Orders of magnitude

Let $f$, $g$, and $h$ be functions from $\mathbb{N}$ to $\mathbb{R}$.

- We say that $g(n)$ is in the **order of magnitude** of $f(n)$ and we write $f(n) \in \Theta(g(n))$ if there exist two strictly positive constants $c_1$ and $c_2$ such that for $n$ big enough we have
  \[ 0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n). \] (1)

- We say that $g(n)$ is an **asymptotic upper bound** of $f(n)$ and we write $f(n) \in O(g(n))$ if there exists a strictly positive constants $c_2$ such that for $n$ big enough we have
  \[ 0 \leq f(n) \leq c_2 \cdot g(n). \] (2)

- We say that $g(n)$ is an **asymptotic lower bound** of $f(n)$ and we write $f(n) \in \Omega(g(n))$ if there exists a strictly positive constants $c_1$ such that for $n$ big enough we have
  \[ 0 \leq c_1 \cdot g(n) \leq f(n). \] (3)

Review of Complexity Notions

Divide-and-Conquer Recurrences

Matrix Multiplication

Merge Sort

Tableau Construction
Examples

- With \( f(n) = \frac{1}{2}n^2 - 3n \) and \( g(n) = n^2 \) we have \( f(n) \in \Theta(g(n)) \).
  Indeed we have
  \[
  c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2.
  \]
  for \( n \geq 12 \) with \( c_1 = \frac{1}{4} \) and \( c_2 = \frac{1}{2} \).

- Assume that there exists a positive integer \( n_0 \) such that \( f(n) > 0 \) and \( g(n) > 0 \) for every \( n \geq n_0 \). Then we have
  \[
  \max(f(n), g(n)) \in \Theta(f(n) + g(n)).
  \]
  Indeed we have
  \[
  \frac{1}{2}(f(n) + g(n)) \leq \max(f(n), g(n)) \leq (f(n) + g(n)).
  \]

- Assume \( a \) and \( b \) are positive real constants. Then we have
  \[
  (n + a)^b \in \Theta(n^b).
  \]
  Indeed for \( n \geq a \) we have

Another example

Let us give another fundamental example. Let \( p(n) \) be a (univariate) polynomial with degree \( d > 0 \). Let \( a_d \) be its leading coefficient and assume \( a_d > 0 \). Then we have

1. \( k \geq d \) then \( p(n) \in \mathcal{O}(n^k) \),
2. \( k \leq d \) then \( p(n) \in \Omega(n^k) \),
3. \( k = d \) then \( p(n) \in \Theta(n^k) \).

Exercise: Prove the following
\[
\sum_{k=1}^{n} k \in \Theta(n^2).
\]
Divide-and-Conquer Algorithms

Divide-and-conquer algorithms proceed as follows.

- **Divide** the input problem into sub-problems.
- **Conquer** on the sub-problems by solving them directly if they are small enough or proceed recursively.
- **Combine** the solutions of the sub-problems to obtain the solution of the input problem.

**Equation satisfied by** $T(n)$. Assume that the size of the input problem increases with an integer $n$. Let $T(n)$ be the time complexity of a divide-and-conquer algorithm to solve this problem. Then $T(n)$ satisfies an equation of the form:

$$T(n) = a \cdot T(n/b) + f(n).$$

(13)

where $f(n)$ is the cost of the combine-part, $a \geq 1$ is the number of recursively calls and $n/b$ with $b > 1$ is the size of a sub-problem.

---

**Divide-and-Conquer Recurrences**

Solving divide-and-conquer recurrences (1/2)

**Divide-and-Conquer Recurrences**

Solving divide-and-conquer recurrences (2/2)

**Labeled tree associated with the equation.** Assume $n$ is a power of $b$, say $n = b^p$. To solve the equation

$$T(n) = a \cdot T(n/b) + f(n).$$

we can associate a labeled tree $A(n)$ to it as follows.

1. If $n = 1$, then $A(n)$ is reduced to a single leaf labeled $T(1)$.
2. If $n > 1$, then the root of $A(n)$ is labeled by $f(n)$ and $A(n)$ possesses $a$ labeled sub-trees all equal to $A(n/b)$.

