CCES – College of Computer Engineering and Sciences

Prince Sattam Bin Abdulaziz University

CS3401 – Algorithm Design and Analysis

Chapter 3

Brute Force & Exhaustive Search

Academic Year 2026 – 2027

Dr. Khalil Chebil

Dr. Esraa Eldesouky

Chapter Overview

Section 1

Introduction to Brute Force

What is Brute Force?

Definition

A **brute-force** algorithm solves a problem by following the problem's definition directly, without any clever optimisation — often by trying *all possibilities* and selecting the best.

Advantages

- **Widely applicable** — works for any problem
- **Simple to implement** — close to the definition
- **Good baseline** — reveals the problem's structure
- **Works for small n** — practical when $n \leq 20$
- **Guaranteed to find a solution** if one exists

Disadvantages

- **Often inefficient** for large inputs
- Can be exponential (2^n or $n!$) for combinatorial problems
- Rarely used in production for large-scale problems

When to use it

When n is small, when you need a correct reference implementation, or when no better algorithm is known. A baseline for comparing more advanced algorithms.

Simple Brute-Force Examples

Computing a^n

Multiply 1 by a exactly n times:

$$a^n = \underbrace{a \times a \times \cdots \times a}_{n \text{ times}}$$

Complexity: $\mathcal{O}(n)$ multiplications.

Better: fast exponentiation uses only $\mathcal{O}(\log n)$. The brute-force version is still correct.

Sum of first n natural numbers

Enumerate all numbers from 1 to n:

$$S = 1 + 2 + \cdots + n \implies \mathcal{O}(n)$$

But this is **not the cheapest solution** since:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \implies \mathcal{O}(1)$$

Lesson

Even for simple problems, brute force may not be optimal.

Section 2

Brute-Force Sorting

Selection Sort

Idea: Repeatedly find the minimum of the unsorted part and swap it to its correct position.

Algorithm

Input: Array $A[0..n-1]$

```
1: for i ← 0 to n − 2
2:     min ← i
3:     for j ← i+1 to n−1
4:         if A[j] < A[min] then
5:             min ← j
6:         end if
7:     end for
8:     swap A[i] and A[min]
9: end for
```

Trace on [89, 45, 68, 17]**:**

89	45	68	17
17	45	68	89
17	45	68	89
17	45	68	89

Complexity:

$$C(n) = \sum_{i=0}^{n-2}(n-1-i) = \frac{(n-1)n}{2} \in \Theta(n^2)$$

Stable: **No** In-place: Yes

- **Stable**: Equal elements remain in their relative order.
- **In-place**: Uses only a constant amount of extra memory.

Bubble Sort

Idea: Repeatedly scan the array, swapping adjacent elements that are out of order. The largest element "bubbles" to its place.

Algorithm

Input: Array $A[0..n-1]$

```
for i ← 0 to n−2
    for j ← 0 to n−2−i
        if A[j+1] < A[j] then
            swap A[j] and A[j+1]
        end if
    end for
end for
```

Key properties:

- After pass i, the i largest elements are in place
- Can be improved: stop early if no swap in a pass

Complexity:

$$C(n) = \sum_{i=0}^{n-2}(n-1-i) = \frac{n(n-1)}{2} \in \Theta(n^2)$$

	Worst	Best
Comparisons	$n^2/2$	$n-1$
Swaps	$n^2/4$	0

Stable: Yes

In-place: Yes

Section 3

String Matching

Brute-Force String Matching

Problem: Find pattern $P[0..m-1]$ in text $T[0..n-1]$.

