Seminar 1 - Recursive algorithms and the recursive mathematical model

1. Check if a number is lucky. A number is considered lucky if it contains only the digits 4 and 7 (ex. 47, 74, 77, 747, etc. are lucky numbers).

Lucky(n) = [ false, n%10 != 4 and n%10!=7,

True, n == 4 or n == 7,

Lucky(n/10), otherwise

Def Lucky(n) :

If n%10!=4 and n%10!=7:

Return False

Elif n==4 or n==7:

Return True

Else:

Return Lucky(n//10)

1. Compute the sum of the proper divisors for a number n. For example, if n = 20, we need to compute: 2+4+5+10 = 21.

Sum\_div(I, n) = { 0, I > n/2,

I + sum\_div(I+1, n) , If n % I == 0

sum\_div(I+1, n) , If n % I != 0}

Def Sum\_div(I, n):

If I > n // 2:

Return 0

Elif n % I == 0:

Return I + Sum\_div(I + 1, n)

Else:

Return Sum\_div(I + 1, n)

**Sum\_div(2, 20)**

Sum\_div(1, 20)

Sum\_div(3, 20)

Sum(n) = sum\_div(2, n)

**List**

* A sequence of elements: l1l2l3...ln

|  |  |
| --- | --- |
| It is possible | It is not possible\* |
| * To compare the length of the list to a CONSTANT number   + N = 0 (is the list empty)   + N = 1 (does the list contain 1 element)   + N = 10   + N < 2 (does the list contain at most 1 element)   + N > 3 * To access elements from the beginning of the list   + L1, l2, l10 * We can divide the list by removing a constant number of elements from the beginning   + L2...ln   + L3...ln   + L5...ln * We can only add an element to the beginning of the list (and a constant number of elements)   + E U l1...ln   + E1 U E2 U L1...ln   + L1 U E U L2....Ln | * To access the length of a list   + If you have two lists: l1..ln and m1...mk, k = n, k > n   + If you have a list l1...ln and a position k, k > n * To access elements from the middle of end of the list   + Li, ln * To divide the list in the middle   + Li....Ln * To add an element at the end of the list or to concatenate two lists   + L1...ln U e   + L1...ln U m1...mk * To change an existing list   + L1 = e |

\*it is possible if we write a separate function for them

1. Compute the product of the even numbers from a list

Product(l1l2...ln)={1, if the list is empty

Product(l2...ln)\*l1,if l1%2==0

Product(l2...ln),ifl1%2==1}

What if I want to return 0 if the list does not have even numbers?

* + Write another function
    - If product returns 1, return 0, otherwise return product
  + Use an extra parameter which is set to true when we find an even number. When we get to the empty list, if the param is true we return 1, otherwise return 0.

Assume that we have the following operations for the list:

* IsEmpty(list) - returns true or false depending on whether the list is empty or not
* FirstElem(list) - returns the first element of the list
* Sublist(list) - returns the list without the first element
* CreateEmpty() - creates and returns an empty list
* AddFirst(elem, list) - returns the list with the element elem added to the beginning

Def Product(list):

If isEmpty(list):

Return 1

If firstElem(list)%2==0:

Return firstElem(list)\*Product(Sublist(list))

Else:

Return Product(Sublist(list))

4.a. Add a value e on position m (m >= 1) in a list (indexing starts at 1). For example: [1,2,3,4,5,6], e = 11, m = 4 => [1, 2, 3, 11, 4, 5, 6].

AddElement(4, 1, 11, [1,2,3,4,5,6]) = [1,2,3,11,4,5,6]

1 U AddElement(4, 2, 11, [2,3,4,5,6]) = 1 U [2,3,11,4,5,6] => [1,2,3,11,4,5,6]

2 U AddElement(4, 3, 11, [3,4,5,6]) = 2 U [3, 11, 4, 5 ,6] => [2,3,11,4,5,6]

3U AddElement(4, 4, 11, [4,5,6]) = 3 U [11, 4, 5,6] => [3,11,4,5,6]

[11, 4, 5, 6]

AddElement(m, I, value ,l1,l2...ln)={ 0, if list is empty

L1 U AddElement(m, I+1, value ,l2,l3..ln), i!=m

value U l1, l2..ln, i==m

4.b: Add a value *e* in a list from *m* to *m(m >=2)*.

Eg: for the list: [1,2,3,4,5,6,7,8,9,10], *e* = 11 and *m* = 4, the result is [1,2,3,11,4,5,6,11,7,8,9,11,10].