Brute Force

(Chapter 3)

Brute Force Technique

- The “obvious and straightforward” approach for solving a problem
- Not really trying to be efficient
- Typically these are easy to implement
- “Force” comes from using computer power not intellectual power or “Just do it!”

Brute Force Examples

What is the brute force solution for these:

1. Computing a^n ($a > 0$, n a nonnegative integer)

   Loop through *n*, multiply by *a* each time
   O(n)

2. Computing n!

   Loop *i* through *n*, multiply by *i* each time
   O(n)

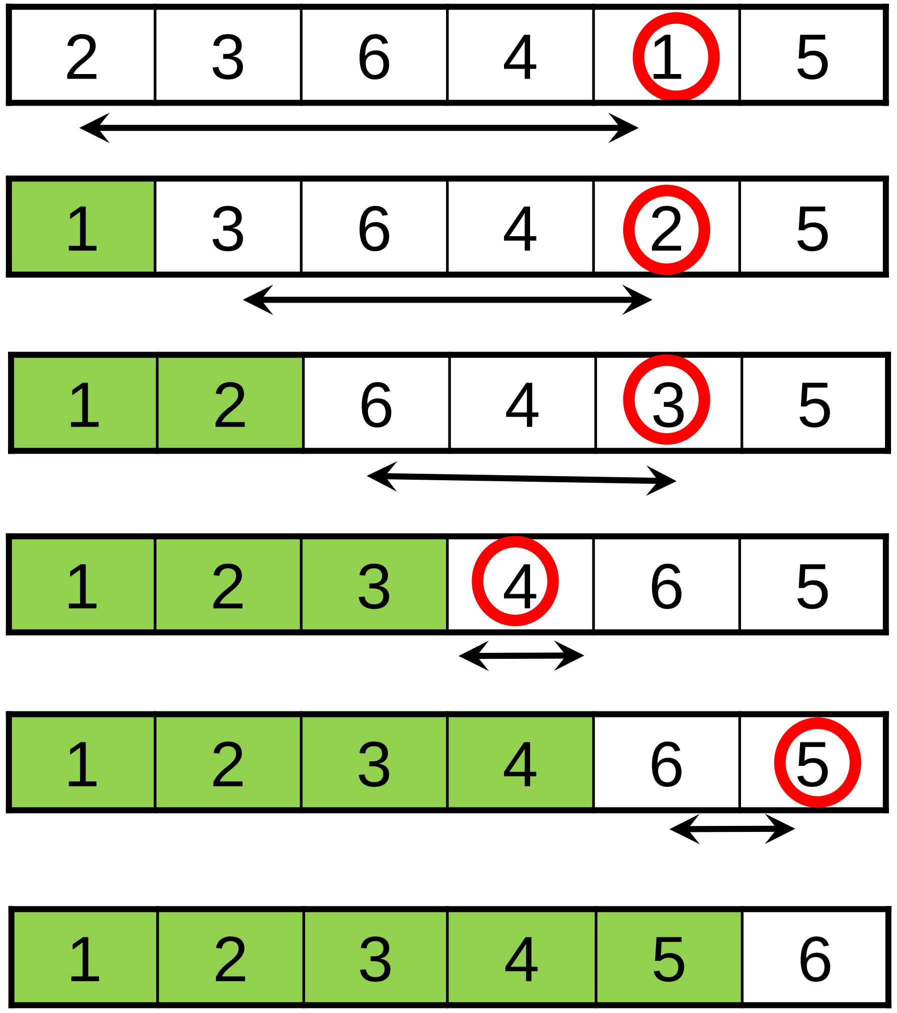
3. Searching for a key of a given value in a list

   Loop *i* through *n*, check if position *i* is the key.
   O(n)

Brute Force

1. Sorting problem
   - Selection sort
   - Bubble sort
2. String matching
3. Optimization problems
   - Knapsack problem
   - Assignment Problem

Selection Sort



Selection Sort

1. Scan the array to find the smallest element.
2. Swap it with the first element.
3. Repeat for the second element.
4. Generally: on pass *i*, find the smallest element in *A[i..n-1]* and swap it with *A[i]*.

