2. **Algorithm Analysis**

- computational tractability
- asymptotic order of growth
- implementing Gale–Shapley
- survey of common running times

---

**A strikingly modern thought**

“As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise—By what course of calculation can these results be arrived at by the machine in the shortest time?” — Charles Babbage (1864)

---

**Models of computation: Turing machines**

**Deterministic Turing machine.** Simple and idealistic model.

**Running time.** Number of steps.

**Memory.** Number of tape cells utilized.

**Caveat.** No random access of memory.

- Single-tape TM requires \( \geq n^2 \) steps to detect \( n \)-bit palindromes.
- Easy to detect palindromes in \( \leq cn \) steps on a real computer.
Models of computation: word RAM

Word RAM.
- Each memory location and input/output cell stores a $w$-bit integer.
- Primitive operations: arithmetic/logic operations, read/write memory, array indexing, following a pointer, conditional branch, …

Running time. Number of primitive operations.
Memory. Number of memory cells utilized.

Caveat. At times, need more refined model (e.g., multiplying $n$-bit integers).

Polynomial running time

Desirable scaling property. When the input size doubles, the algorithm should slow down by at most some constant factor $C$.

Def. An algorithm is poly-time if the above scaling property holds.

There exist constants $c > 0$ and $d > 0$ such that, for every input of size $n$, the running time of the algorithm is bounded above by $c n^d$ primitive computational steps.

Choose $C = 2^d$

Polynomial running time

We say that an algorithm is efficient if it has a polynomial running time.

Theory. Definition is (relatively) insensitive to model of computation.

Practice. It really works!
- The poly-time algorithms that people develop have both small constants and small exponents.
- Breaking through the exponential barrier of brute force typically exposes some crucial structure of the problem.

Exceptions. Some poly-time algorithms in the wild have galactic constants and/or huge exponents.

Q. Which would you prefer: $20n^{120}$ or $n^{0.02 \ln n}$ ?

Brute force

Brute force. For many nontrivial problems, there is a natural brute-force search algorithm that checks every possible solution.
- Typically takes $2^n$ steps (or worse) for inputs of size $n$.
- Unacceptable in practice.
Worst-case analysis

Worst case. Running time guarantee for any input of size $n$.
- Generally captures efficiency in practice.
- Draconian view, but hard to find effective alternative.

Exceptions. Some exponential-time algorithms are used widely in practice because the worst-case instances don’t arise.

Big O notation

Upper bounds. $f(n)$ is $O(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

Ex. $f(n) = 32n^2 + 17n + 1$.
- $f(n)$ is $O(n^2)$.
- $f(n)$ is neither $O(n)$ nor $O(n \log n)$.

Typical usage. Insertion sort makes $O(n^2)$ compares to sort $n$ elements.
Let \( f(n) = 3n^2 + 17n \log_2 n + 1000 \). Which of the following are true?

A. \( f(n) \) is \( O(n^2) \).

B. \( f(n) \) is \( O(n^3) \).

C. Both A and B.

D. Neither A nor B.

Analysis of algorithms: quiz 1

Big \( O \) notation: properties

**Reflexivity.** \( f \) is \( O(f) \).

**Constants.** If \( f \) is \( O(g) \) and \( c > 0 \), then \( cf \) is \( O(g) \).

**Products.** If \( f_1 \) is \( O(g_1) \) and \( f_2 \) is \( O(g_2) \), then \( f_1f_2 \) is \( O(g_1g_2) \).

**Pf.**

- \( \exists c_1 > 0 \) and \( n_1 \geq 0 \) such that \( 0 \leq f_1(n) \leq c_1 \cdot g_1(n) \) for all \( n \geq n_1 \).
- \( \exists c_2 > 0 \) and \( n_2 \geq 0 \) such that \( 0 \leq f_2(n) \leq c_2 \cdot g_2(n) \) for all \( n \geq n_2 \).
- Then, \( 0 \leq f_1(n) \cdot f_2(n) \leq c_1 \cdot c_2 \cdot g_1(n) \cdot g_2(n) \) for all \( n \geq \max \{ n_1, n_2 \} \).

**Sums.** If \( f_1 \) is \( O(g_1) \) and \( f_2 \) is \( O(g_2) \), then \( f_1 + f_2 \) is \( O(\max \{ g_1, g_2 \}) \).

**Transitivity.** If \( f \) is \( O(g) \) and \( g \) is \( O(h) \), then \( f \) is \( O(h) \).

**Ex.** \( f(n) = 5n^3 + 3n^2 + n + 1234 \) is \( O(n^3) \).

Bottom line. OK to abuse notation in this way; not OK to misuse it.

Big \( O \) notational abuses

**One-way “equality.”** \( O(g(n)) \) is a set of functions, but computer scientists often write \( f(n) = O(g(n)) \) instead of \( f(n) \in O(g(n)) \).

