CS 315 Data Structures Fall 2008

Lab # 2, Sept 5, 2008

This lab will weigh *3 points*.

The goal of this lab is to understand the basic technique of algorithm analysis – by mathematical analysis as well as by experimental approach.

First you should review the summation formula for arithmetic and geometric series.

Arithmetic series: If $A_1 + A_2 + \ldots + A_n$ forms an arithmetic series (i.e., the difference between consecutive terms is a constant), then:

$A_1 + A_2 + \ldots + A_n = n\,A$ where A = the average of the first and the last terms, i.e., $A = (A_1 + A_n)/2$

Geometric series: has the property that the ratio of successive terms is a constant. (Example: 1 + ½ + ¼ + 1/8 + …)

The sum of the n terms of a geometric series is:

$a + ar + ar^2 + \ldots + ar^{n-1} = (a\,r^n - 1) / (r - 1)$

O notation: O notation is used to approximate a function in terms of some simple, standard functions. These approximations usually hold only when n (the argument) is large. The simple, standard functions we will use are n^k (polynomial function), log n (logarithm) and c^n (exponential function). Of the polynomial functions, the most important cases are k = 0 (constant), k = 1 (linear) and k = 2 (quadratic). Most of the estimates will be one of the terms or some product of these terms such as n log n. Basic rules for simplification are as follows:

1) $c^n >> n^k >> \log n >> d$ (for any $c > 1$ and any $k > 0$, and for any d) Here, c, k, d etc. are constants.

2) $n^K >> n^L$ if $K > L$

3) approximate A + B by A if A >> B. So a sum involving many terms can be approximated by the fastest growing term.

4) Finally in a term like $5\ n^3$, you can drop a constant coefficient and approximate it as n^3. The final resulting expression gives an order of magnitude estimate.

We use O notation to express this fact.

Example: Consider the expression $(3n + 5 \log n + 6)\ (2 \log n + 23) + 4n$. We will check that this expression is $O(n \log n)$.

The sum $3n + 5 \log n + 6 = O(n)$ since n is the biggest term $2 \log n + 23 = O(\log n)$ so the product is $O(n \log n)$.

Analysis of iterative algorithms

Problems:

For each of the following algorithms, determine the number of operations of the specified type. You are to answer these questions without using the computer.

1)
```
for (int j = 1; j < 100; j++) {
    B[j] = 0;
    for (int k = j+ 1; k < 100; ++k)
      B[j] += B[k];
   }
```

How many times is the instruction **`B[j] += B[k]`** executed?

2)
```
for (int i = 0; i < 30; ++i)
    for (int j = 0; j < 40; ++j)
       for (int k = 0; k < 20; ++k)
          C[i][j] = C[i][j] + A[i][k]*B[k][j];
```

How many multiplications are performed by the above code segment?

3)
```
A[0] = 1; A[1] = 1;
   for (int j = 2; j < 1000; ++j)
    if (j % 2 == 0)
       A[j] = A[j-1]+3*A[j-2];
    else
       A[j] = A[j-1]+A[j-2];
```

What is the total number of operations performed by the above program? (Include all the operations such as assignments, comparisons, additions and multiplications.)

For the following problems, first determine the number of operations of the specified type as a function of n, the input size. Then express your answer using O notation.

4) Shown below is a function that takes as input a vector of n integers and determines the maximum and the minimum numbers in the array. Determine the number of comparisons (between keys) performed by the program as a function of n in the best-case and in the worst-case. In the code, swap(x,y) swaps the keys x and y. (Assume that n is even.)

```
void findminmax (vector<int> A, int& min, int& max) {
  int n = A.size();
  for (int j = 0; j < n-1; j = j+2)
     if (A[j] > A[j+1]) swap(A[j], A[j+1]);
  min = A[0]; max = A[1];
  for (int j = 0; j < n-1; j = j+2) {
    if (A[j] < min) min = A[j];
    if (A[j+1] > max) max = A[j+1];
  }
 }
}
```

5) In the lecture, we presented the inner loop of insertion sorting. The complete program for insertion sorting is as follows:

```
for (j = 1; j < n; ++j) {
      temp = A[j];
      for (k = j – 1; k >= 0 && A[k] > temp; --k)
        A[k+1] = A[k];
      A[k+1] = temp;
     }
```

Estimate the number of comparisons performed by the above algorithm in the worst-case and express your answer using O notation. Do the same thing for the best case. Describe the best and the worst-case inputs.

Recursive program analysis

To analyze a recursive function, we should create recurrence equation. Here is an example:

```
int f(int[] A, int n) {
if (n == 1) return A[0];
else return A[0] + f(A, n – 1);
}
```

To determine the number of additions performed by the above function, let s(n) denote this number. Then, we see that s(n) can be given by the following formula:
s(n) = 0 if n = 1
        1 + s(n – 1) else.

Now we can solve for s(n) by computing the values of small n:

s(1) = 0, s(2) = 1, s(3) = 2, … Thus, we can guess that s(n) = n – 1. It is easy to prove this guess by induction on n.

6) Recall the recursive algorithm for computing x that you wrote in lab # 1:

```
int exp (int x, int n) {
  if (n == 0) return 1; else
  if (n % 2 == 0)
   {
     int temp =  exp(x, n/2);
     return temp * temp;
   }
 else
     return x * exp(x, n – 1);
}
```

Compute the number of multiplications performed by the above function for n = 100, 200, 300, …, 1000 and plot a graph of this data. (x – axis for n, and y-axis for the number of multiplications.)

7) Let t(n) be the number of multiplications performed by the above algorithm. (Note that the number of multiplications performed depends only on n, not on x.)

Write a recurrence formula for t(n). (The formula will express t(n) in terms of t(n – 1) or t(n/2) depending on whether n is odd or even.)

What should be submitted and when?

The solutions to the problems (1) to (6) are due at the end of the lab today. The plot can be done using excel (or a similar tool). Typeset your solutions to all the problems electronically and mail it to cs351submission@gmail.com.) Please make sure to include your name in the submission.