Algorithm

```
1: for i ← 0 to n−m
2:     j ← 0
3:     while j < m and P[j] = T[i+j]
4:         j ← j + 1
5:     end while
6:     if j = m then return i
7:     end if
8: end for
9: return −1
```

Visual trace — pattern `NOT` in text:

```
N O B O D Y _ N O T I C E D _ H I
N O T
  T
    N O T
...
          N O T
```

Worst case: $\mathcal{O}(mn)$
Typical: $\Theta(n)$

Section 4

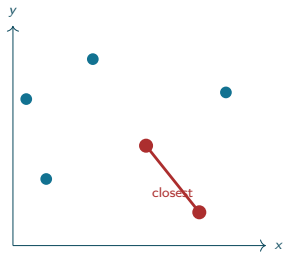
Geometric Brute Force

Closest Pair of Points

Problem: Given n points in the 2-D plane, find the two points with the minimum Euclidean distance.

Brute-Force Algorithm

1: $d_{\min} \leftarrow \infty$
2: **for** $i \leftarrow 1$ **to** $n-1$
3: **for** $j \leftarrow i+1$ **to** n
4: $d \leftarrow \sqrt{(x_i-x_j)^2 + (y_i-y_j)^2}$
5: **if** $d < d_{\min}$ **then** $d_{\min} \leftarrow d$
6: **end if**
7: **end for**
8: **end for**
9: **return** $d_{\min}$



Optimisation: Compare $(x_i-x_j)^2 + (y_i-y_j)^2$ instead of taking the square root — avoids costly `sqrt()` calls.

$\mathcal{O}(n^2)$

Convex Hull — Brute Force

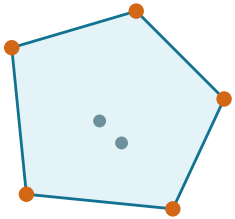
Definition

The **convex hull** of a set of points S is the smallest convex polygon enclosing all points.

Brute-force idea: A directed edge (p_i, p_j) is on the convex hull iff all other points lie on the *same side* of the line through p_i and p_j.
Test via cross product:

$$(p_j - p_i) \times (p_k - p_i) = (x_j - x_i)(y_k - y_i) - (x_k - x_i)(y_j - y_i)$$

- > 0: p_k is *left* of line $p_i p_j$
- < 0: p_k is *right* of line $p_i p_j$
- $= 0$: p_k is *on* line $p_i p_j$



Convex hull (orange vertices)

Interior points (dark)

Complexity: $\mathcal{O}(n^3)$ (check all $O(n^2)$ edges, each with $O(n)$ point tests)
More details `https://chebil.github.io/ConvexHull/`

Section 5

Exhaustive Search

Exhaustive Search — What & When

Definition

Exhaustive search is a brute-force approach to combinatorial problems: generate *all* candidate solutions, evaluate each, and keep the best.

Typical structure:

1. Generate all permutations / combinations / subsets of the input in a systematic way
2. Evaluate each candidate solution (discard infeasible ones)
3. Track the best feasible solution found so far
4. When done, return the best

When is it acceptable?

- $n \leq 20$ for subset problems ($2^{20} \approx 10^6$)
- $n \leq 12$ for permutation problems ($12! \approx 5 \times 10^8$)
- When no efficient algorithm is known
- As a correctness baseline

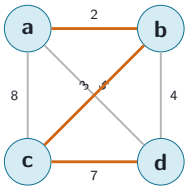
Travelling Salesman Problem (TSP)

Problem: Given n cities and distances between every pair, find the *shortest* tour that visits each city exactly once and returns to the start.

Brute-force: Try all $(n-1)!/2$ distinct tours.

Example (4 cities):

Tour	Cost
$a \to b \to c \to d \to a$	$2+3+7+5 = 17$
$a \to b \to d \to c \to a$	$2+4+7+8 = 21$
$a \to c \to b \to d \to a$	$8+3+4+5 = 20$
$a \to c \to d \to b \to a$	$8+7+4+2 = 21$
$a \to d \to b \to c \to a$	$5+4+3+8 = 20$
$a \to d \to c \to b \to a$	$5+7+3+2 = \mathbf{17}$



Optimal: cost 17

Exponential growth

$\Theta(n!)$ — for $n = 20$: $\approx 1.2 \times 10^{17}$ tours. At 10^9 tours/sec: 4 million years.

