Data Structures and Algorithms (CS F211) – T1

Prof.N.L.Bhanu Murthy

BITS Pilani
Hyderabad Campus

Problem-1 Use the definition of Big-Oh to prove that $0.001n^3 - 1000n^2 \log n - 100n + 5$ is $O(n^3)$.

Problem-2 Prove or disprove each of the following.

1. $f(n) = O(g(n))$ implies $g(n) = O(f(n))$.
2. $f(n) + g(n) = \Theta(min(f(n), g(n)))$.
3. $f(n) = O(g(n))$ implies $g(n) = \Omega(f(n))$.

Problem-3 Rank the following functions by asymptotic growth rate in non-decreasing order:
$2^{64}-1$, n^3, $0.0001n^2$, $10000n$, $\log n^2$, $2^{\log n}$, $n \log n$, $n2^n$, 2^{1000}, n, $n^2 \log n$, 2^n, $\log n$, n^{100}, 4^n, $\log n^3$, n^n.

Problem-4 Is $2^{n+1} = O(2^n)$? Is $2^{2n} = O(2^n)$?

Problem-5 Use the definition of Big-Oh to prove that $n^{1+0.001}$ is not $O(n)$.

Problem-6 Express the function $n^3/1000 - 100n^2 - 100n + 3$ in terms of Θ-notation innovate achieve lead

Problem-7 Prove that $o(g(n)) \cap \omega(g(n))$ is the empty set.

Problem-8 Let processing time of an algorithm of Big-Oh complexity $O(f(n))$ be directly proportional to $f(n)$. Let three such algorithms A, B, and C have time complexity $O(n^2)$, $O(n^{1.5})$, and $O(n \log n)$ respectively. During a test, each algorithm spends 10 seconds to process 100 data items. Derive the time each algorithm should spend to process 10,000 items.

(a) Input A[n]

```
c=0
for i=1 to n
    for j=1 to n
        c=c+1
        A[c%n]= A[c%n]+1
    end
end
```

(b)
```
Input A[n]
c=0
for i=1 to n*n
        for j=1 to n*n*n
                c=c+1
                A[c%n]=A[c%n]+1
        end
end
```

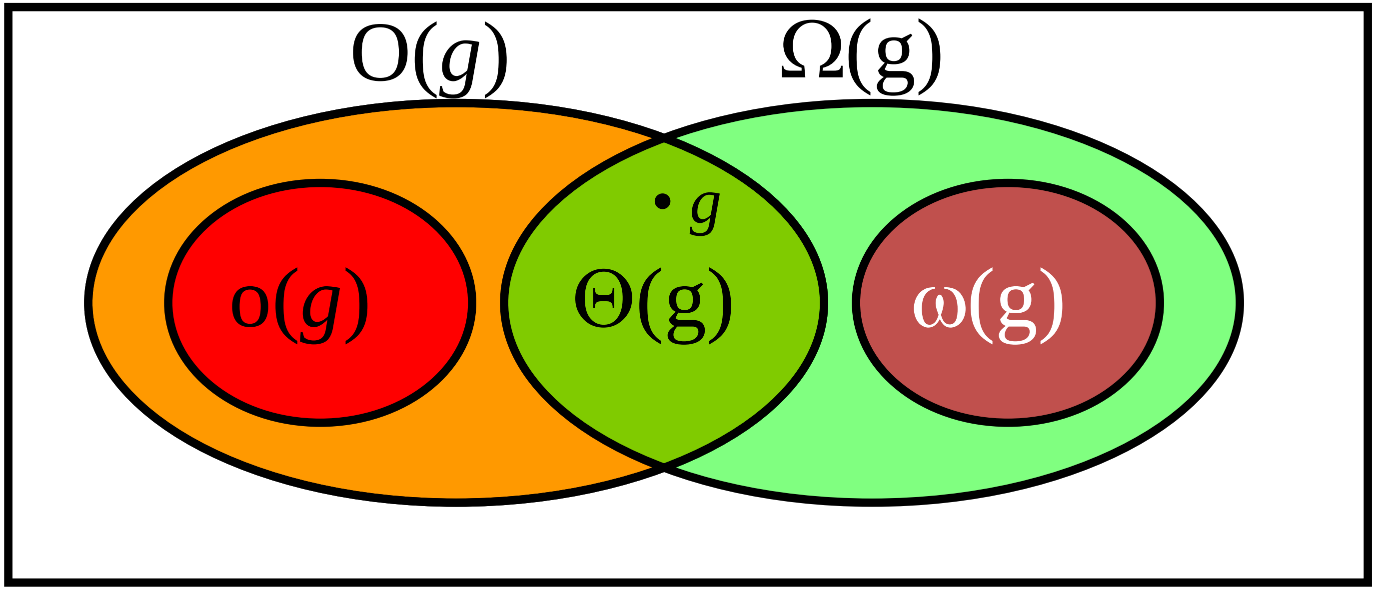
```
(c) Input A[n]
    c=0
    m=1
    for i=1 to n
            for j=1 to m*n
                    c=c+1
                    A[c%n]= A[c%n]+1
            end
            m = m/2;
    end
```