The labeled tree $A(n)$ associated with $T(n) = a \cdot T(n/b) + f(n)$ has height $p + 1$. Moreover the sum of its labels is $T(n)$.

---

**IDEA:** Compare $n^{\log_b a}$ with $f(n)$. 

Master Theorem: case \( n^{\log_b a} \gg f(n) \)

- \( n^{\log_b a} \gg f(n) \)
  - \( n^{\log_b a} \gg f(n) \)
  - \( n^{\log_b a} \gg f(n) \)
  - \( n^{\log_b a} \gg f(n) \)
  - \( n^{\log_b a} \gg f(n) \)
  - \( n^{\log_b a} \gg f(n) \)
  - \( n^{\log_b a} \gg f(n) \)

- \( h = \log_b n \)
  - \( h = \log_b n \)
  - \( h = \log_b n \)
  - \( h = \log_b n \)
  - \( h = \log_b n \)
  - \( h = \log_b n \)
  - \( h = \log_b n \)

- \( T(1) = \Theta(n^{\log_b a}) \)
  - \( T(1) = \Theta(n^{\log_b a}) \)
  - \( T(1) = \Theta(n^{\log_b a}) \)
  - \( T(1) = \Theta(n^{\log_b a}) \)
  - \( T(1) = \Theta(n^{\log_b a}) \)
  - \( T(1) = \Theta(n^{\log_b a}) \)
  - \( T(1) = \Theta(n^{\log_b a}) \)

**GEOMETRICALLY INCREASING**

Specifically, \( f(n) = \Omega(n^{\log_b a} - \epsilon) \) for some constant \( \epsilon > 0 \).

- \( f(n/b) \)
  - \( f(n/b) \)
  - \( f(n/b) \)
  - \( f(n/b) \)
  - \( f(n/b) \)
  - \( f(n/b) \)
  - \( f(n/b) \)

- \( a f(n/b) \)
  - \( a f(n/b) \)
  - \( a f(n/b) \)
  - \( a f(n/b) \)
  - \( a f(n/b) \)
  - \( a f(n/b) \)
  - \( a f(n/b) \)

- \( a^2 f(n/b^2) \)
  - \( a^2 f(n/b^2) \)
  - \( a^2 f(n/b^2) \)
  - \( a^2 f(n/b^2) \)
  - \( a^2 f(n/b^2) \)
  - \( a^2 f(n/b^2) \)
  - \( a^2 f(n/b^2) \)

- \( a^3 f(n/b^3) \)
  - \( a^3 f(n/b^3) \)
  - \( a^3 f(n/b^3) \)
  - \( a^3 f(n/b^3) \)
  - \( a^3 f(n/b^3) \)
  - \( a^3 f(n/b^3) \)
  - \( a^3 f(n/b^3) \)

- \( \cdots \)
  - \( \cdots \)
  - \( \cdots \)
  - \( \cdots \)
  - \( \cdots \)
  - \( \cdots \)
  - \( \cdots \)

- \( a^{\log_b n} T(1) = \Theta(n^{\log_b a}) \)
  - \( a^{\log_b n} T(1) = \Theta(n^{\log_b a}) \)
  - \( a^{\log_b n} T(1) = \Theta(n^{\log_b a}) \)
  - \( a^{\log_b n} T(1) = \Theta(n^{\log_b a}) \)
  - \( a^{\log_b n} T(1) = \Theta(n^{\log_b a}) \)
  - \( a^{\log_b n} T(1) = \Theta(n^{\log_b a}) \)
  - \( a^{\log_b n} T(1) = \Theta(n^{\log_b a}) \)

Master Theorem: case \( f(n) \gg n^{\log_b a} \)

- \( f(n) \gg n^{\log_b a} \)
  - \( f(n) \gg n^{\log_b a} \)
  - \( f(n) \gg n^{\log_b a} \)
  - \( f(n) \gg n^{\log_b a} \)
  - \( f(n) \gg n^{\log_b a} \)
  - \( f(n) \gg n^{\log_b a} \)

