Assignment 2, Data Structures

Jay R Bolton

October 25, 2011

2.1, 2.2, 2.6, 2.10, 2.14, 2.22

2.1

$$2/N, 37, \sqrt{N}, N, NloglogN, NlogN, Nlog(N^2), Nlog^2N, N^{1.5}, N^2, N^2logN, 2^{N/2}, 2^N$$

2.2 A is true.

2.6 I have found the exact complexities. I first turned each problem into a series of nested summations. Then I reduced them starting from the inside. I used the rules in the book for sums of squares and cubes, and had to google the sum of fourths.

$\Theta(N)$

(1)

```
int one(int n) {
  int i, sum=0;
  for(i = 0; i < n; i++) {
    sum++;
  }
  return sum;
}
```

(2) $\Theta(N^2)$

```
int two(int n) {
  int i, j, sum=0;
  for (i=0; i<n; i++)
    for (j=0; j<n; j++)
      sum++;
  return sum;
}
```

(3) $\Theta(N^3)$

```
int three(int n) {
  int i, j, sum=0;
  for (i=0; i<n; i++)
    for (j=0; j<n*n; j++)
      sum++;
  return sum;
}
```

(4) $\Theta\left(\frac{N(N-1)}{2}\right)$

```
int four(int n) {
  int i, j, sum=0;
  for (i=0; i<n; i++)
    for (j=0; j<i; j++)
      sum++;
  return sum;
}
```

(5) $\Theta\left(\frac{6(n-1)^5+15(n-1)^4+10(n-1)^3-n+1-10n^3+15n^2-5n}{60}\right)$

This is derived from the sum: $\sum_i^{n-1}\left(\sum_j^{i^2-1}\left(\sum_k^{j-1}1\right)\right)$

```
int five(int n) {
  int i, j, k, sum=0;
  for (i=0; i<n; i++)
    for (j=0; j<i*i; j++)
      for (k=0; k<j; k++)
        sum++;
  return sum;
}
```

(6) $\Theta\left(\frac{(3i^4-10i^3+9i^2-2i)}{24}\right)$

This was (painstakingly) calculated from the sum: $\sum_i^{n-1}\left(\sum_j^{i-1}\left(\sum_k^{i*j-1}1\right)\right)$

```
int six(int n) {
  int i, j, k, sum=0;
  for(i=1; i<n; i++)
    for(j=1; j<i*i; j++)
      if(j%i == 0)
        for(k=0; k<j; k++)
          sum++;
  return sum;
}
```

2.10 **a.** $a_i = [2, 1, 8, 4], x = 3, f(x) = 4x + 8x + x + 2$

```
poly = 0
poly = 3 * 0 + 4 = 4
poly = 3 * 4 + 8 = 20
poly = 3 * 20 + 1 = 61
poly = 3 * 61 + 2
= 185
```

b. It continually factors out x from left to right:
$f(x) = 4x + 8x + x + 2 = ((4x + 8)x + 1)x + 2$

b. $\Theta(N)$
(Counting the number of multiplications as the measurement of running time)

2.14 If we measure our running time by the number of times we cross out a number, then we do:

$f(n) = n/2 + n/3 + n/5 + n/7...n/q$

Where q is the last prime less than or equal to the square root of n. Factoring out n, we get:

$f(n) = n(1/2 + 1/3 + 1/5 + 1/7...1/q)$

In Euler's proof that the sum of the harmonic primes diverges (I looked it up on wikipedia), it states that the asymptotic upper bound of the harmonic primes up to n is loglogn. Thus, it seems like we can get a fairly tight upper bound with:

$f(n)\epsilon\mathcal{O}(nloglogn)$

2.22 No. In the case where we are searching for an element that is greater than the maximum element of the list, we will loop endlessly on the second to last element.