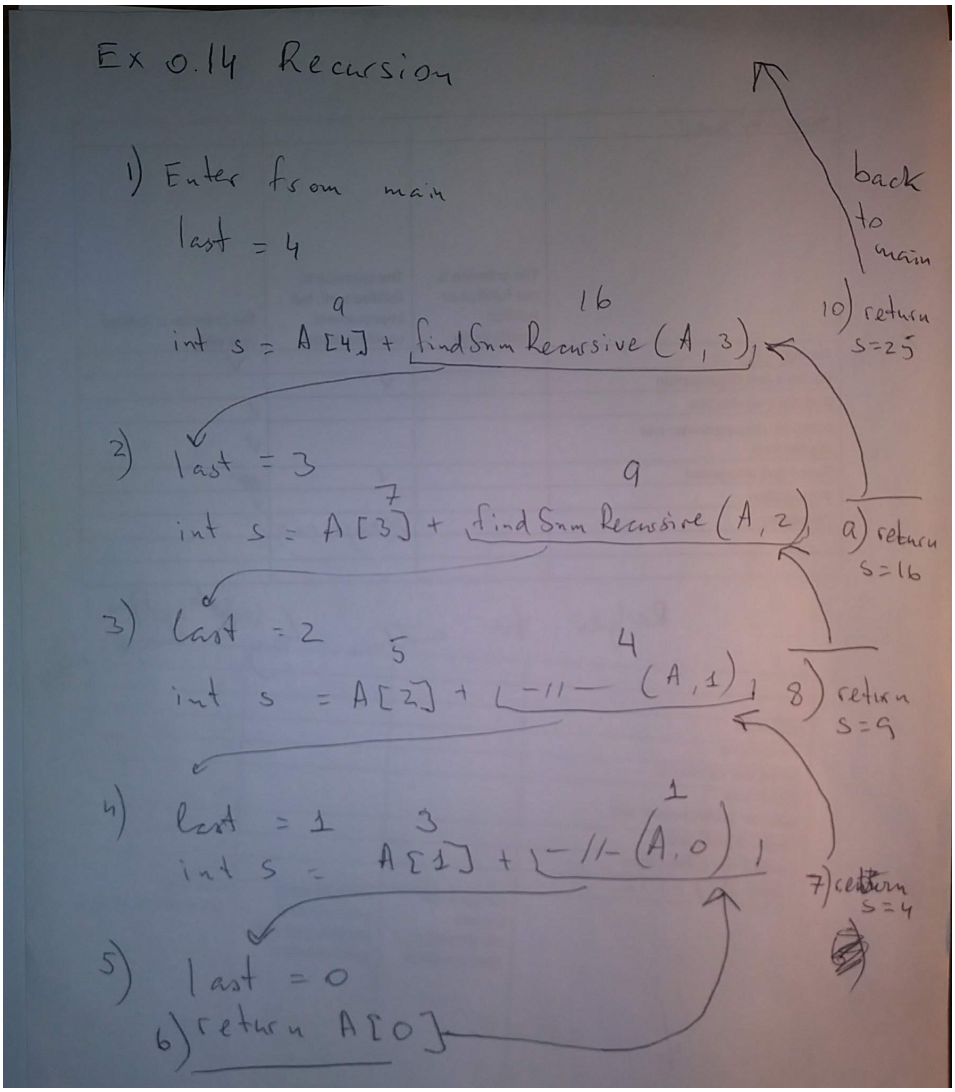
Dmitri Stanchevici
Unit 3
Module 0

Ex. 0.14

makeArray (int size) with size=5

i	**A[i]**
0	1
1	3
2	5
3	7
4	9

Sum of first two elements: 4
Sum of first three elements: 9
Sum of first four elements: 16
Sum of five elements: 25


Ex 0.14 Recursion
1) Enter from main
last = 4
back to main
9
16
int s = A[4] + findSum Recursive (A, 3);
10) return s=25
3) last = 3
7
9
int s = A[3] + findSum Recursive (A, 2);
9) return s=16
3) last = 2
5
4
int s = A[2] + -11- (A, 1);
8) return s=9
4) last = 1
3
1
int s = A[1] + -11- (A, 0);
7) return s=4
5) last = 0
6) return A[0]

Ex. 0.16

Searching for value N-1:

N=1,000
Found=true Time taken: 0
Found2=true Time taken: 0

N=10,000
Found=true Time taken: 0
Found2=true Time taken: 0

N=100,000
Found=true Time taken: 2
Found2=true Time taken: 0

When looking for value N-1, the first difference occurs at N=100,000.