- \( h = \log_b n \)
  - \( h = \log_b n \)
  - \( h = \log_b n \)
  - \( h = \log_b n \)
  - \( h = \log_b n \)
  - \( h = \log_b n \)
  - \( h = \log_b n \)

- \( T(1) = \Theta(f(n)) \)
  - \( T(1) = \Theta(f(n)) \)
  - \( T(1) = \Theta(f(n)) \)
  - \( T(1) = \Theta(f(n)) \)
  - \( T(1) = \Theta(f(n)) \)
  - \( T(1) = \Theta(f(n)) \)
  - \( T(1) = \Theta(f(n)) \)

**GEOMETRICALLY DECREASING**

Specifically, \( f(n) = \Omega(n^{\log_b a} + \epsilon) \) for some constant \( \epsilon > 0 \).*

- \( f(n/b) \)
  - \( f(n/b) \)
  - \( f(n/b) \)
  - \( f(n/b) \)
  - \( f(n/b) \)
  - \( f(n/b) \)
  - \( f(n/b) \)

- \( a f(n/b) \)
  - \( a f(n/b) \)
  - \( a f(n/b) \)
  - \( a f(n/b) \)
  - \( a f(n/b) \)
  - \( a f(n/b) \)
  - \( a f(n/b) \)

- \( a^2 f(n/b^2) \)
  - \( a^2 f(n/b^2) \)
  - \( a^2 f(n/b^2) \)
  - \( a^2 f(n/b^2) \)
  - \( a^2 f(n/b^2) \)
  - \( a^2 f(n/b^2) \)
  - \( a^2 f(n/b^2) \)

- \( a^3 f(n/b^3) \)
  - \( a^3 f(n/b^3) \)
  - \( a^3 f(n/b^3) \)
  - \( a^3 f(n/b^3) \)
  - \( a^3 f(n/b^3) \)
  - \( a^3 f(n/b^3) \)
  - \( a^3 f(n/b^3) \)

- \( \cdots \)
  - \( \cdots \)
  - \( \cdots \)
  - \( \cdots \)
  - \( \cdots \)
  - \( \cdots \)
  - \( \cdots \)

- \( a^{\log_b n} T(1) = \Theta(n^{\log_b a}) \)
  - \( a^{\log_b n} T(1) = \Theta(n^{\log_b a}) \)
  - \( a^{\log_b n} T(1) = \Theta(n^{\log_b a}) \)
  - \( a^{\log_b n} T(1) = \Theta(n^{\log_b a}) \)
  - \( a^{\log_b n} T(1) = \Theta(n^{\log_b a}) \)
  - \( a^{\log_b n} T(1) = \Theta(n^{\log_b a}) \)
  - \( a^{\log_b n} T(1) = \Theta(n^{\log_b a}) \)

*and \( f(n) \) satisfies the regularity condition that \( a f(n/b) \leq c f(n) \) for some constant \( c < 1 \).

More examples

- Consider the relation:

  \[
  T(n) = 2 T(n/2) + n^2. 
  \]  

  We obtain:

  \[
  T(n) = n^2 + \frac{n^2}{2} + \frac{n^2}{4} + \frac{n^2}{8} + \cdots + \frac{n^2}{2^p} + n T(1). 
  \]  

  Hence we have:

  \[
  T(n) \in \Theta(n^2). 
  \]

- Consider the relation:

  \[
  T(n) = 3 T(n/3) + n. 
  \]  

  We obtain:

  \[
  T(n) \in \Theta(\log_3(n) n). 
  \]
Master Theorem when $b = 2$

Let $a > 0$ be an integer and let $f, T : \mathbb{N} \rightarrow \mathbb{R}_+$ be functions such that

(i) $f(2n) \geq 2f(n)$ and $f(n) \geq n$.

(ii) If $n = 2^p$ then $T(n) \leq a T(n/2) + f(n)$.