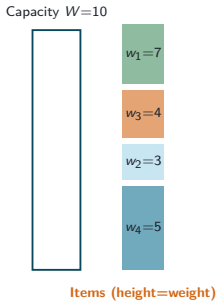
0-1 Knapsack Problem

Problem: n items with weights w_i and values v_i; knapsack capacity W. Select a subset to maximise total value without exceeding capacity.

Brute force: Examine all 2^n subsets.

Example ($W = 10$, 4 items):

Subset	Weight	Value	
$\{1\}$	7	42	
$\{2\}$	3	12	
$\{3,4\}$	9	**65**	✓ optimal
$\{1,2\}$	10	54	
$\{1,3\}$	11	—	infeasible



Complexity

$\mathcal{O}(2^n)$ subsets to check. For $n = 40$: $\approx 10^{12}$ subsets.

The Assignment Problem

Problem: Assign n workers to n jobs (one-to-one) to minimise total cost. Cost of worker i doing job j is $C[i, j]$.

Brute force: Try all $n!$ permutations of jobs.

Example ($n = 4$):

	Job 1	Job 2	Job 3	Job 4
P1	9	2	7	8
P2	6	4	3	7
P3	5	8	1	8
P4	7	6	9	4

Assignment $\langle 1, 2, 3, 4 \rangle$: $9 + 4 + 1 + 4 =$ 18 (optimal)

Brute-force complexity

Must check all $n!$ permutations: for $n = 20$ that is $\approx 2.4 \times 10^{18}$ assignments.

Better approach

The *Hungarian Algorithm* solves the Assignment Problem in $\mathcal{O}(n^3)$ — we will revisit this in Ch. 5.

This problem is NP-hard in general. The assignment problem is solvable in polynomial time because of its special structure.

Section 6

BFS — Exhaustive Graph Search

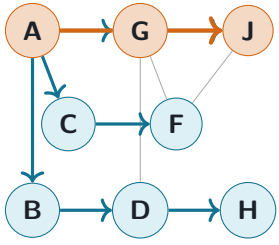
Shortest Path via Breadth-First Search

BFS Algorithm

Input: Graph G, source s, target t

```
1:  Enqueue s; parent[s] ← null
2:  while queue not empty
3:      v ← dequeue
4:      if v = t then break
5:      end if
6:      for each neighbor u of v
7:          if u ∉ parent then
8:              parent[u] ← v
9:              enqueue u
10:         end if
11:     end for
12: end while
13: Trace back via parent to get path
```

Complexity: $\mathcal{O}(|V| + |E|)$



Shortest path: $A \to G \to J$

Parents

Node	Parent
A	–
B	A
C	A
G	A
D	B
F	C
J	G
H	D

Queue trace (front at left)

Processed	Queue	Processed	Queue
Start	$[A]$	G	$[D, F, J]$
A	$[B, C, G]$	D	$[F, J, H]$
B	$[C, G, D]$	F	$[J, H]$
C	$[G, D, F]$	J	$[H]$; stop

Chapter 3 Summary

Brute-force algorithms covered:

- Selection sort, Bubble sort — $\Theta(n^2)$
- String matching — $\mathcal{O}(mn)$
- Closest pair — $\mathcal{O}(n^2)$
- Convex hull — $\mathcal{O}(n^3)$

Exhaustive search:

- TSP — $\Theta(n!)$
- 0-1 Knapsack — $\mathcal{O}(2^n)$
- Assignment — $\Theta(n!)$
- BFS shortest path — $\mathcal{O}(|V| + |E|)$

Key insight

Brute force is a *starting point*, not a destination. Understanding it clarifies what a better algorithm must improve upon.

Next chapter

Chapter 4: Decrease & Conquer + Divide & Conquer — recursive strategies that break problems into smaller (or equal) sub-instances for dramatic efficiency gains.