CS 161 - HW1

Name: Yu-Chen Lin
UID: 705315195
E-mail: ericlin8545@cs.ucla.edu

Question 1.

In this question, we are asked to conduct a function that can show the element in the Padovan sequence given the index. In this question, I used the recursion to achieve the effect of finding the element. And here are some execution results:

(PAD 0) => 1 (PAD 4) => 2 (PAD 8) => 7
(PAD 1) => 1 (PAD 5) => 3 (PAD 9) => 9
(PAD 2) => 1 (PAD 6) => 4 (PAD 10) => 12
(PAD 3) => 2 (PAD 7) => 5 (PAD 11) => 16

When I try some larger input values such as 100, I find that the execution time will be extremely long. That's because the execution time grows exponentially when the input value N is increased since every PAD function stack needs to conduct the other two PAD functions. Thus, it's pretty time-consuming when the input value N is too large.

Question 2.

In here, we are asked to record the number of additions that are used when executing the function PAD with different input value N. And here are some execution results of the function SUMS:

(SUMS 0) => 0 (SUMS 4) => 1 (SUMS 8) => 6
(SUMS 1) => 0 (SUMS 5) => 2 (SUMS 9) => 8
(SUMS 2) => 0 (SUMS 6) => 3 (SUMS 10) => 11
(SUMS 3) => 1 (SUMS 7) => 4 (SUMS 11) => 15

We can see that the result of the function SUMS is exactly one less than the results of the function PAD. That's because the base case in the function PAD is 1 but in SUMS it's 0. And for the number of additions in PAD, it's related to the result of PAD. The result of PAD means there are 1s of exactly that number of been summed up. For PAD(N) number of 1s to be summed up, there should be PAD(N) - 1 number of additions.

Question 3.

In this question, I use a similar way as the programming practice "*compute list length (deep)*" in lecture 2. So I processed the TREE with a DFS way in a recursion form. For the base cases: If there is an empty element, I just returned an empty list, and if there is only an atom element, then I return a question symbol. And in other cases, I divided the TREE as the first item and the rest to conduct the recursion. And here are the execution results:

(ANON '42) => ?
(ANON 'FOO) => ?

(ANON '(((L E) F) T)) => (((? ?) ?) ?)
(ANON '(5 FOO 3.1 -0.2)) => (? ? ? ?)
(ANON '(1 (FOO 3.1) -0.2)) => (? (? ?) ?)
(ANON '(((1 2) (FOO 3.1)) (BAR -0.2))) => (((? ?) (? ?)) (? ?))
(ANON '(R (I (G (H T))))) => (? (? (? (? ?))))
(ANON nil) => nil
(ANON ()) => nil
(ANON '(nil nil nil (? ?))) => (nil nil nil (? ?))