Then for $n = 2^p$ we have

(1) if $a = 1$ then

$$T(n) \leq (2 - 2/n) f(n) + T(1) \in O(f(n)),$$

(19)

(2) if $a = 2$ then

$$T(n) \leq f(n) \log_2(n) + T(1) n \in O(\log_2(n) f(n)),$$

(20)

(3) if $a \geq 3$ then

$$T(n) \leq \frac{2}{a - 2} \left( n^{\log_2(a) - 1} - 1 \right) f(n) + T(1) n^{\log_2(a)} \in O(f(n) n^{\log_2(a) - 1}).$$

(21)

Master Theorem when $b = 2$

Hence

$$T(2^p) \leq a^p T(1) + f(2^p) \sum_{j=0}^{p-1} \left( \frac{a}{2} \right)^j.$$

(25)

Moreover

$$f(2^p) \geq 2 f(2^{p-1})$$

$$f(2^p) \geq 2^2 f(2^{p-2})$$

$$\vdots$$

$$f(2^p) \geq 2^j f(2^{p-j})$$

Thus

$$\sum_{j=0}^{p-1} a^j f(2^{p-j}) \leq f(2^p) \sum_{j=0}^{p-1} \left( \frac{a}{2} \right)^j.$$

(24)

For $a = 1$ we obtain

$$T(2^p) \leq T(1) + f(2^p) \sum_{j=0}^{p-1} \left( \frac{1}{2} \right)^j$$

$$= T(1) + f(2^p) \frac{1}{2} - 1$$

$$= T(1) + f(n) (2 - 2/n).$$

(26)

For $a = 2$ we obtain

$$T(2^p) \leq 2^p T(1) + f(2^p) p$$

$$= n T(1) + f(n) \log_2(n).$$

(27)
Master Theorem cheat sheet

For \( a \geq 1 \) and \( b > 1 \), consider again the equation

\[
T(n) = a \cdot T(n/b) + f(n).
\]  

(28)

- We have:

\[
(\exists \varepsilon > 0) \ f(n) \in O(n^{\log_b a - \varepsilon}) \implies T(n) \in \Theta(n^{\log_b a})
\]

(29)

- We have:

\[
(\exists \varepsilon > 0) \ f(n) \in \Theta(n^{\log_b a \log k n}) \implies T(n) \in \Theta(n^{\log_b a \log k + 1} n)
\]

(30)

- We have:

\[
(\exists \varepsilon > 0) \ f(n) \in \Omega(n^{\log_b a + \varepsilon}) \implies T(n) \in \Theta(f(n))
\]

(31)

### Master Theorem quizz!

- \( T(n) = 4T(n/2) + n \)

- \( T(n) = 4T(n/2) + n^2 \)

- \( T(n) = 4T(n/2) + n^3 \)

- \( T(n) = 4T(n/2) + n^2 / \log n \)

### Plan

1. Review of Complexity Notions
2. Divide-and-Conquer Recurrences
3. Matrix Multiplication
4. Merge Sort
5. Tableau Construction

### Matrix multiplication

We will study three approaches:

- a naive and iterative one
- a divide-and-conquer one
- a divide-and-conquer one with memory management consideration

\[
\begin{bmatrix}
c_{11} & c_{12} & \cdots & c_{1n} \\
c_{21} & c_{22} & \cdots & c_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
c_{n1} & c_{n2} & \cdots & c_{nn}
\end{bmatrix}
\begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1n} \\
a_{21} & a_{22} & \cdots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{n1} & a_{n2} & \cdots & a_{nn}
\end{bmatrix}
\begin{bmatrix}
b_{11} & b_{12} & \cdots & b_{1n} \\
b_{21} & b_{22} & \cdots & b_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
b_{n1} & b_{n2} & \cdots & b_{nn}
\end{bmatrix}
\]
Matrix Multiplication

Naive iterative matrix multiplication

cilk_for (int i=1; i<n; ++i) {
cilk_for (int j=0; j<n; ++j) {
for (int k=0; k<n; ++k {
C[i][j] += A[i][k] * B[k][j];
}
}
}

• Work: ?
• Span: ?
• Parallelism: ?