(e)
```
Input A[n]
m=1, c=0
for i=1 to n
        for j=1 to m
                c=c+1
        end
        m = m*2
end
```

```
(f) Input A[n]
    m = n-1
    while( m >= 1)
            print A[m]
            m = floor(m/3)
    end
```

```
(g) Input A[n]
    m = n-1
    while( m >= 1)
            for i=0 to m
                    print A[i]
            end
            m = floor(m/3)
    end
```

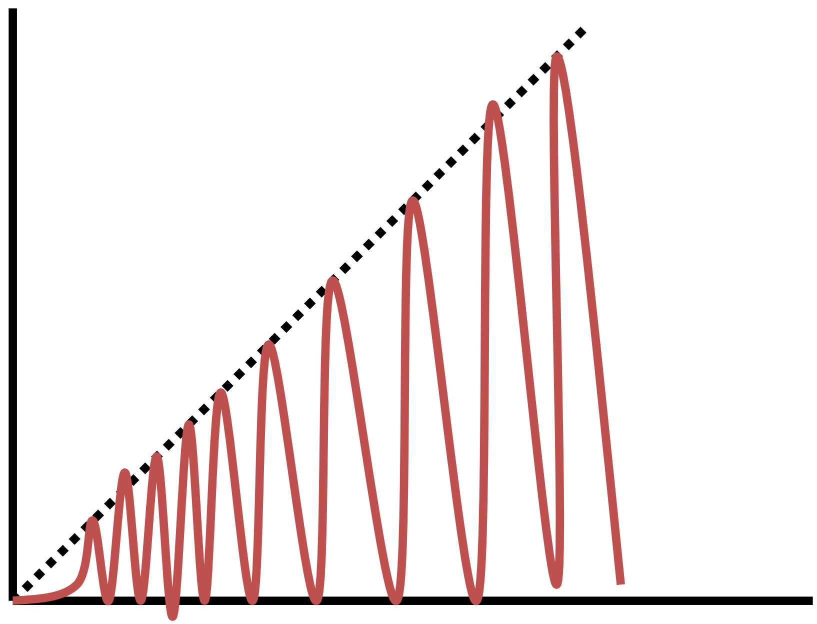
Problem-9 Show that $\log^3 n$ is $o(n^{1/3})$.


O(g)
Ω(g)
• g
o(g)
Θ(g)
ω(g)

Is the following statement true: $o(f) \subset O(x) - \Theta(x)$

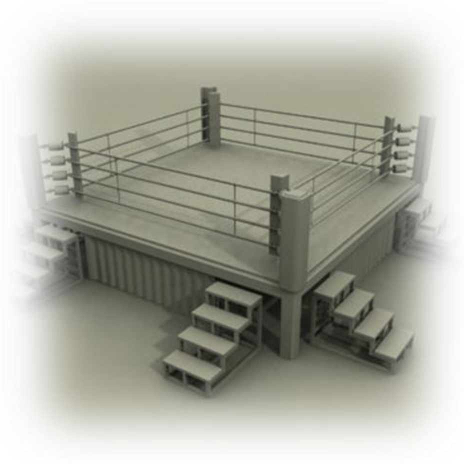
Is the following statement true: $o(f) \subset O(x) - \Theta(x)$

A function that is O(*x*), but neither o(*x*) nor Θ(*x*):



Problem-10 Show that the summation $\sum_{i=1}^{n} \lceil \log_2 i \rceil$ is $\Omega(n \log n)$.

