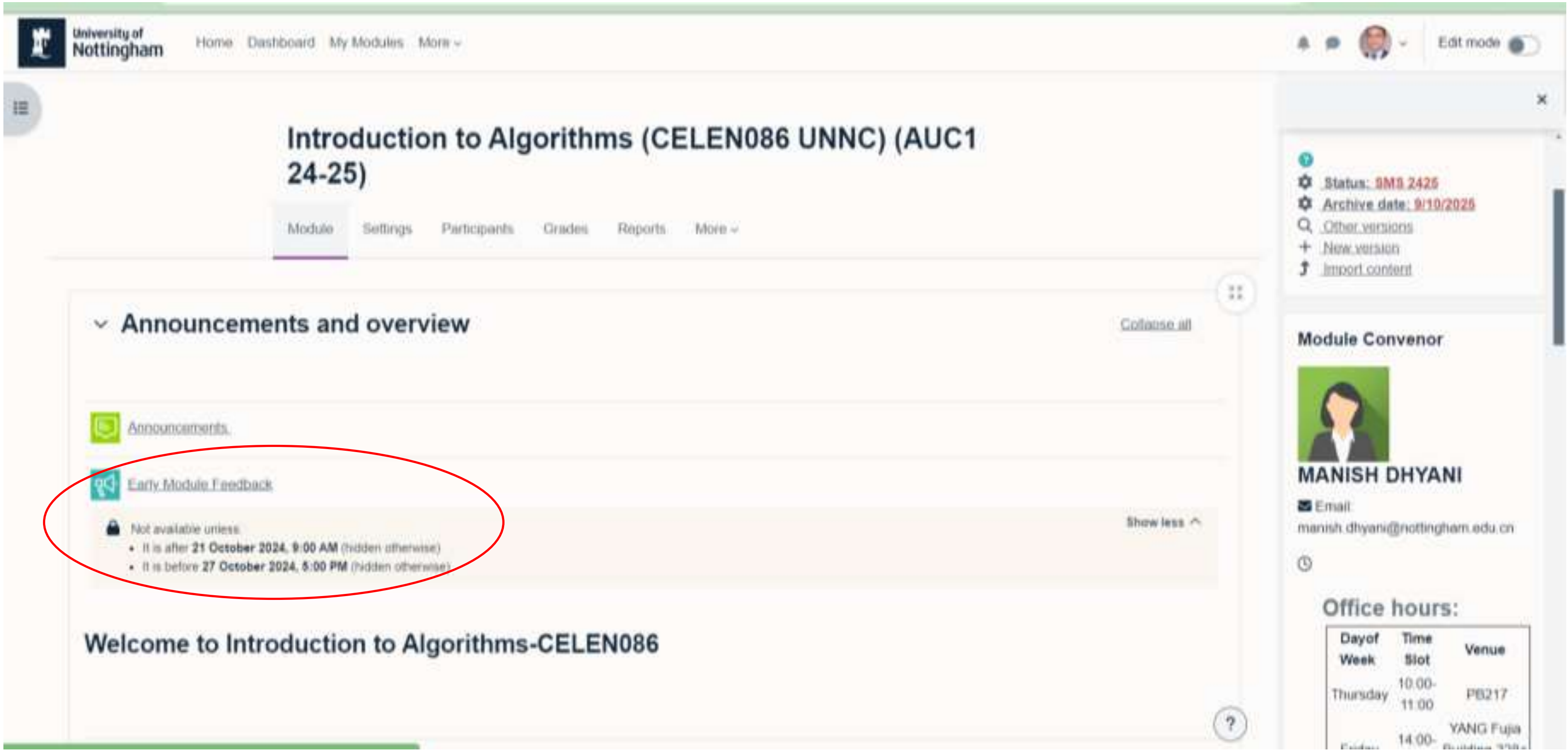
Introduction to Algorithms

CELEN086

Seminar 3
(w/c 21/10/2024)

Early Module Feedback



Available on CELEN086 Moodle page from Monday 21 October 9:00 AM to Sunday 27 October 5:00 PM.

Early Module Feedback (EMF)

Respond to 4 questions and make relevant comments about the module

1. The module content was of sufficient quality to assist my learning on this module

2. Module materials were clear about what was expected of me

3. I was given sufficient opportunity to contact my teachers/faculty on this module

4. The overall experience of studying this module has contributed to my learning

5. In your opinion, what is working well on the module so far? If there are any suggestions for the remaining weeks on the module, please also leave your comments here.