Matrix multiplication based on block decomposition

\[
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix} =
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix} \cdot
\begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
\]

= \begin{bmatrix}
A_{11}B_{11} + A_{12}B_{12} & A_{11}B_{12} + A_{12}B_{22} \\
A_{21}B_{11} + A_{22}B_{12} & A_{21}B_{12} + A_{22}B_{22}
\end{bmatrix}

The divide-and-conquer approach is simply the one based on blocking, presented in the first lecture.

Divide-and-conquer matrix multiplication

// C <- C + A * B
void MMult(T *C, T *A, T *B, int n, int size) {
T *D = new T[n*n];
cilk_spawn MMult(C11, A11, B11, n/2, size);
cilk_spawn MMult(C12, A11, B12, n/2, size);
cilk_spawn MMult(C22, A21, B12, n/2, size);
cilk_spawn MMult(C21, A21, B11, n/2, size);
cilk_spawn MMult(D11, A12, B21, n/2, size);
cilk_spawn MMult(D12, A12, B22, n/2, size);
cilk_sync;
MAdd(C, D, n, size); // C += D;
delete[] D;
}

• Work: Θ(n^3)
• Span: Θ(n)
• Parallelism: Θ(n^2)
Matrix Multiplication

Divide-and-conquer matrix multiplication

Template <typename T>

```cpp
template <typename T>
void MMult2(T *C, T *A, T *B, int n, int size) {
    // base case & partition matrices
    cilk_spawn MMult2(C11, A11, B11, n/2, size);
    cilk_spawn MMult2(C12, A11, B12, n/2, size);
    cilk_spawn MMult2(C21, A21, B11, n/2, size);
    cilk_spawn MMult2(C22, A21, B12, n/2, size);
    cilk_spawn MMult2(C11, A12, B21, n/2, size);
    cilk_spawn MMult2(C12, A12, B22, n/2, size);
    cilk_spawn MMult2(C22, A22, B22, n/2, size);
    cilk_sync;
}
```

**Work** ? **Span** ? **Parallelism** ?

Besides, saving space often saves time due to hierarchical memory.
Merging two sorted arrays

```c
void Merge(T *C, T *A, T *B, int na, int nb) {
    while (na>0 && nb>0) {
        if (*A <= *B) {
            *C++ = *A++; na--;
        } else {
            *C++ = *B++; nb--;
        }
    }
    while (na>0) {
        *C++ = *A++; na--;
    }
    while (nb>0) {
        *C++ = *B++; nb--;
    }
}
```

Time for merging $n$ elements is $\Theta(n)$.

Parallel merge sort with serial merge

```c
template <typename T>
void MergeSort(T *B, T *A, int n) {
    if (n==1) {
        B[0] = A[0];
    } else {
        T* C[n];
        cilk_spawn MergeSort(C, A, n/2);
        MergeSort(C+n/2, A+n/2, n-n/2);
        cilk_sync;
        Merge(B, C, C+n/2, n/2, n-n/2);
    }
}
```

- $T_1(n) = 2T_1(n/2) + \Theta(n)$ thus $T_1(n) = \Theta(n \log n)$.
- $T_\infty(n) = T_\infty(n/2) + \Theta(n)$ thus $T_\infty(n) = \Theta(n)$.
- $T_1(n)/T_\infty(n) = \Theta(\log n)$. Puny parallelism!
- We need to parallelize the merge!

- **Work?**
- **Span?**
Parallel merge

Idea: if the total number of elements to be sorted in $n = n_a + n_b$ then the maximum number of elements in any of the two merges is at most $3n/4$.

threshold $P_M_p(n)$ be the $p$-processor running time of $P$-MERGE.

In the worst case, the span of $P$-MERGE is

$$PM_{∞}(n) \leq PM_{∞}(3n/4) + Θ(lg n) = Θ(lg^2 n)$$

The worst-case work of $P$-MERGE satisfies the recurrence

$$PM_1(n) \leq PM_1(αn) + PM_1((1 − α)n) + Θ(lg n)$$

Recall $PM_1(n) \leq PM_1(αn) + PM_1((1 − α)n) + Θ(lg n)$ for some $1/4 \leq α ≤ 3/4$.