**Ex.** Consider \( g_1(n) = 5n^3 \) and \( g_2(n) = 3n^2 \).

- We have \( g_1(n) \in O(n^3) \) and \( g_2(n) \in O(n^2) \).
- But, do not conclude \( g_1(n) = g_2(n) \).

**Domain and codomain.** \( f \) and \( g \) and real-valued functions.

- The domain is typically the natural numbers: \( \mathbb{N} \rightarrow \mathbb{R} \).
- Sometimes we extend to the reals: \( \mathbb{R}_{\geq 0} \rightarrow \mathbb{R} \).
- Or restrict to a subset.

Typical usage. Any compare-based sorting algorithm requires \( \Omega(n \log n) \) compares in the worst case.

Vacuous statement. Any compare-based sorting algorithm requires at least \( O(n \log n) \) compares in the worst case.
Analysis of algorithms: quiz 2

Which is an equivalent definition of big Omega notation?

A. \( f(n) \) is \( \Omega(g(n)) \) iff \( g(n) \) is \( O(f(n)) \).

B. \( f(n) \) is \( \Omega(g(n)) \) iff there exist constants \( c > 0 \) such that \( f(n) \geq c \cdot g(n) \geq 0 \) for infinitely many \( n \).

C. Both A and B.

D. Neither A nor B.

Analysis of algorithms: quiz 3

Which is an equivalent definition of big Theta notation?

A. \( f(n) \) is \( \Theta(g(n)) \) iff \( f(n) \) is both \( O(g(n)) \) and \( \Omega(g(n)) \).

B. \( f(n) \) is \( \Theta(g(n)) \) iff \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = c \) for some constant \( 0 < c < \infty \).

C. Both A and B.

D. Neither A nor B.

Asymptotic bounds and limits

**Proposition.** If \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = c \) for some constant \( 0 < c < \infty \), then \( f(n) \) is \( \Theta(g(n)) \).

**Pf.**

- By definition of the limit, there exists \( n_0 \) such that for all \( n \geq n_0 \)

\[
\frac{1}{2} c \leq \frac{f(n)}{g(n)} \leq 2c
\]

- Multiplying by \( g(n) \) yields \( \frac{1}{2} c \cdot g(n) \leq f(n) \leq 2 c \cdot g(n) \) for all \( n \geq n_0 \).

Thus, \( f(n) \) is \( \Theta(g(n)) \) by definition, with \( c_1 = \frac{1}{2} c \) and \( c_2 = 2c \).

**Proposition.** If \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \), then \( f(n) \) is \( O(g(n)) \) but not \( \Theta(g(n)) \).
Asymptotic bounds for some common functions

**Polynomials.** Let \( f(n) = a_0 + a_1 n + \ldots + a_d n^d \) with \( a_d > 0 \). Then, \( f(n) \) is \( \Theta(n^d) \).

\[
\lim_{n \to \infty} \frac{a_0 + a_1 n + \ldots + a_d n^d}{n^d} = a_d > 0
\]

**Logarithms.** \( \log_a n \) is \( \Theta(\log_b n) \) for every \( a > 1 \) and every \( b > 1 \).

\[
\frac{\log_a n}{\log_b n} = \frac{1}{\log_b a} \quad \text{no need to specify base (assuming it is a constant)}
\]

**Logarithms and polynomials.** \( \log_a n \) is \( O(n^d) \) for every \( a > 0 \) and every \( d > 0 \).

\[
\lim_{n \to \infty} \frac{\log_a n}{n^d} = 0
\]

**Exponentials and polynomials.** \( n^d \) is \( O(r^n) \) for every \( r > 1 \) and every \( d > 0 \).

\[
\lim_{n \to \infty} \frac{n^d}{r^n} = 0
\]

**Factorials.** \( n! = 2^{\Theta(n \log n)} \).

\[
Pf. \text{ Stirling's formula: } n! \sim \sqrt{2\pi n} \left( \frac{n}{e} \right)^n
\]

---

### 2. ALGORITHM ANALYSIS

- computational tractability
- asymptotic order of growth
- implementing Gale–Shapley
- survey of common running times

**Big O notation with multiple variables**

**Upper bounds.** \( f(m, n) \) is \( O(g(m, n)) \) if there exist constants \( c > 0, m_0 \geq 0, \) and \( n_0 \geq 0 \) such that \( f(m, n) \leq c \cdot g(m, n) \) for all \( n \geq n_0 \) and \( m \geq m_0 \).

**Ex.** \( f(m, n) = 32mn^2 + 17mn + 32n^3 \).
- \( f(m, n) \) is both \( O(mn^2 + n^3) \) and \( O(mn^3) \).
- \( f(m, n) \) is neither \( O(n^3) \) nor \( O(mn^2) \).