Ex. 0.17
N=20
Searched value=5

Iteration 1:
start=0 end=19
mid=9

Iteration 2:
start=0 end=8
mid=4

Iteration 3:
start=5 end=8
mid=6

Iteration 4:
start=5 end=5
mid=5

On every iteration after the first and before the last, the searched range is halved by readjusting the values of start or end, depending of whether the searched value may appear before or after mid.

Ex. 0.19

Search value = 16

Before while loop

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
start																			end

while (start <= end)

Iteration 1:
mid = 10

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
start									mid										end

16 > 10 -> start = mid+1

Iteration 2:
mid = 11

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
										start				mid					end

16 > 15 -> start = mid+1

Iteration 3:
mid = 18

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
															start		mid		end

16 < 18 -> end = mid-1

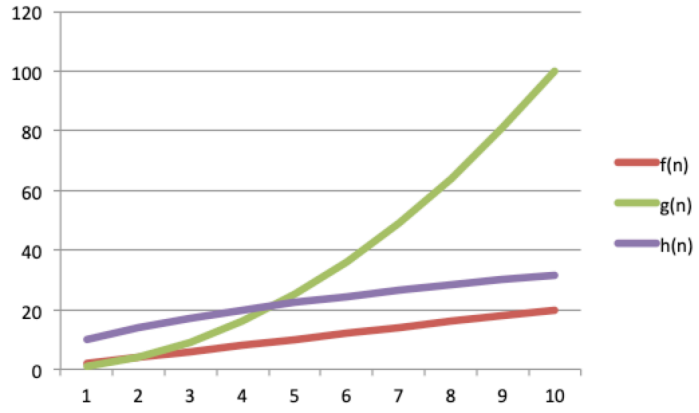
Iteration 4:
mid=16

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
															start	end			

Search value == found

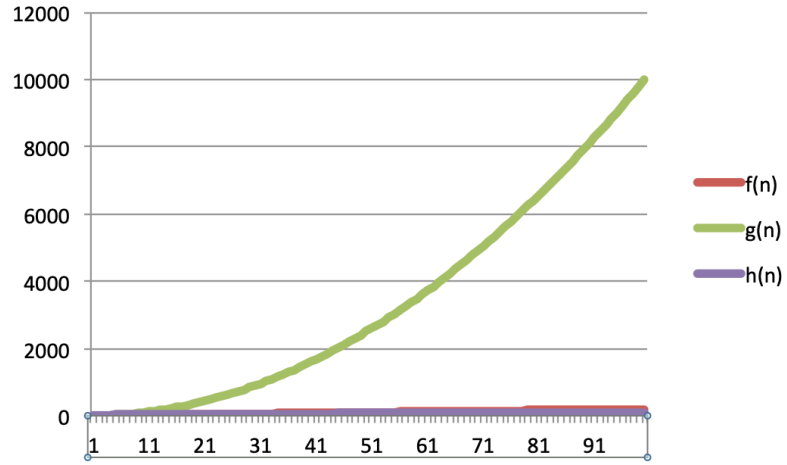
Ex. 0.21

For n = 1, 2, 3,...10



Ex. 0.21

For n=10, 20, 30,...100



When nLow=10, nHigh=100, and nStep=10, the third algorithm becomes better than the first at **value 30.**

To refine the analysis, I changed nStep to 1. When nLow=10, nHigh=100, and nStep=1, the third algorithm becomes better than the first at **value 26.**

Ex. 0.24

Because n is the exponent of the e number in the h function, every increase in n after 2 or 3 results in a sharp increase in the output of h. Naturally, when the exponents are in the tens (10, 20, ...), this increase is even more dramatic. 2.718 to the power of 10 is over 22 thousand, the power of 20 is almost 500 million, the power of 30 is between 10 and 11 trillion, and so on.

Ex. 0.28

Number of divides by 2