Overview

- General Information.
- Algorithms and Computational Problems.
- Analysis of Algorithms.
- Growth of Functions.
- Divide and Conquer.
What is a Computational Problem?

- The statement of a computational problem specifies in general terms the relation between input and output.

- **INPUT:** A set \( Z \) of \( n \) points in the Euclidean plane.

- **OUTPUT:** *Convex Hull*: Smallest convex set containing \( Z \).
Instance of a problem
Instance of a problem
What is an Algorithm?

- A set of computational steps that transforms a set of input values and produces a set of output values.
Multiplication: Two Algorithms

- There can be several algorithms that can solve the same problem.

\[
\begin{array}{c}
57 \\
\times \ 
19 \\
\hline
513 \\
+ \ 57 \\
\hline
1083 \\
\end{array}
\]

traditional
Algorithms

Introduction

Multiplication: Two Algorithms

- There can be several algorithms that can solve the same problem.

\[
\begin{array}{c}
57 \\
\times 19 \\
\hline
513 \\
+ 57 \\
\hline
1083
\end{array}
\]

traditional

\[
\begin{array}{c}
57 \\
\times 19 \\
\hline
28 \ 38 \ 0 \\
14 \ 76 \ 0 \\
7 \ 152 \ 152 \\
3 \ 304 \ 304 \\
1 \ 608 \ + \ 608 \\
\hline
1083
\end{array}
\]
a la russe

- Traditional multiplication and multiplication a la russe are two different solutions of the same problem of multiplying positive integers. Multiplication of 57 by 19 is a particular instance of this problem.
What is a Correct Algorithm?

- Always outputs correct answer.
- Halts.
Objectives of This Course

- Design of algorithms.
- Analysis of algorithms.
- Data structures.
- Paradigms.
- Easy vs. hard problems.
Algorithms

Insertion Sort

1. for j=2 to length(A) do c1 n
2. key=A[j] c2 n-1
3. i=j-1 c3 n-1
4. while i>0 and A[i]>key do c4 t2+t3+...+tn
5. A[i+1]=A[i] c5 t2-1+t3-1+...+tn-1
6. i=i-1 c6 t2-1+t3-1+...+tn-1
7. A[i+1]=key c7 n-1

\[ T(n) = c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4 \sum_{j=2}^{n} t_j + (c_5 + c_6) \sum_{j=2}^{n} (t_j - 1) \]
Insertion Sort - Best Case

1. for \( j=2 \) to length(A) do 
2. \( \text{key} = A[j] \)
3. \( i = j - 1 \)
4. while \( i > 0 \) and \( A[i] > \text{key} \) do
5. \( A[i+1] = A[i] \)
6. \( i = i - 1 \)
7. \( A[i+1] = \text{key} \)

\[
T(n) = c_1n + (c_2 + c_3 + c_7)(n-1) + c_4 \sum_{j=2}^{n} t_j + (c_5 + c_6) \sum_{j=2}^{n} (t_j - 1)
\]

Best-case: \( t_j = 1 \) for all \( j = 2, 3..., n \)

\[
T(n) = c_1n + (c_2 + c_3 + c_7)(n-1) + c_4(n-1) + 0
\]
Insertion Sort - Worst Case

1. for \( j=2 \) to length(A) do  
2. \( \text{key}=A[j] \)  
3. \( i=j-1 \)  
4. while \( i>0 \) and \( A[i]>\text{key} \) do  
5. \( A[i+1]=A[i] \)  
6. \( i=i-1 \)  
7. \( A[i+1]=\text{key} \)

\[
T(n) = c_1 n + (c_2 + c_3 + c_7)(n-1) + c_4 \sum_{j=2}^{n} t_j + (c_5 + c_6) \sum_{j=2}^{n} (t_j - 1) =
\]

Worst-case: \( t_j = j \) for all \( j = 2, 3..., n \)

\[
T(n) = \frac{c_4}{2} n(n+1) - 1 + \frac{c_5 + c_6}{2} (n-1)n + c_1 n + (c_2 + c_3 + c_7)(n-1)
\]
Convex Hull by Insertion

CONVEX-HULL-BY-INSERTION(Z)

