Computer Programming II (TA C252, Lect 11)

Murali P

Today’s Agenda

- **Efficiency & Complexity**
  - Issues
  - Order of Complexity

What is Efficiency ?

- Efficiency – Design & Implementation issue.
- Resources
  - CPU Time
  - Storage
- Resource Usage
  - Measured proportional to (input) size

Design Level Measurement

- Machine-independent measurement needed
  - Often referred to as “algorithmic complexity”
  - Individual statement considered as “unit” time
    - Not applicable for function calls and loops
  - Individual variable considered as “unit” storage
    - Not applicable for arrays (or other collections)

Complexity Example [1]

- Example 1 (Y and Z are input)

  X = 3 * Z;

  X = Y + X;

  // 2 units of time and 1 unit of storage

Complexity Example [2]

- Example 2 (a and N are input)

```
j = 0;
while (j < N) do
   a[j] = a[j] * a[j];
    b[j] = a[j] + j;
    j = j + 1;
endwhile;
// 3N + 1 units of time and N+1 units of
   storage
```

Order of complexity [1]

- Example 1 (Y and Z are input)

```
X = 3 * Z;
X = Y + X;
// Constant Unit of time and Constant Unit of storage
```

Order of complexity [2]

- Example 2 (a and N are input)

```
j = 0;
while (j < N) do
   a[j] = a[j] * a[j];
    b[j] = a[j] + j;
    j = j + 1;
endwhile;
// time units prop. to N and storage prop. to N
```

Order of complexity [3]

- Example 2 (a and N are input)

```
j = 0;
while (j < N) do
    k = 0;
    while (k < N) do
          a[k] = a[j] + a[k];
         k = k + 1;
     endwhile;
    b[j] = a[j] + j;
    j = j + 1;
endwhile;
```

// time prop. to N^2 and storage prop. to N

Order Notation

- Purpose
  - Capture proportionality
  - Machine independent measurement
  - Asymptotic growth (I.e. large values of input size N)

Motivation for Order Notation

$\log_2 N$	N	N^2	N^3	2^N
1	2	4	8	4
3.3	10	10^2	10^3	$>10^3$
6.6	100	10^4	10^6	$>10^{25}$
9.9	1000	10^6	10^9	$>10^{250}$
13.2	10000	10^8	10^{12}	$>10^{2500}$

Motivation for Order Notation

- Examples
  - $100 * \log_2 N < N$ for $N > 1000$
  - $70 * N + 3000 < N^2$ for $N > 100$
  - $10^5 * N^2 + 10^6 * N < 2^N$ for $N > 26$

Motivation for Order Notation

$\log_2 N/10$	$N/10$	$N^2/10$	$N^3/10$	$2^N/10$
0.1	0.2	0.4	0.8	0.4
0.33	1	10	10^2	$>10^2$
0.66	10	10^3	10^5	$>10^{24}$
0.99	100	10^5	10^8	$>10^{249}$
1.32	1000	10^7	10^{11}	$>10^{2499}$

Speed factor of 10

Motivation for Order Notation

$\log_2 N/10^4$	$N/10^4$	$N^2/10^4$	$N^3/10^4$	$2^N/10^4$
0.0001	0.0002	0.0004	0.0008	0.0004
0.00033	0.001	0.01	0.1	>0.1
0.00066	0.01	1	10^2	$>10^{21}$
0.00099	0.1	10^2	10^5	$>10^{246}$
0.00132	1	10^4	10^8	$>10^{2496}$

Speed factor of 10^4

Order Notation

- Asymptotic Complexity

  g(n) is O(f(n))   if there is a constant c

  such that

  $g(n) <= c(f(n))$

i.e. $\lim_{n \to \infty} (g(n) / f(n)) = c$ and $c <> 0$

Order Notation

- Examples

  $17*N + 5$ is $O(N)$

  $5*N^3 + 10*N^2 + 3$ is $O(N^3)$

  $C1*N^k + C2*N^{k-1} + \ldots + Ck*N + C$ is $O(N^k)$

  $2^N + 4*N^3 + 16$ is $O(2^N)$

  $5*N*\log(N) + 3*N$ is $O(N*\log(N))$

  1789 is $O(1)$

Linear Search - Complexity

```
function search(X, A, N)
j = 0;
while (j < N)
      if (A[j] == X)  return j;
      j++;
 endwhile;
return “Not_found”;
```

Linear Search - Complexity

- Time Complexity:

  “if” statement introduces possibilties

  - Best-case: O(1)
  - Worst case: O(N)
  - Average case: ???

Linear Search - Complexity

- Average case
  - Assume elements in A are randomly distributed.
  - Then for N different cases,

    1, 2, 3, … N

    are equally possible values of number of iterations.
  - So expected value: (1 + 2 + … + N) / N

Linear Search - Complexity

- Average case

  $(\Sigma_{i=0 \text{ to } N}\ I)/N = (N(N+1)/2\ )/N = (N+1)/2$

  is O(N)
- Space Complexity

  O(1) i.e. constant space.

Binary Search Algorithm

```
low = 1;  high = N;
while (low <= high) do
   mid = (low + high) /2;
   if (A[mid] = = x)  return x;
   else if (A[mid] < x) low = mid +1;
   else high = mid – 1;
endwhile;
   return Not_Found;
```

Binary Search - Complexity

- Often not interested in best case.
- Worst case:
  - Loop executes until low <= high
  - Size halved in each iteration
  - N, N/2, N/4, … 1
  - How many steps ?

Binary Search - Complexity

- Worst case:
  - K steps such that $2^K = N$ i.e. $\log_2 N$ steps

  is O(log(N))

Binary Search - Complexity

- Average case:
  - 1st iteration: 1 possible value
  - 2nd iteration: 2 possible values (left or right half)
  - 3rd iteration: 4 possible values (left of left, right of left, right of right, right of left)
  - i^{th} iteration: 2^{i-1} possible values

Binary Search - Complexity

- Average Case:
  1 + 2 + 2 + 3 + 3 + 3 + 3 + … (upto logN steps)
  Sigma($i*2^{i-1}$) for i = 1 to logN
  Evaluates to O(logN)
  1 element can be found with 1 comparison
  2 elements → 2
  4 elements → 3
  Above Sum =sum over all possibilities

- Space Complexity
  is O(1)

x^y - Complexity

- Assume $y = 2^k$
  - k steps in the iteration.
  - Complexity is O(k)
- General case

  $\text{Sigma}(\log_2 2^k)$ for $k = 1$ to $\log_2 y$

  $\log(\text{Product}(2^k))$ for $k = 1$ to $\log_2^y$

  $\log(y)$

```c
#include<stdio.h>
#include<math.h>
int pow2(int,int);
int main()
{
    // Solve X^Y where Y = 2^k
    int i,X,k,result,Y_next,Power;
    printf("Enter the X and Y values\n");
    scanf("%d %d",&X,&Y);
     Y_next = Y;
    Power = 1; result=1;
    while (Y_next > 0)
    {
      if (Y_next % 2 == 1)
        result = result * pow2(X, Power);
      Power = 2 * Power;
      Y_next = Y_next / 2;
    }
    printf("Result: %d\n",result);
    return 0;
}
```

```c
int pow2(int X, int Y)
{
        int i, result, k ;
        k=ceil((log(Y))/(log(2)));
        result = X; // the atomic solution is X
        for(i=0; i<k; i++)
                result = result * result ;
        return result;
}
```

```
gcc XpowY3.c –lm
./a.out
Enter the X and Y values
2 8
Result: 256
```