**Typical usage.** Breadth-first search takes \( O(m + n) \) time to find a shortest path from \( s \) to \( t \) in a digraph with \( n \) nodes and \( m \) edges.

### Efficient implementation

**Goal.** Implement Gale–Shapley to run in \( O(n^2) \) time.

**Gale–Shapley** (preference lists for \( n \) hospitals and \( n \) students)

**Initialize** \( M \) to empty matching.

**While** (some hospital \( h \) is unmatched)
- \( s \leftarrow \) first student on \( h \)'s list to whom \( h \) has not yet proposed.

**If** \( s \) is unmatched
- Add \( h \rightarrow s \) to matching \( M \).

**Else If** \( s \) prefers \( h \) to current partner \( h' \)
- Replace \( h' \rightarrow s \) with \( h \rightarrow s \) in matching \( M \).

**Else**
- \( s \) rejects \( h \).

**Return** stable matching \( M \).
Efficient implementation

**Goal.** Implement Gale–Shapley to run in $O(n^2)$ time.

**Representing hospitals and students.** Index hospitals and students 1, …, $n$.

**Representing the matching.**
- Maintain two arrays $\text{student}[h]$ and $\text{hospital}[s]$.
  - if $h$ matched to $s$, then $\text{student}[h] = s$ and $\text{hospital}[s] = h$
  - use value 0 to designate that hospital or student is unmatched
- Can add/remove a pair from matching in $O(1)$ time.
- Maintain set of unmatched hospitals in a queue (or stack).
- Can find an unmatched hospital in $O(1)$ time.

---

Data representation: making a proposal

**Hospital makes a proposal.**
- Key operation: find hospital's next favorite student.
- For each hospital: maintain a list of students, ordered by preference.
- For each hospital: maintain a pointer to student for next proposal.

![Graph of matching](image)

**Bottom line.** Making a proposal takes $O(1)$ time.

---

Data representation: accepting/rejecting a proposal

**Student accepts/rejects a proposal.**
- Does student $s$ prefer hospital $h$ to hospital $h'$?
- For each student, create inverse of preference list of hospitals.

<table>
<thead>
<tr>
<th>pref[i]</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
<th>5th</th>
<th>6th</th>
<th>7th</th>
<th>8th</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

\[\text{student prefers hospital 4 to 6 since rank}[4] < \text{rank}[6]\]

<table>
<thead>
<tr>
<th>rank[i]</th>
<th>4th</th>
<th>8th</th>
<th>2nd</th>
<th>5th</th>
<th>6th</th>
<th>7th</th>
<th>3rd</th>
<th>1st</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

**Bottom line.** After $\Theta(n^2)$ preprocessing time (to create the $n$ ranking arrays), it takes $O(1)$ time to accept/reject a proposal.

---

Stable matching: summary

**Theorem.** Can implement Gale–Shapley to run in $O(n^2)$ time.

**Pf.**
- $\Theta(n^2)$ preprocessing time to create the $n$ ranking arrays.
- There are $O(n^2)$ proposals; processing each proposal takes $O(1)$ time.

**Theorem.** In the worst case, any algorithm to find a stable matching must query the hospital's preference list $\Omega(n^2)$ times.
2. **Algorithm Analysis**

- computational tractability
- asymptotic order of growth
- implementing Gale–Shapley
- survey of common running times

### Linear time

**Linear time.** Running time is $O(n)$.

**Merge two sorted lists.** Combine two sorted linked lists $A = a_1, a_2, \ldots, a_n$ and $B = b_1, b_2, \ldots, b_n$ into a sorted whole.

**$O(n)$ algorithm.** Merge in mergesort.

```
i ← 1; j ← 1.
WHILE (both lists are nonempty)
    IF ($a_i \leq b_j$) append $a_i$ to output list and increment $i$.
    ELSE append $b_j$ to output list and increment $j$.
Append remaining elements from nonempty list to output list.
```

### Target Sum

**TARGET-SUM.** Given a sorted array of $n$ distinct integers and an integer $T$, find two that sum to exactly $T$?

<table>
<thead>
<tr>
<th>–20</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>35</th>
<th>40</th>
<th>60</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T = 60$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
i
↑

j
↑
```
Logarithmic time

**Logarithmic time.** Running time is $O(\log n)$.

**Search in a sorted array.** Given a sorted array $A$ of $n$ numbers, is a given number $x$ in the array?

$O(\log n)$ algorithm. Binary search.

- Invariant: If $x$ is in the array, then $x$ is in $A[lo..hi]$.
- After $k$ iterations of **WHILE** loop, $(hi - lo + 1) \leq n / 2^k \Rightarrow k \leq 1 + \log_2 n$.

```plaintext
lo ← 1; hi ← n.