Champion Problem



```
int i, j;                      O(1) time
j = 1;                         O(1) time
for (i = 2;  i <= n;  i++)     O(1) time for operations like i = 2, i <= n and i++)
    if (A[i] > A[j])           O(1) time
        j = i;                 O(1) time
return j;                      O(1) time
```

$O(n)$ iterations

In fact n -1 iterations

Adding everything together
yields
an upper bound on the worst-case time complexity.

Complexity is O(1) + O(1) + O(n) {O(1) + O(1) + O(1)} + O(1) = O(n)

Practice exercises – Find sum of 'n' integers

Devise an algorithm that finds the sum of all the integers in a list.

procedure *sum*(a_1, a_2, …, a_n: integers)
    s := 0   {sum of elems so far}

    for i := 1 **to** n {go thru all elements}
        s := $s + a_i$ {add current item}
        {at this point s is the sum of all items}
    return s

Practice exercises

What is the complexity of the following algorithm?

```
Algorithm Awesome(A: list [1,2,..n])
        var int i, j, k, sum
        for i from 1 to 100
                for j from 1 to 1000
                        for k from 1 to 50
                                sum = sum + i + j + k
```

Practice exercises

Work out the computational complexity of the following piece of code:

```
for( int i = n;  i > 0;  i /= 2 ) {
  for( int j = 1;  j < n;  j *= 2 ) {
    for( int k = 0;  k < n;  k += 2 ) {
        ... // constant number of operations
    }
  }
}
```

In the outer `for`-loop, the variable `i` keeps halving so it goes round $\log_2 n$ times. For each `i`, next loop goes round also $\log_2 n$ times, because of doubling the variable `j`. The innermost loop by `k` goes round $\frac{n}{2}$ times. Loops are nested, so the bounds may be multiplied to give that the algorithm is $O\left(n(\log n)^2\right)$.

Practice exercises

Work out the computational complexity of the following piece of code assuming that $n = 2^m$:

```
for( int i = n; i > 0; i-- ) {
        for( int j = 1; j < n; j *= 2 ) {
                for( int k = 0; k < j; k++ ) {
                        for(int m = 0; m < 10000; m++)
                                sum = sum + m;
                                        }
                                }
                        }
```

Practice exercises

```
for( int i = n; i > 0; i-- ) {                    -------L1
        for( int j = 1; j < n; j *= 2 ) {    --------L2
              for( int k = 0; k < j; k++ ) {    --------L3
                  for(int m = 0; m < 10000; m++)     --------L4
                       sum = sum + m;
                             }
                       }
              }
```

The outer most for-loop, i.e., L1, runs for O(n) times.

For each j in L2 loop, the L3 loop runs for j times, so that the two inner loops, L2 and L3

together go round $1 + 2 + 4 + . . . + 2^{m-1} = 2^m – 1 = n - 1 = O(n)$ times.

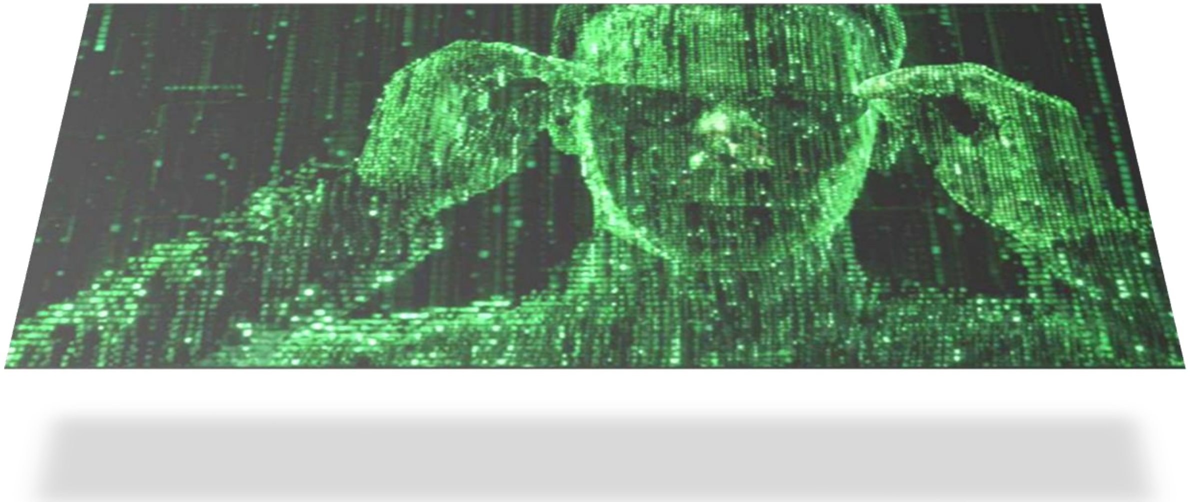
The inner-most loop, i.e., L4, is of O(1) complexity.