1. \( CH = \text{CONVEX-HULL-3}(z_1,z_2,z_3) \)
2. do \( i = 4 \) to \( n \) \( n-2 \) times
3. if OUTSIDE(\( z_i, CH \)) then \( n-3 \) times
4. \( CH = \text{INSERT}(z_i, CH) \) \( n'-3 \) times

1 2 3 4 5 6 7 8 9 10
Convex Hull by Insertion

CONVEX-HULL-BY-INSERTION(Z)

1. CH = CONVEX-HULL-3(z1,z2,z3)  
2. do i = 4 to n  
3. if OUTSIDE(zi,CH) then  
4. CH = INSERT(zi,CH)

---

![Convex Hull Diagram](image-url)
Convex Hull by Insertion

CONVEX-HULL-BY-INSERTION(Z)

1. \( CH = \text{CONVEX-HULL-3}(z_1, z_2, z_3) \)  \( c_1 \)
2. do \( i = 4 \) to \( n \)  \( c_2 \) \( n-3 \) times
3. if OUTSIDE\((z_i, CH)\) then  \( c_3 (i-1) \) \( n-3 \) times
4. \( CH = \text{INSERT}(z_i, CH) \)  \( c_4 (i'-3) \) \( n'-3 \) times
Convex Hull by Insertion

CONVEX-HULL-BY-INSERTION(Z)

1. CH = CONVEX-HULL-3(z1,z2,z3)  
2. do i = 4 to n  
3. if OUTSIDE(zi,CH) then  
4. CH = INSERT(zi,CH)

\[ c_1 \]
\[ c_2 \text{ n-3 times} \]
\[ c_3(i-1) \text{ n-3 times} \]
\[ c_4(i'-3) \text{ n’-3 times} \]
Convex Hull by Insertion

CONVEX-HULL-BY-INSERTION(Z)

1. CH = CONVEX-HULL-3(z1, z2, z3)  \(c_1\)
2. do i = 4 to n \(c_2\) n-3 times
3. if OUTSIDE(zi, CH) then \(c_3(i-1)\) n-3 times
4. CH = INSERT(zi, CH) \(c_4(i'-3)\) n’-3 times

---

Diagram showing the convex hull of a set of points.

Points labeled 1 to 10 and a shaded triangle representing the convex hull.
Convex Hull by Insertion

CONVEX-HULL-BY-INSERTION(Z)

1. CH = CONVEX-HULL-3(z1, z2, z3)  \(c_1\)
2. do i = 4 to n \(n-3\) times \(c_2\)
3. if OUTSIDE(zi, CH) then \(i-1\) \(n-3\) times \(c_3\)
4. CH = INSERT(zi, CH) \(i'-3\) \(n'-3\) times \(c_4\)
Convex Hull by Insertion

CONVEX-HULL-BY-INSERTION(Z)

1. CH = CONVEX-HULL-3(z1,z2,z3) c1
2. do i = 4 to n c2 n-3 times
3. if OUTSIDE(zi,CH) then c3(i-1) n-3 times
4. CH = INSERT(zi,CH) c4(i’-3) n’-3 times
Convex Hull by Insertion

CONVEX-HULL-BY-INSERTION(Z)

1. \( CH = \text{CONVEX-HULL-}3(z_1, z_2, z_3) \)
2. \( \text{do } i = 4 \text{ to } n \) \( n-3 \) times
3. \( \text{if OUTSIDE}(z_i, CH) \text{ then} \)
4. \( CH = \text{INSERT}(z_i, CH) \)

\( c_1 \)
\( c_2 \text{ n-3 times} \)
\( c_3(i-1) \text{ n-3 times} \)
\( c_4(i'-3) \text{ n'-3 times} \)
Convex Hull by Insertion

CONVEX-HULL-BY-INSERTION(Z)

1. \( \text{CH} = \text{CONVEX-HULL-3}(z_1, z_2, z_3) \)  
   \( \text{c1} \)
2. \( \text{do } i = 4 \text{ to } n \)  
   \( \text{c2 } n-3 \text{ times} \)
3. \( \text{if OUTSIDE}(z_i, \text{CH}) \) then  
   \( \text{c3}(i-1) \text{ n-3 times} \)
4. \( \text{CH} = \text{INSERT}(z_i, \text{CH}) \)  
   \( \text{c4}(i'-3) \text{ n’-3 times} \)
Convex Hull by Insertion