WHILE (lo ≤ hi)
    mid ← \(\lceil (lo + hi) / 2 \rceil\).
    IF (x < A[mid]) hi ← mid − 1.
    ELSE IF (x > A[mid]) lo ← mid + 1.
    ELSE RETURN yes.
RETURN no.
```

Linearithmic time

**Linearithmic time.** Running time is $O(n \log n)$.

**Sorting.** Given an array of $n$ elements, rearrange them in ascending order.

$O(n \log n)$ algorithm. Mergesort.

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
MERGESORT Example
EMERGESORT Example
EMERGESORT Example
EMERGESORT Example
EMERGESORT Example
EMERGESORT Example
EMERGESORT Example
EMERGESORT Example
EMERGESORT Example
EMERGESORT Example
EMERGESORT Example
EMERGESORT Example
EMERGESORT Example
EMERGESORT Example
EMERGESORT Example
```

Largest Empty Interval

**Largest-Empty-Interval.** Given $n$ timestamps $x_1, \ldots, x_n$ on which copies of a file arrive at a server, what is largest interval when no copies of file arrive?
**Quadratic time**

**Quadratic time.** Running time is \( O(n^2) \).

**Closest pair of points.** Given a list of \( n \) points in the plane \((x_1, y_1), \ldots, (x_n, y_n)\), find the pair that is closest to each other.

**\( O(n^2) \) algorithm.** Enumerate all pairs of points.

\[
\begin{align*}
\min & \leftarrow \infty, \\
\text{FOR} & \ i = 1 \ \text{TO} \ n \\
& \quad \text{FOR} \ j = i + 1 \ \text{TO} \ n \\
& \quad \quad d \leftarrow (x_i-x_j)^2 + (y_i-y_j)^2. \\
& \quad \quad \text{IF} \ (d < \min) \\
& \quad \quad \quad \min \leftarrow d.
\end{align*}
\]

**Remark.** \( \Omega(n^2) \) seems inevitable, but this is just an illusion. [see Chapter 5]

---

**Cubic time**

**Cubic time.** Running time is \( O(n^3) \).

**3-SUM.** Given an array of \( n \) distinct integers, find three that sum to 0.

**\( O(n^3) \) algorithm.** Enumerate all triples.

\[
\begin{align*}
\text{FOR} & \ i = 1 \ \text{TO} \ n \\
& \quad \text{FOR} \ j = i + 1 \ \text{TO} \ n \\
& \quad \quad \text{FOR} \ k = j + 1 \ \text{TO} \ n \\
& \quad \quad \quad \text{IF} \ (a_i + a_j + a_k = 0) \\
& \quad \quad \quad \quad \text{RETURN} \ (a_i, a_j, a_k).
\end{align*}
\]

**Remark.** \( \Omega(n^3) \) seems inevitable, but \( \Omega(n^2) \) is not hard.

---

**Polynomial time**

**Polynomial time.** Running time is \( O(n^k) \) for some constant \( k > 0 \).

**Independent set of size \( k \).** Given a graph, are there \( k \) nodes such that no two are joined by an edge?

**\( O(n^3) \) algorithm.** Enumerate all subsets of \( k \) nodes.

\[
\begin{align*}
\text{FOREACH} & \ \text{subset} \ S \ \text{of} \ k \ \text{nodes:} \\
& \ \quad \text{Check whether} \ S \ \text{is an independent set.} \\
& \ \quad \text{IF} \ (S \ \text{is an independent set}) \\
& \ \quad \quad \text{OUTPUT} \ S.
\end{align*}
\]

- Check whether \( S \) is an independent set of size \( k \) takes \( O(k^2) \) time.
- Number of \( k \)-element subsets = \( \binom{n}{k} = \frac{n(n-1)(n-2) \times \cdots \times (n-k+1)}{k(k-1)(k-2) \times \cdots \times 1} \leq \frac{n^k}{k!} \)

\( poly-time \) for \( k = 17 \), but not practical
Exponential time

**Exponential time.** Running time is $O(2^k)$ for some constant $k > 0$.

**Independent set.** Given a graph, what is maximum cardinality of an independent set?

**$O(n^2 2^n)$ algorithm.** Enumerate all subsets.

\[
S^* \leftarrow \emptyset.
\]

**FOREACH** subset $S$ of nodes:

  Check whether $S$ is an independent set.

  **IF** ($S$ is an independent set and $|S| > |S^*|$)

  $S^* \leftarrow S$.

  **RETURN** $S^*$.

---

Analysis of algorithms: quiz 4

Which is an equivalent definition of exponential time?

**A.** $O(2^n)$

**B.** $O(2^{cn})$ for some constant $c > 0$.

**C.** Both A and B.

**D.** Neither A nor B.

---

Analysis of algorithms: quiz 4 doesn't include $3n$ includes $3n$ but doesn't include $n! = 2^{\Theta(n \log n)}$