Outline

In this seminar, we will study and review on following topics:

- Design recursive algorithm
- Trace recursive algorithm
- GCD and prime number
- Helper function

You will also learn useful Math/CS concepts and vocabularies.

Recursion

When designing recursive algorithm, think about

- Can we reduce the problem into a simpler version?

  Recursive formula

- What is the simplest version that we can solve directly?

  Base case

Practise

Consider the following recursive algorithm and answer the given questions.

```
Algorithm: whoKnows( n)
Requires: an integer n
Returns : an integer
1.    if (n == 0 || n == 1)
2.          return 0
3.    else
4.          return  n + whoKnows(n - 1) + whoKnows(n - 2)
5.    endif
```

A. Identify the base case in the given algorithm **whoknows**. line 1 and line2

B. Identify the recursive step in the given algorithm **whoknows**. line 3 and line 4

C. Trace the given algorithm for n =4.

whoknows(4) -------- 4 + whoknows(3) + whoknows(2)=

Trace whoknows(4):

11 whoknows(4)

4 + whoknows(3) + whoknows(2)

5 2

3+ whoknows(2) + whoknows(1)

2 + whoknows(1) + whoknows(0)

2 0 0 0

2+ whoknows(1) + whoknows(0)

0 0

Example

Design a algorithm sum(n) that computes sum of the first n positive integers.{without recursion}

Algorithm: sum(n)
Requires: one positive integer n
Returns: sum of first n positive integers

1. Let s $= n * (n + 1)/2$
2. return s

Example

Design a recursive algorithm sum(n) that computes sum of the first n positive integers.

Algorithm: sum(n)
Requires: one positive integer n
Returns: sum of first n positive integers

```
if  n == 1
    return  1 // base case
else
    return  n+ sum(n-1) // recursive step
endif
```

How many function calls we have made?

4 times (including the first call sum(4)).

Trace sum(4)

n=4
n==1? False. return 4+sum(3) =4+6=10
n=3
n==1? False. return 3+sum(2) =3+3=6
n=2
n==1? False. return 2+sum(1) =2+1=3
n=1
n==1? True. return 1

backtracking

Practice

Write a recursive algorithm power(x,n) that computes x^n.

Algorithm: power(x,n)
Requires: a number x and a positive integer n
Returns: the value x^n

```
if n == 1
    return x
else
    return x*power(x,n-1)
endif
```

Trace power(5,3)

x=5, n=3

n==1? False. return 5*power(5,2) =5*25=125

x=5, n=2

n==1? False. return 5*power(5,1) =5*5=25

x=5, n=1

n==1? True. return 5

backtracking

Modify the above algorithm to consider for both values of n positive or negative.

Practice

Design a recursive algorithm to multiply two numbers **<u>with out using multiplication operation.</u>**

Algorithm: mul(m,n)
Requires: two integers m and n.
Returns: an integer i.e. result of $m * n$

Greatest Common Divisor

Trace gcd(64,48).

x=64, y=48

Line 1 False, Line 3 False

Line 6: return gcd(16,48)

x=16, y=48

Line 1 False, Line 3 True

Line 4: return gcd(16,32)

x=16, y=32

Line 1 False, Line 3 True

Line 4: return gcd(16,16)

x=16, y=16

Line 1 True

Line 2: return 16

Algorithm: gcd(x,y)
Requires: two positive integer x and y
Returns: the greatest common divisor

```
1. if x == y
2.       return x
3. elseif x < y
4.       return gcd(x,y-x)
5. else
6.       return gcd(x-y,y)
7. end
```

Do we need backtrack? Not necessary.

In tail recursion, there is no other operation to perform after executing the recursive function itself; the function can directly return the result of the recursive call.

Practice

Trace Euclid(308,161).

x=308, y=161

Line 4: return Euclid(161, 308 mod 161)

x=161, y=147

Line 4: return Euclid(147, 161 mod 147)

x=147, y=14

Line 4: return Euclid(14, 147 mod 14)

x=14, y=7

Line 4: return Euclid(7, 14 mod 7)

x=7, y=0

Line 2: return 7

Algorithm: Euclid(x,y)
Requires: two positive integer x and y, x>=y
Returns: the greatest common divisor

```
1. if y == 0
2.       return x // base case
3. else
4.       return Euclid(y, x mod y)
5. endif
```

In the header, is the condition x>=y necessary?

