Dylan Forbes

The Typo Problem

1 Problem

What is the asymptotic complexity, with respect to n, of the number of times the following code will print?

```
for (int i=0; i < n; i++) {
        for (int j=0; j < i; j++) {
                if (j & i == 0) {
                        System.out.println(i + " " + j);
                }
        }
 }
```

What makes this problem interesting is the appearance of the bitwise AND operator (&) in the loop. For those unfamiliar with the subject matter, the bitwise AND of two integers A and B—expressed as $A\&B$ in this context—gives the result when the binary representations of A and B are compared bit (binary digit) by bit, where each comparison gives a 0 unless both bits are 1. For example, 5&7 can be calculated as follows:

$$5_{ten}\&7_{ten} \quad = \quad 101_{two}\&111_{two} \quad = \quad \begin{array}{cccc} & 1 & 0 & 1 \\ \& & 1 & 1 & 1 \\ \hline & 1 & 0 & 1 \end{array} \quad ; \quad 101_{two} = 5_{ten}.$$

2 Solution

This paper proceeds as follows. First, I examine how the number of times the program prints within a single iteration of the inner loop—denoted by $N(i)$—increases with i. Then, after observing a clear pattern, I posit a formula for the sum $S(x)$ of the values of $N(i)$ over a range of indeces $2^x \le i < 2^{x+1}$. Then, I use that formula to derive a formula $T(n)$ for the sum of $N(i)$ where i spans 0 to n; that is, the number of times the outer loop will print for a given n.

To begin, it is helpful to make a table of some relevant values:

i_{ten}	i_{two}	$J(i) = \{j : 0 \le j < i \wedge j\&i = 0\}$	$N(i) = \lvert J(i)\rvert$
0	0		0
1	1	0	1
2	10	0,1	2
3	11	0	1
4	100	0,1,10,11	4
5	101	0,10	2
6	110	0,1	2
7	111	0	1
8	1000	0,1,10,11,100,101,110,111	8
9	1001	0,10,100,110	4
10	1010	0,1,100,101	4
11	1011	0,100	2
12	1100	0,1,10,11	4
13	1101	0,10	2
14	1110	0,1	2
15	1111	0	1
16	10000	(16 items)	16
⋮	⋮	⋮	⋮

n	$T(n) = \Sigma_{i=0}^{n-1} N(i)$
0	0
1	0
2	1
3	3
4	4
5	8
6	10
7	12
8	13
9	21
10	25
11	29
12	31
13	35
14	37
15	39
16	40
⋮	⋮

The left-hand table represents the inner loop: $J(i)$ is the set of integers j less than or equal to i such that $i\&j = 0$, and $N(i)$ is the number of items in $J(i)$, and thus the number of times the program will print within one iteration of the outer loop (with j spanning 0 to i). The right-hand table represents the outer loop: $T(n)$ is the sum of each $N(i)$ where i spans 0 to n, and is thus the number of times the program will print for the given value of n.

Hence, our goal is to determine how quickly $T(n)$ increases with n.

As mentioned previously, I will begin by observing a pattern in the values of $N(i)$. If we arrange the values horizontally, directly aligned with the corresponding indeces i, the pattern is clear to see:

i	1	2		4				8								$\cdots$
$N(i)$	1	2	1	4	2	2	1	8	4	4	2	4	2	2	1	$\cdots$

It appears to be a recursive pattern, wherein $N(1) = 1$, and, informally, each successive "block" of values of $N(i)$ is formed by taking two copies A and B of the previous block, doubling the values of A, and appending B to the end.

The remainder of this paper rests on the assumption that the just-described pattern holds true for all of N. I do not prove this claim; rather, I take it as a well-justified assumption—the purpose of this paper is not to rigorously prove what the result is, but merely to arrive at the result, for it is interesting enough in its own right.