Loops are nested, so the bounds may be multiplied to give that the algorithm is $O(n^2)$.

Matrix Multiplication

Input: two $n \times n$ matrices A and B.

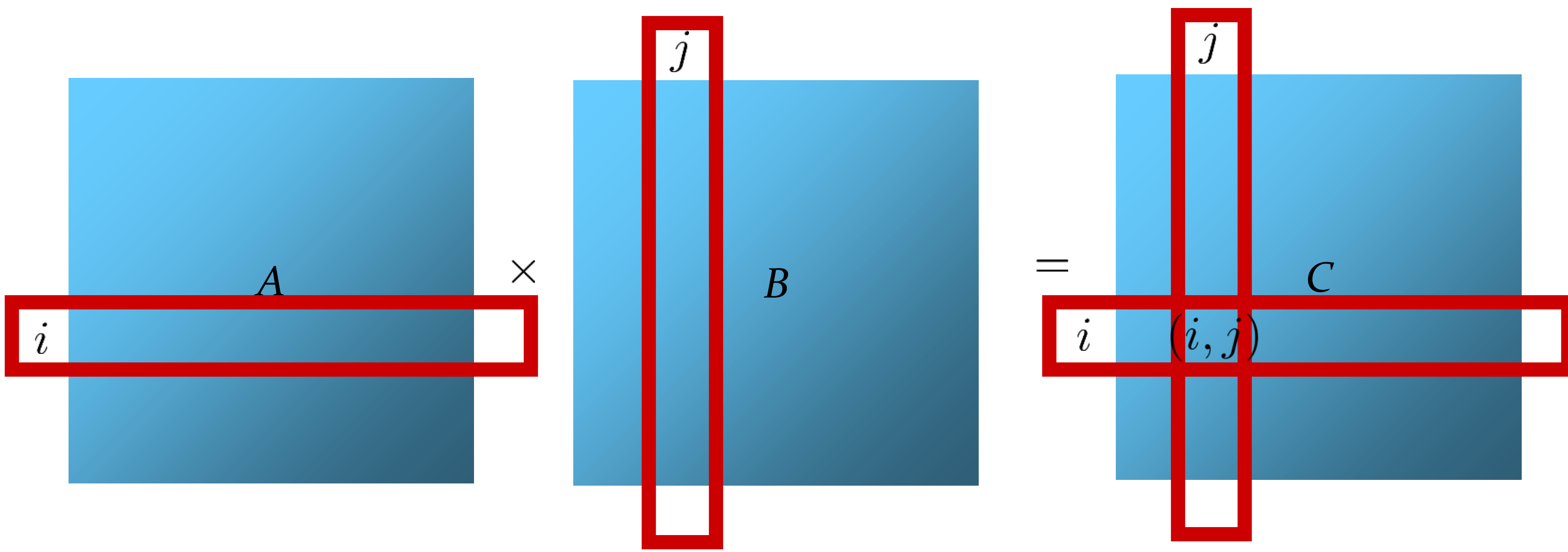
Output: the product matrix $C = A \times B$



Naïve algorithm

for $i = 1$ to n
  for $j = 1$ to n
    Let $C(i, j)$ be the inner product of
      the i-th row of A and the j-th column of B.

Since each inner product of two n-element vectors takes $\Theta(n)$ time, the time complexity of the naive algorithm is $\Theta(n^3)$.


j
A
$\times$
B
$=$
C
j
i
i
(i, j)

Is $o(n^3)$ time possible?

innovate achieve lead

Thank You!!