CONVEX-HULL-BY-INSERTION(Z)

1. CH = CONVEX-HULL-3(z1,z2,z3)  \( c_1 \)
2. do i = 4 to n  \( c_2 \) \( n-3 \) times
3. if OUTSIDE(zi,CH) then  \( c_3(i-1) \) \( n-3 \) times
4. CH = INSERT(zi,CH)  \( c_4(i’-3) \) \( n’-3 \) times

- Correct?
- Best case?
- Worst case?
Analysis of Algorithms
Prediction of Resources Required by an Algorithm

- Time.
- Space.
- As a function of instance size.
- Worst-case versus average case? Determining convex hull using CONVEX-HULL-BY-INSERTION can differ significantly even for problem instances of the same size.
- The same applies to INSERTION-SORT and to most algorithms for any computational problem.

- Pros for worst-case analysis:
  - Upper bound for all instances.
  - Occurs often for many problems (e.g., data base search).
  - Average case often takes as long as worst-case (e.g., INSERTION-SORT).

- Contras for average case analysis:
  - What is average input?
  - Complicated math.

- Growth of the function is of interest.
Random-access Machine

- Finite program.

- Finite collection of registers, each of which can store a single integer or real number.

- Memory consisting of a finite array of words.

- In one step (constant time) random-access machine can
  - perform a single arithmetic, logical or pointer operation on the content of a specified register,
  - fetch into a specified register the content of a word whose address is in a register,
  - store the contents of a register in a word whose address is in another register.
Models of Computation

- **Sequential**: One operation at a time.

- **Deterministic**: Future behaviour of the machine is uniquely determined by its present configuration.
Models of Computation

- **Sequential**: One operation at a time.
- **Deterministic**: future behaviour of the machine is uniquely determined by its present configuration.

Cost Measures

- **Uniform cost measure**: charge for each operation the same (independent of the size of numbers involved).
- **Logarithmic cost measure**: charge for each operation time proportional to the number of bits needed to represent operands. In particular important in numerical algorithms.
Models of Computation

- **Sequential**: One operation at a time.

- **Deterministic**: future behaviour of the machine is uniquely determined by its present configuration.

Cost Measures

- **Uniform cost measure**: charge for each operation the same (independent of the size of numbers involved).

- **Logarithmic cost measure**: charge for each operation time proportional to the number of bits needed to represent operands. In particular important in numerical algorithms.

Complexity Measures

- **Static**: independent of the input values (e.g., program length).

- **Dynamic**: running time and/or storage space as a function of instance size. Storage space is usually (but not always) a linear function of the instance size.
Search in a Sorted Set

- Linear Search requires $cn + d$ operations.

- Binary Search requires $a \log_2(n + 1) + b$ operations.
Asymptotic Running Times

- $f, g$: real-valued functions of nonnegative variables,
- $c, d$: positive constants,
- $f$ is $O(g)$ iff $\exists c \exists n_0 \forall n \geq n_0 : 0 \leq f(n) \leq cg(n)$.
- $f$ is $\Omega(g)$ iff $\exists d \exists n_0 \forall n \geq n_0 : f(n) \geq dg(n) \geq 0$.
- $f$ is $\Theta(g)$ iff $f$ is $O(g)$ and $\Omega(g)$.

- If an algorithm takes $O(g)$ in the worst case, this means that all but some small problem instances are solved in at most $cg(n)$ for appropriately chosen constant $c$. Bound from above.
- If an algorithm takes $\Omega(g)$ in the worst case, this means that for all but some small problem instances, there is an instance which requires at least $dg(n)$ time for appropriately chosen constant $d$. Bound from below.
Asymptotic Running Times - Example

- $\frac{1}{2}n^2 - 3n$ is $\Theta(n^2)$

- Determine $c$ and $d$ such that

$$dn^2 \leq \frac{1}{2}n^2 - 3n \leq cn^2$$

for all $n \geq n_0$

- Choose $c = \frac{1}{2}$, $d = \frac{1}{14}$, and $n_0 = 7$
Average Running Times

- Averaging over the possible inputs.
- Probabilistic choices: for worst-case input data take the average over possible choices.
- Amortization: average over a worst-case sequence of operations.

