CS146 - Homework 2

Frank Mock

February 16, 2014

2.1

Order is from the slowest growth to the greatest

- 37
- $\sqrt{N}$
- $2/N$
- N
- $Nlog(log(N))$
- $Nlog(N)$
- $Nlog(N^2)$
- $Nlog^2N$
- $N^{1.5}$
- N^2
- N^2logN
- N^3
- $2^{N/2}$
- 2^N

2.4

Prove that for any constant, k, $log^k n = o(n)$.

$log^k n = (logn)^k$ which is an exponential expression.

A function is polylogarithmically bounded is $f(n) = O(log^k n)$ for some constant k.

The growth rates of polynomials and exponentials can be related as such: $\forall$ real constants a and b such that $a > 1$ $\lim_{n\to\infty} \frac{n^b}{a^n} = 0$

This means that any exponential function with a base greater than one grows faster than any polynomial function. $n^b = o(a^n)$.

Using the same logic, we can also relate the growth of polynomials and polylagarithms by substituting $logn$ for n and 2^a for a in the limit above.

$$\lim_{n\to\infty} \frac{log^b n}{(2^a)^{logn}} = \lim_{n\to\infty} \frac{log^b n}{n^a} = 0$$

The above limit shows that $log^b n = o(n^a)$ for any constant $a > 0$ and that the polynomial function grows faster than any polylogarithmic function. This is what was to be proved.

(Introduction To Algorithms by Cormen pages 55 - 57 was referenced to help complete this proof.)

2.7

1.a

O(n) because, 1 for the assignment of line 1, 1 + n + n for the for-loop, and n for line 3 which gives 3n + 2

1.b

$n = 10$ 1816 ns

$n = 100$ 3283 ns

$n = 1000$ 18159 ns

1.c

The actual running times don't exactly match, but considering other factors pertaining to my machine they are reasonable.

2.a

O(n^2) because, 1 for the assignment of line one, 1 + n + n for line 2, 1 + n + n for line 3, and n for line 4 which gives $4n^2 + n + 3$

2.b

$n = 10$ 3561 ns

$n = 100$ 252476 ns

$n = 1000$ 11127741 ns

2.c

The actual run-time results are worse than my estimate of O(n^2), but it's probably due to my machine

Exercise 2.7 continued from previous page.

3.a

O(n^3) because, 1 for assignment of line one, 1 + n + n for line two, 1 + n^2 +n for line three, and n for line four which gives about $2n^3 + 3n^2 + n + 3$

3.b

$n = 10$ 20883 ns

$n = 100$ 10460966 ns

$n = 1000$ 977206042 ns

3.c

When I compare these run-times to the O(n^2) run-times of the previous code analysis, these seem to be follow the estimated run-time

4.a

O(n^2) because, 1 for the assignment of line one, 1 + n + n for line two, 1 + n + n for line three, and n for line four. Which gives approximately $4n^2 + n + 3$

4.b

$n = 10$ 3352 ns

$n = 100$ 158749 ns

$n = 1000$ 4228959 ns

4.c

When I compare these to code item 2 which I said was also O(n^2), the numbers look pretty close.

5.a

O(n^3) My computer had a hard time computing this code so I expect it to be at least cubic.

5.b

$n = 10$ 230546 ns

$n = 100$ 97647736 ns

$n = 1000$ *My computer could not compute for this value of n. I aborted execution because it never ended*

5.c

The run-time is comparable to the Big-Oh estimate

6.a

O(n^3) because, 1 for line one, 1 + n + n for line two, 1 + n^2 + n for line three, $\frac{n}{2}$ for line four, 1 + n + n for line five and n for line six. This gives approximately $4n^3 + 7n^2 + 2n + 5$

6.b

$n = 10$ 29054 ns

$n = 100$ 19675406 ns

$n = 1000$ *My computer could not compute for this value of n. I aborted execution because it never ended*

6.c

The run-time is comparable to the Big-Oh estimate.

2.11

a. linear $\frac{.5ms}{100} = \frac{n}{500}$ $\quad 500(.5ms) = 100n \quad n = 2.5ms$

b. O(NlogN) $\frac{.5ms}{100} = \frac{nlogn}{500} \quad 500(.5) = 100n \quad \frac{5}{2} = nlogn \quad 2^{\frac{5}{2}} = 2^{nlogn}$
$2^{\frac{5}{2}} = n^{nlog2} \quad 2^{\frac{5}{2}} = n^n$

c. quadratic: For input size of 1 it takes $\frac{.5ms}{100} = \frac{n^2ms}{1} \quad \sqrt{\frac{.5}{100}} = n$
then for input size of 500 it takes, $500(\sqrt{\frac{.5}{100}}) \approx 35.35ms$

d. cubic: For input size of 1 $\frac{.5ms}{100} = \frac{n^3ms}{1} \quad \sqrt[3]{\frac{.5}{100}} = n$ then for an input size of 500 it takes $500(\sqrt[3]{\frac{.5}{100}}) \approx 85.50ms$

2.15

Since the array is sorted $A_1 < A_2 < A_3 < ... < A_n$ I would use some sort of divide and conquer routine which would have a run-time of O($logN$)

```
boolean findInteger(int[] A, int i)
{
    int begin = 0;
    int end = n - 1;

    do
    {
       int x = (begin + end)/2;  //keep dividing in half
       if(A[x] == i)
           return true;
       else if(i < A[x])
           end = x - 1;
       else
           begin = x + 1;
    }while(begin <= end)
    return false;
}
```

2.25

a. Program A because, $150Nlog_2N$ can be re-written as $Nlog_2N^{150}$ which is referred to as a polylogarithmic function. I showed in homework question 2.4 any polynomial grows faster than any polylogarithmic function as n gets very large.

b. Program B is better for small values of n and is the program I would use if I knew for certain that the input data would never be very large.

   The chart below shows that for large values of N Program B is better and for small values of N Program A is better.

N	N^2	$150Nlog_2N$
20	400	12,966
60	3,600	53,162
80	6,400	75,863
150	22,500	162,648
1,000	1,000,000	1,494,868
10,000	100,000,000	19,931,569
100,000	10,000,000,000	249,144,607

c. Program A. Please refer to chart above.

d. No, it is not possible that program B will run better on all possible inputs.