Selection Sort (pseudocode)

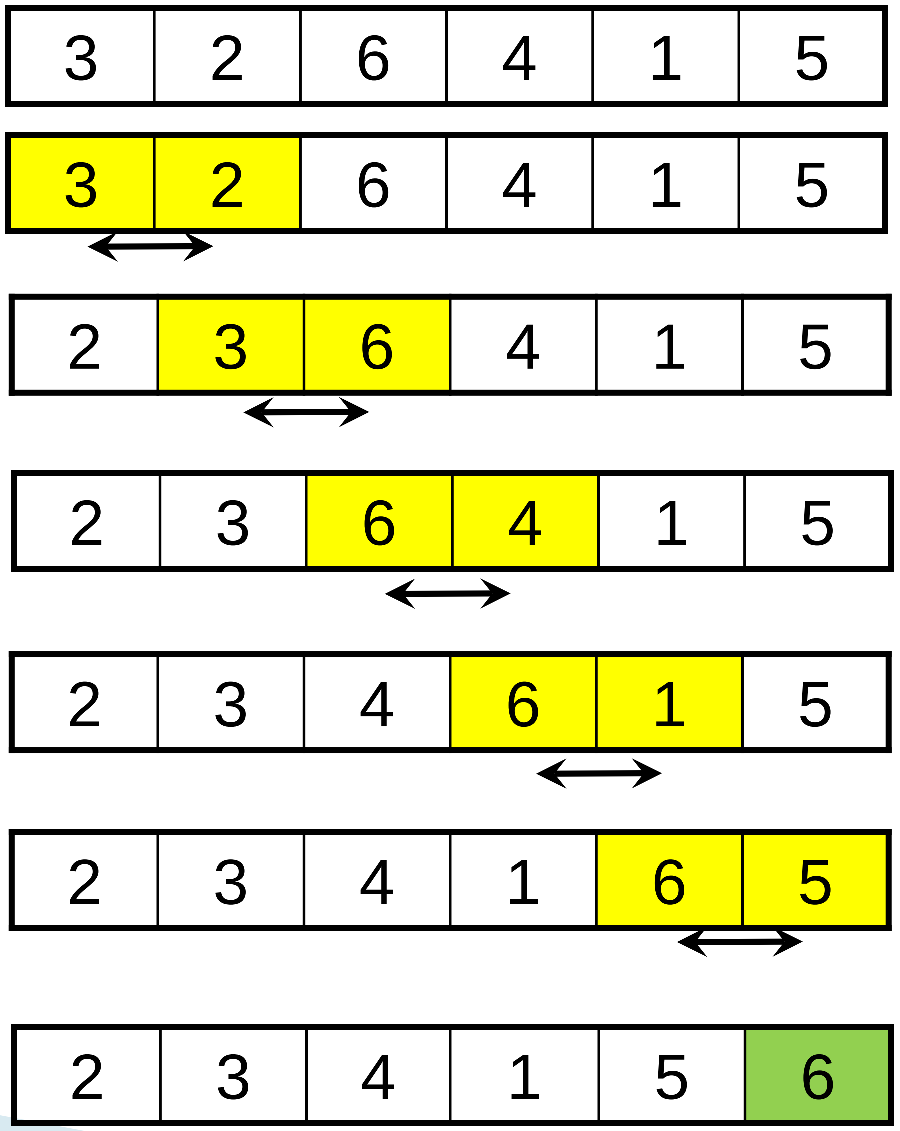
```
ALGORITHM  $SelectionSort(A[0..n-1])$
    //Sorts a given array by selection sort
    //Input: An array $A[0..n-1]$ of orderable elements
    //Output: Array $A[0..n-1]$ sorted in nondecreasing order
    for $i \leftarrow 0$ to $n-2$ do
        $min \leftarrow i$
        for $j \leftarrow i+1$ to $n-1$ do
            if $A[j] < A[min]$
                $min \leftarrow j$
        swap $A[i]$ and $A[min]$
```