Stack Operations - Example

- Each operation consists of 0 or more pops followed by 1 push.
- $i$-th operation may in the worst case require $i - 1$ pops followed by 1 push.
- Disregarding that a number of pops in one operation influences the number of pops in the following operations, the worst case time complexity bound for $m$ operations will be

$$\sum_{i=1}^{m} i = \frac{m(m + 1)}{2}$$

- If the dependences between operations are taken into account, much better bound can be obtained by observing that the total number of pops cannot exceed the total number of pushes. Hence the bound is $2m$.

<table>
<thead>
<tr>
<th>$m$</th>
<th>worst-case</th>
<th>amortized</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>55</td>
<td>20</td>
</tr>
<tr>
<td>100</td>
<td>5050</td>
<td>200</td>
</tr>
<tr>
<td>1000</td>
<td>500500</td>
<td>2000</td>
</tr>
</tbody>
</table>
**Efficient Algorithms**

- Polynomial algorithms: bounded by a polynomial function of the input size.

- Problems for which polynomial algorithms exist are said to be *tractable*. Other problems are said to be *intractable*.

- The set of tractable problems is denoted by P.

- The usability of polynomial algorithms degrades *gracefully* with the problem size. Non-polynomial algorithms become useless very rapidly. Providing faster machines does not really help.

- Polynomial algorithms exploit some underlying structure in the problem. Non-polynomial algorithms usually require brute-force search.
### Running Time Estimates

- One step takes one microsecond.

<table>
<thead>
<tr>
<th>SIZE</th>
<th>COMPL.</th>
<th>20</th>
<th>50</th>
<th>100</th>
<th>200</th>
<th>500</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1000n$</td>
<td></td>
<td>0.02s</td>
<td>0.05s</td>
<td>0.1s</td>
<td>0.2s</td>
<td>0.5s</td>
<td>1s</td>
</tr>
<tr>
<td>$10^3n \log n$</td>
<td></td>
<td>0.09s</td>
<td>0.3s</td>
<td>0.6s</td>
<td>1.5s</td>
<td>4.5s</td>
<td>10s</td>
</tr>
<tr>
<td>$100n^2$</td>
<td></td>
<td>0.04s</td>
<td>0.25</td>
<td>1s</td>
<td>4s</td>
<td>25s</td>
<td>2m</td>
</tr>
<tr>
<td>$10n^3$</td>
<td></td>
<td>0.02s</td>
<td>1s</td>
<td>10s</td>
<td>1m</td>
<td>21m</td>
<td>2.7h</td>
</tr>
<tr>
<td>$n^{\log n}$</td>
<td></td>
<td>0.4s</td>
<td>1.1h</td>
<td>220d</td>
<td>125c</td>
<td>5\times10^{10}y</td>
<td></td>
</tr>
<tr>
<td>$2^{n/3}$</td>
<td></td>
<td>0.0001s</td>
<td>0.1s</td>
<td>2.7h</td>
<td>3\times10^6y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$2^n$</td>
<td></td>
<td>1s</td>
<td>35y</td>
<td>3\times10^6y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$3^n$</td>
<td></td>
<td>58m</td>
<td>2\times10^{11}y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Maximum Size of a Solvable Problem

<table>
<thead>
<tr>
<th>TIME COMPLEXITY</th>
<th>$1_s$</th>
<th>$10^2_s$</th>
<th>$10^4_s$</th>
<th>$10^6_s$</th>
<th>$10^8_s$</th>
<th>$10^{10}s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1000n$</td>
<td>$10^3$</td>
<td>$10^5$</td>
<td>$10^7$</td>
<td>$10^9$</td>
<td>$10^{11}$</td>
<td>$10^{13}$</td>
</tr>
<tr>
<td>$1000n \log n$</td>
<td>140</td>
<td>$8 \times 10^3$</td>
<td>$5 \times 10^5$</td>
<td>$4 \times 10^7$</td>
<td>$3 \times 10^9$</td>
<td>$3 \times 10^{11}$</td>
</tr>
<tr>
<td>$100n^2$</td>
<td>100</td>
<td>$10^3$</td>
<td>$10^4$</td>
<td>$10^5$</td>
<td>$10^6$</td>
<td>$10^7$</td>
</tr>
<tr>
<td>$10n^3$</td>
<td>46</td>
<td>210</td>
<td>$10^3$</td>
<td>$5 \times 10^3$</td>
<td>$2 \times 10^4$</td>
<td>$10^5$</td>
</tr>
<tr>
<td>$n^{\log n}$</td>
<td>22</td>
<td>36</td>
<td>54</td>
<td>79</td>
<td>112</td>
<td>156</td>
</tr>
<tr>
<td>$2^{n/3}$</td>
<td>59</td>
<td>79</td>
<td>99</td>
<td>119</td>
<td>139</td>
<td>159</td>
</tr>
<tr>
<td>$2^n$</td>
<td>19</td>
<td>26</td>
<td>33</td>
<td>39</td>
<td>46</td>
<td>53</td>
</tr>
<tr>
<td>$3^n$</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>25</td>
<td>29</td>
<td>33</td>
</tr>
</tbody>
</table>
Recurrences