What will happen if the user call Euclid(161,308)?

Redesign the gcd algorithm

Properties of GCD:

- Property 1: gcd(x,y) = x, if x == y;
- Property 2: gcd(x,y) = gcd(x,y-x), if y > x;
- Property 3: gcd(x,y) = gcd(x-y,y), if x > y.

If neither of x nor y is the divisor of the other:

gcd(x,y)=gcd(x, y mod x)

gcd(x,y)=gcd(x mod y, y)

Algorithm: gcd(x,y)
Requires: two positive integer x and y
Returns: the greatest common divisor

```
…
…// base case

…// when x>y
   return gcd(x, y mod x)

…// when x<y
   return gcd(x mod y, y)
```

Homework exercise 1:

Following this idea to design the improved algorithm in finding gcd(x,y).

- Base cases?
- Trace your complete algorithm.

It can help you better understand the Euclid algorithm structure.

<u>f(x)</u>

Requires: A number x > 0

Returns: ?????

```
return g(x, 0)
```

```
g(x, c)
Require: A number x > 0, a number c
Return: ??
if x == 1 then
    return c
else
    if x mod 2 == 0 then
        return g(x / 2, c + 1)
    else
        return g(1 + x * 3, c + 1)
    end if
end if
```

is g(x, c) recursive?

how do we know?

base case?

recursive step?

calculate f(3)

f(x)

Requires: A number x > 0

Returns: ?????

1: return g(x, 0)

calculate f(3)

answer

f(3) = g(3, 0)
= g(10, 1)
= g(5, 2)
= g(16, 3)
= g(8, 4)
= g(4, 5)
= g(2, 6)
= g(1, 7)
=7

```
g(x, c)
Require: A number x > 0, a number c
Return: ??
1: if x == 1 then
2:    return c
3: else
4:    if x mod 2 == 0 then
5:       return g(x / 2, c + 1)
6:    else
7:       return g(1 + x * 3, c + 1)
8:    end if
9: end if
```

The algorithm counts the number of steps required to get from x to 1.

<u>f(x)</u>

Requires: A number x > 0

Returns: ??

```
return  g(x, 0)
```

<u>g(x, c)</u>

Require: A number x > 0, a number c

Return: ??

```
if x == 1 then
   return c
else
   if x mod 2 == 0 then
      return g(x / 2, c + 1)
   else
      return g(1 + x * 3, c + 1)
   end if
end if
```

The "Halberstam Function" of an integer value is defined as follows:

> Given a positive integer value, generate the sequence
> If it is even, half it.
> Otherwise multiply by 3 and add 1.

The Halberstam value of an integer value "n" is the number of times this operation needs to be repeated before the value 1 (one) is reached. It is guaranteed that you will eventually reach the value 1. If we start with the value 3, for example, the sequence goes 10, 5, 16, 8, 4, 2, 1; 7 steps. If we start with the value 7, the sequence goes 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1; 16 steps.

<u>a(m,n)</u>

Requires: Two numbers m, n

Returns: ???

```
if m ==0 then
    return n + 1
else
    if n == 0 then
        return a(m – 1, 1)
    else
        return a(m – 1, a(m, n – 1))
    end if
end if
```

Calculate a(1, 2).

a(1,2) = a(0, a(1, 1))
= a(0, a(0, a(1, 0)))
= a(0, a(0, a(0, 1)))
= a(0, a(0, 2))
= a(0, 3)
= 4

This algorithm is known as the Ackermann function.

Can you guess roughly how big the numbers would be if you were to calculate a(2, 2) a(3, 3) a (4, 4) ?

The Ackermann function

Values of $A(m, n)$

$m \backslash n$	0	1	2	3	4	[redacted]
0	1	2	3	4	5	
1	2	3	4	5	6	
2	3	5	7	9	11	
3	5	13	29	61	125	
4	13	65533	$2^{65536}-3$	$2^{2^{65536}}-3$	$2^{2^{2^{65536}}}-3$	

(Wikipedia)

Home Work Question :

An abundant number is a natural number whose distinct proper factors have a sum exceeding that number.
Thus, 12 is abundant because $1+2+3+4+6 > 12$.

Write an algorithm **isabundant** which tests whether or not a number is abundant.

Note : You may /may not use helper algorithm to complete the task.