Eric Roberts
CS 106B

Handout #24A
February 9, 2015

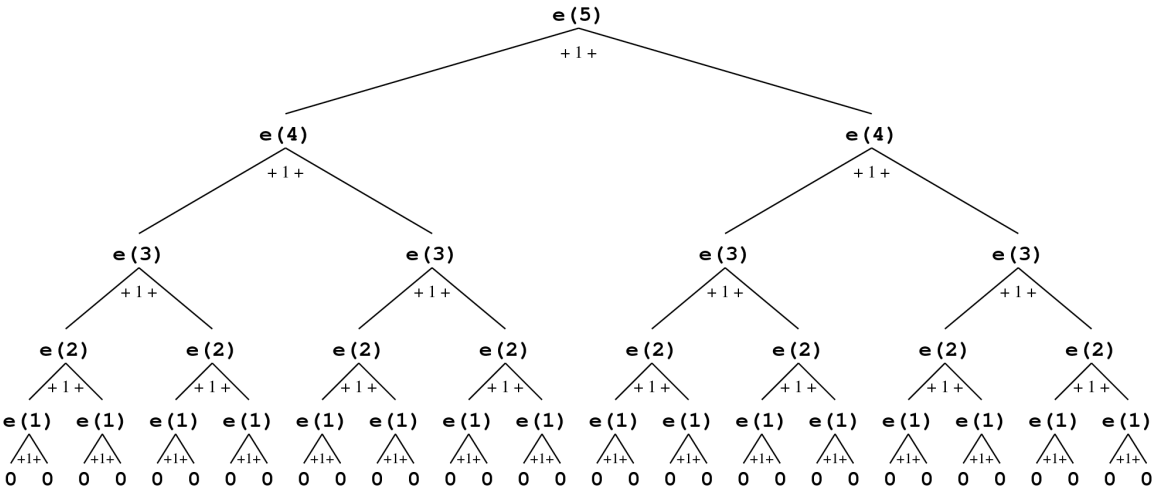
Solutions to Problem Set #1

Problem 1: Computational complexity

1a.

i. If you think about what's happening in the **`enigma`** function, it should be clear that the function computes the number of moves required to solve the Tower of Hanoi problem. Thus, the value of **`enigma(5)`** is 31.

ii. To understand the complexity order of the computation, it helps to draw a tree of the computations involved, which (after abbreviating **`enigma`** to **`e`** to save space and replacing the bottom row with its value) looks like this for **`enigma(5)`**:



Each new level doubles the amount of work, so the total amount of work must be $O(2^N)$. Another way to obtain this same result is that the calculation of **`enigma(`***N***`)`** requires twice as many additions as the original Tower of Hanoi puzzle requires moves to solve the problem for *N* disks. If Tower of Hanoi is exponential, this function must be as well.

iii. Making the proposed change means that there is only one recursive call at each level, so the computational complexity is reduced to $O(N)$.

1b.

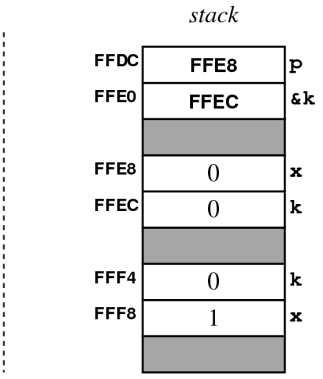
i. **`mystery("retest")`** → **`"street"`**

ii. The loop inside this function runs *N* times, once for each character in the argument. On each cycle, the string concatenation operation requires time proportional to the length of the **`result`** string, which is growing by one character. The number of operations is therefore $1 + 2 + 3 + \cdots + N$, which is $O(N^2)$.

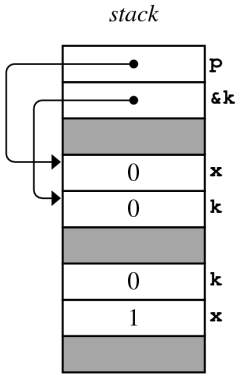
2. Heap-stack diagrams

2a.

explicit addresses

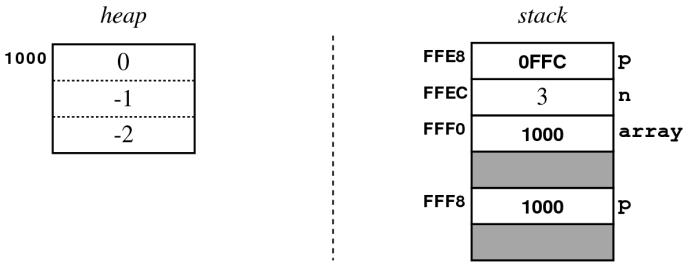


arrow representation

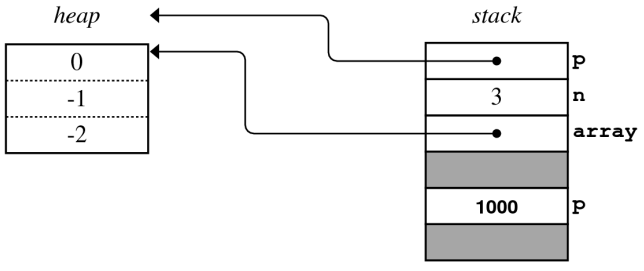


2b.

explicit addresses



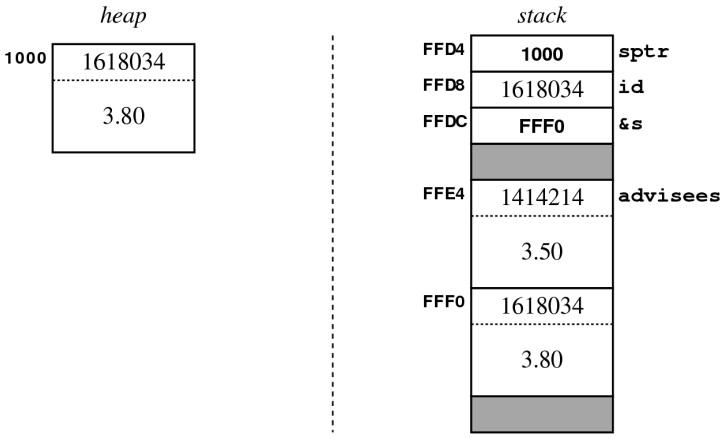
arrow representation



Note that the variable `p` ends up pointing outside the bounds of the array.

2c.

explicit addresses



arrow representation