- Substitution method.
- Iterative method.
- Recursive-tree method.
- Master theorem.
Substitution Method

- Make a qualified guess.
- Substitute in the recurrence relation.
- Prove.

- Recurrence relation: $T(n) = 2T(\lfloor n/2 \rfloor) + n$.
- Solution guess: $T(n) = O(n \log n)$ which means that $T(n) \leq cn \log n$ for an appropriately chosen constant $c > 0$.

$$
T(n) \leq 2(c \lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor)) + n \\
\leq cn \log(\lfloor n/2 \rfloor) + n \\
\leq cn \log n - cn \log 2 + n \\
\equiv cn \log n - cn + n \\
\leq cn \log n (\text{if } c \geq 1)
$$

- Boundary condition? See in the book (Section 4.1)!
Algorithms

Introduction

Iteration Method

- Expand the recurrence.
- Use algebra to bound.

Towers of Hanoi

\[
\text{HANOI}(n, \text{from}, \text{to}, \text{spare}) \quad \text{if} \quad (n > 0) \quad \text{then} \\
\quad \text{HANOI}(n-1, \text{from}, \text{spare}, \text{to}); \\
\quad \text{move} (\text{from}, \text{to}); \\
\quad \text{HANOI}(n-1, \text{spare}, \text{to}, \text{from});
\]
Iteration Method

- Expand the recurrence.
- Use algebra to bound.

**Towers of Hanoi**

HANOI(n, from, to, spare)
  
  if (n > 0) then
  
  HANOI(n-1, from, spare, to);
  move(from, to);
  HANOI(n-1, spare, to, from);

  \[
  T(n) = \begin{cases} 
    0 & \text{if } n = 0 \\
    2T(n-1) + 1 & \text{if } n > 0
  \end{cases}
  \]

  \[
  T(n) = 2T(n-1) + 1 \\
  = 2(2T(n-2) + 1) + 1 \\
  = 2(2(2T(n-3) + 1) + 1) + 1 \\
  = 2^3T(n-3) + 2^2 + 2^1 + 2^0 \\
  = 2^n0 + 2^{n-1} + 2^{n-2} + \ldots + 2^0 \\
  = \sum_{i=0}^{n-1} 2^i \\
  = 2^n - 1
  \]
Recursion-Tree Method
Sorting Using Divide-and-Conquer

- Divide the instance into two or more smaller instances of the same problem.
- Solve smaller problems recursively.
- Combine solutions to obtain a solution to the original instance.
Master Theorem

- $T(n) = aT(n/b) + f(n)$, $a \geq 1$, $b > 1$,

- $T(n)$ can be bounded asymptotically as follows:
  - If $f(n) = O(n^{\log_b a} / n^\varepsilon)$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
  - If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$.
  - If $f(n) = \Omega(n^{\log_b a} n^\varepsilon)$ for some constant $\varepsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large $n$, then $T(n) = \Theta(f(n))$.

- General idea: $f(n)$ is compared with $n^{\log_b a}$.
  - if $f(n)$ (polynomially) smaller, then $T(n) = \Theta(n^{\log_b a})$.
  - if $f(n)$ (polynomially) greater, then $T(n) = \Theta(f(n))$.
  - same size, then $T(n) = \Theta(f(n))$.