To solve this hairy equation we use the substitution method.

We assume there exist some constants $a, b > 0$ such that $PM_1(n) \leq an − blg n$ holds for all $1/4 \leq α \leq 3/4$.

After substitution, this hypothesis implies:

$$PM_1(n) \leq an − blg n − blg n + Θ(lg n)$$

We can pick $b$ large enough such that we have $PM_1(n) \leq an − blg n$ for all $1/4 \leq α \leq 3/4$ and all $n > 1/

Then pick $a$ large enough to satisfy the base conditions.

Finally we have $PM_1(n) = Θ(n)$.  

```cpp
template <typename T>
void P_Merge(T *C, T *A, T *B, int na, int nb) {
    if (na < nb) {
        P_Merge(C, B, A, nb, na);
    } else if (na==0) {
        return;
    } else {
        int ma = na/2;
        int mb = BinarySearch(A[ma], B, nb);
        C[ma+mb] = A[ma];
        cilk_spawn P_Merge(C, A, B, ma, mb);
        P_Merge(C+ma+mb+1, A+ma+1, B+mb, na-ma-1, nb-mb);
        cilk_sync;
    }
}
```

```cpp

def Pirr = an − blg n − blg n + Θ(lg n)

We can pick $b$ large enough such that we have $PM_1(n) \leq an − blg n$ for all $1/4 \leq α \leq 3/4$ and all $n > 1/

Then pick $a$ large enough to satisfy the base conditions.

Finally we have $PM_1(n) = Θ(n)$.  

```cpp
```
Parallel merge sort with parallel merge

```cpp
template<typename T>
void P_MergeSort(T *B, T *A, int n) {
    if (n==1) {
        B[0] = A[0];
    } else {
        T C[n];
        cilk_spawn P_MergeSort(C, A, n/2);
        P_MergeSort(C+n/2, A+n/2, n-n/2);
        cilk_sync;
        P_Merge(B, C, C+n/2, n/2, n-n/2);
    }
}
```

The work satisfies $T_1(n) = 2T_1(n/2) + \Theta(n)$ (as usual) and we have $T_1(n) = \Theta(n \log n)$.

The worst case critical-path length of the MERGE-SORT now satisfies $T_\infty(n) = T_\infty(n/2) + \Theta(\log^2 n) = \Theta(\log^3 n)$.

The parallelism is now $\Theta(n \log n) / \Theta(\log^3 n) = \Theta(n / \log^2 n)$.

Constructing a tableau $A$ satisfying a relation of the form:

$$A[i, j] = R(A[i-1, j], A[i-1, j-1], A[i, j-1]). \quad (32)$$

The work is $\Theta(n^2)$.
Recursive construction

Parallel code

- $T_1(n) = 4T_1(n/2) + \Theta(1)$, thus $T_1(n) = \Theta(n^2)$.
- $T_\infty(n) = 3T_\infty(n/2) + \Theta(1)$, thus $T_\infty(n) = \Theta(n\log_2 3)$.
- **Parallelism**: $\Theta(n^{2-\log_2 3}) = \Omega(n^{0.41})$.

A more parallel construction

Parallel code

- $T_1(n) = 9T_1(n/3) + \Theta(1)$, thus $T_1(n) = \Theta(n^2)$.
- $T_\infty(n) = 5T_\infty(n/3) + \Theta(1)$, thus $T_\infty(n) = \Theta(n^{\log_3 5})$.
- **Parallelism**: $\Theta(n^{2-\log_3 5}) = \Omega(n^{0.53})$.
- This nine-way d-n-c has more parallelism than the four way but exhibits more cache complexity (more on this later).

Acknowledgements

- Charles E. Leiserson (MIT) for providing me with the sources of its lecture notes.
- Matteo Frigo (Intel) for supporting the work of my team with Cilk++ and offering us the next lecture.
- Yuzhen Xie (UWO) for helping me with the images used in these slides.
- Liyun Li (UWO) for generating the experimental data.