A result of this pattern is that the sum of the values of one particular block is thrice the sum of the values in the preceding block., for each block consists of one regular copy and one doubled copy of the previous block. That is, if we define

$$S(x) = \sum_{i=2^x}^{2^{x+1}-1} N(i)$$

as the sum of the values in a particular block of values of N, then we must have that

$$S(x) = \begin{cases} 1 & \text{if } x = 0 \\ 3S(x-1) & \text{if } x > 0. \end{cases}$$

And this clearly simplifies to

$$S(x) = 3^x.$$

Now, because $S(x)$ gives the sum of one particular block, and because each block starts with an index n that is a power of two, we can construct a formula that gives values of $T(n)$ just for $n = 2^x$, simply by summing up the first x blocks. For example, to calculate $T(8)$, we would add the first 3 blocks: $S(0) + S(1) + S(2) = \Sigma\{1\} + \Sigma\{2,1\} + \Sigma\{4,2,2,1\} = 1 + 3 + 9 = 13$, which the table indicates is the correct value. Hence, we can write

$$T(2^x) = \sum_{k=0}^{x-1} 3^k.$$

Then, for $n = 2^x$, we have

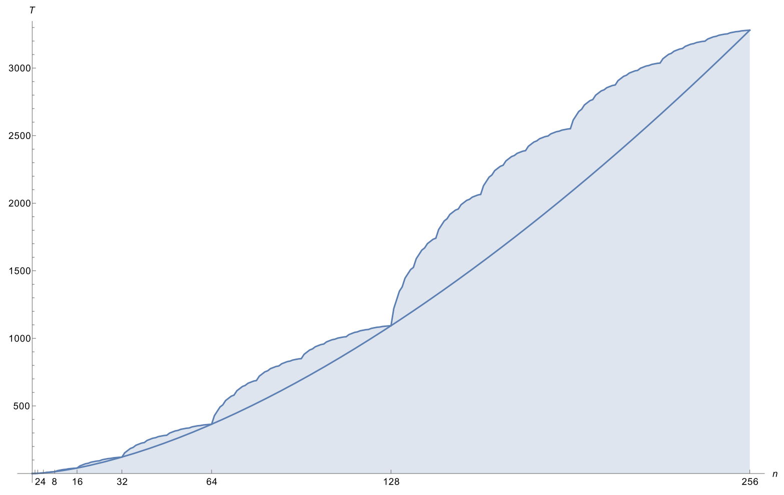
$$T(n) = \sum_{k=0}^{\log_2(n)-1} 3^k.$$

which simplifies to

$$T(n) = \frac{1}{2}(3^{\log_2 n} - 1).$$

To see that this is correct, we can construct another table of values, letting $T(n)$ be the original values generated iteratively from the algorithm itself (which can be seen in the original table), and letting $T?(n)$ be the values given by the previously derived formula for $T(n)$, which we only showed works for $n = 2^x$.

n	$T(n)$	$T?(n)$
0	0	−0.5
1	0	0
2	1	1
3	3	2.35
4	4	4
5	8	5.91
6	10	8.06
7	12	10.42
8	13	13
9	21	15.77
10	25	18.73
11	29	21.86
12	31	25.17
13	35	28.64
14	37	32.27
15	39	36.06
16	40	40
⋮	⋮	⋮



Now, although $T?(n) = T(n)$ only for $n = 2^x$, this still means both functions must have the same asymptotic complexity—as can be seen in the above plot, $T(n)$ can never "break away" from $T?(n)$. Thus,

$$T(n) \in O(\frac{1}{2}(3^{\log_2 n} - 1)).$$

Then, finally, because

$$\lim_{n\to\infty} \frac{n + \frac{1}{2}(3^{\log_2 n} - 1)}{3^{\log_2 n}} = \frac{1}{2} \cdot \lim_{n\to\infty} \frac{3^{\log_2 n}}{3^{\log_2 n}} = \frac{1}{2}$$

exists and is finite,

$$T(n) \in O(3^{\log_2 n}),$$

Therefore, the asymptotic complexity, with respect to n, of the number of times the program will print is $O(3^{\log_2 n})$, which is between $O(n \log n)$ and $O(n^2)$.