- $T(n) = 9T(n/3) + n$: $a = 9$, $b = 3$, $f(n) = n$, $n^{\log_b a} = n^{\log_3 9} = n^2$. Since $f(n) = O(n^{\log_3 9 - \varepsilon}) = O(n)$ for $\varepsilon = 1$, case 1 implies that $T(n) = \Theta(n^2)$. 
Master Theorem

- \( T(n) = aT(n/b) + f(n) \), \( a \geq 1, b > 1 \),

- \( T(n) \) can be bounded asymptotically as follows:
  - If \( f(n) = O(n^{\log_b a} / n^\varepsilon) \) for some constant \( \varepsilon > 0 \), then \( T(n) = \Theta(n^{\log_b a}) \).
  - If \( f(n) = \Theta(n^{\log_b a}) \), then \( T(n) = \Theta(n^{\log_b a} \log n) \).
  - If \( f(n) = \Omega(n^{\log_b a} n^\varepsilon) \) for some constant \( \varepsilon > 0 \), and if \( af(n/b) \leq cf(n) \) for some constant \( c < 1 \) and all sufficiently large \( n \), then \( T(n) = \Theta(f(n)) \).

- General idea: \( f(n) \) is compared with \( n^{\log_b a} \).
  - if \( f(n) \) (polynomially) smaller, then \( T(n) = \Theta(n^{\log_b a}) \).
  - if \( f(n) \) (polynomially) greater, then \( T(n) = \Theta(f(n)) \).
  - same size, then \( T(n) = \Theta(n^2) \).

- \( T(n) = 9T(n/3) + n: a = 9, b = 3, f(n) = n, n^{\log_b a} = n^{\log_3 9} = n^2. \) Since \( f(n) = O(n^{\log_3 9 - \varepsilon}) = O(n) \) for \( \varepsilon = 1 \), case 1 implies that \( T(n) = \Theta(n^2) \).

- \( T(n) = T(2n/3) + 1: a = 1, b = 3/2, f(n) = 1, n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1. \) Since \( f(n) = \Theta(1) \), case 2 implies that \( T(n) = \Theta(\log n) \).
Master Theorem

- \( T(n) = aT(n/b) + f(n), \ a \geq 1, \ b > 1, \)

- \( T(n) \) can be bounded asymptotically as follows:
  - If \( f(n) = O(n^{\log_b a}/n^\varepsilon) \) for some constant \( \varepsilon > 0 \), then \( T(n) = \Theta(n^{\log_b a}). \)
  - If \( f(n) = \Theta(n^{\log_b a}) \), then \( T(n) = \Theta(n^{\log_b a} \log n). \)
  - If \( f(n) = \Omega(n^{\log_b a}/n^\varepsilon) \) for some constant \( \varepsilon > 0 \), and if \( af(n/b) \leq cf(n) \) for some constant \( c < 1 \) and all sufficiently large \( n \), then \( T(n) = \Theta(f(n)). \)

- General idea: \( f(n) \) is compared with \( n^{\log_b a}. \)
  - if \( f(n) \) (polynomially) smaller, then \( T(n) = \Theta(n^{\log_b a}). \)
  - if \( f(n) \) (polynomially) greater, then \( T(n) = \Theta(f(n)). \)
  - same size, then \( T(n) = \Theta(f(n)). \)

- \( T(n) = 9T(n/3) + n: a = 9, b = 3, f(n) = n, n^{\log_b a} = n^{\log_3 9} = n^2. \) Since \( f(n) = O(n^{\log_3 9 - \varepsilon}) = O(n) \) for \( \varepsilon = 1 \), case 1 implies that \( T(n) = \Theta(n^2). \)

- \( T(n) = T(2n/3) + 1: a = 1, b = 3/2, f(n) = 1, n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1. \) Since \( f(n) = \Theta(1), \) case 2 implies that \( T(n) = \Theta(\log n). \)

- \( T(n) = 3T(n/4) + n \log n: a = 3, b = 4, f(n) = n \log n, n^{\log_b a} = O(n^{0.793}). \) Since \( f(n) = \Omega(n^{\log_4 3+\varepsilon}), \) where \( \varepsilon \approx 0.2, \) case 3 implies that \( T(n) = \Theta(n \log n). \)
Summary

- Algorithm Design - Paradigms
- Algorithm Correctness
- Algorithm Complexity