Efficiency? $O(n^2)$

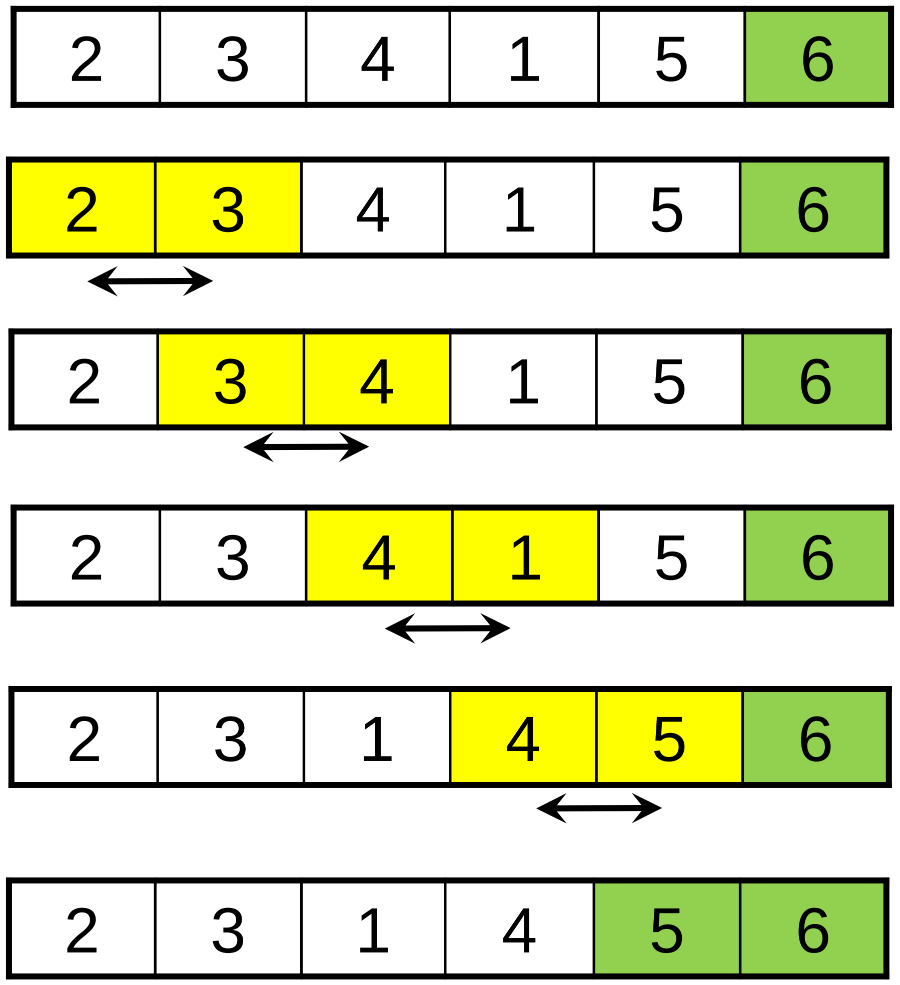
Brute Force

1. Sorting problem
   - Selection sort
   - Bubble sort
2. String matching
3. Optimization problems
   - Knapsack problem
   - Assignment Problem

Bubble sort



Bubble sort



Bubble sort

```
ALGORITHM  BubbleSort(A[0..n − 1])
    //Sorts a given array by bubble sort
    //Input: An array A[0..n − 1] of orderable elements
    //Output: Array A[0..n − 1] sorted in nondecreasing order
    for i ← 0 to n − 2 do
        for j ← 0 to n − 2 − i do
            if A[j + 1] < A[j]
                swap A[j] and A[j + 1]
```

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

Efficiency? O(n^2)

Brute Force

1. Sorting problem
    - Selection sort
    - Bubble sort
2. String matching
3. Optimization problem
    - Traveling Salesman problem
    - Knapsack problem
    - Assignment Problem

String Matching Problem

Pattern: compress

Text: We introduce a general framework which is suitable to capture an essence of compressed pattern matching

The String Matching Problem

String Matching

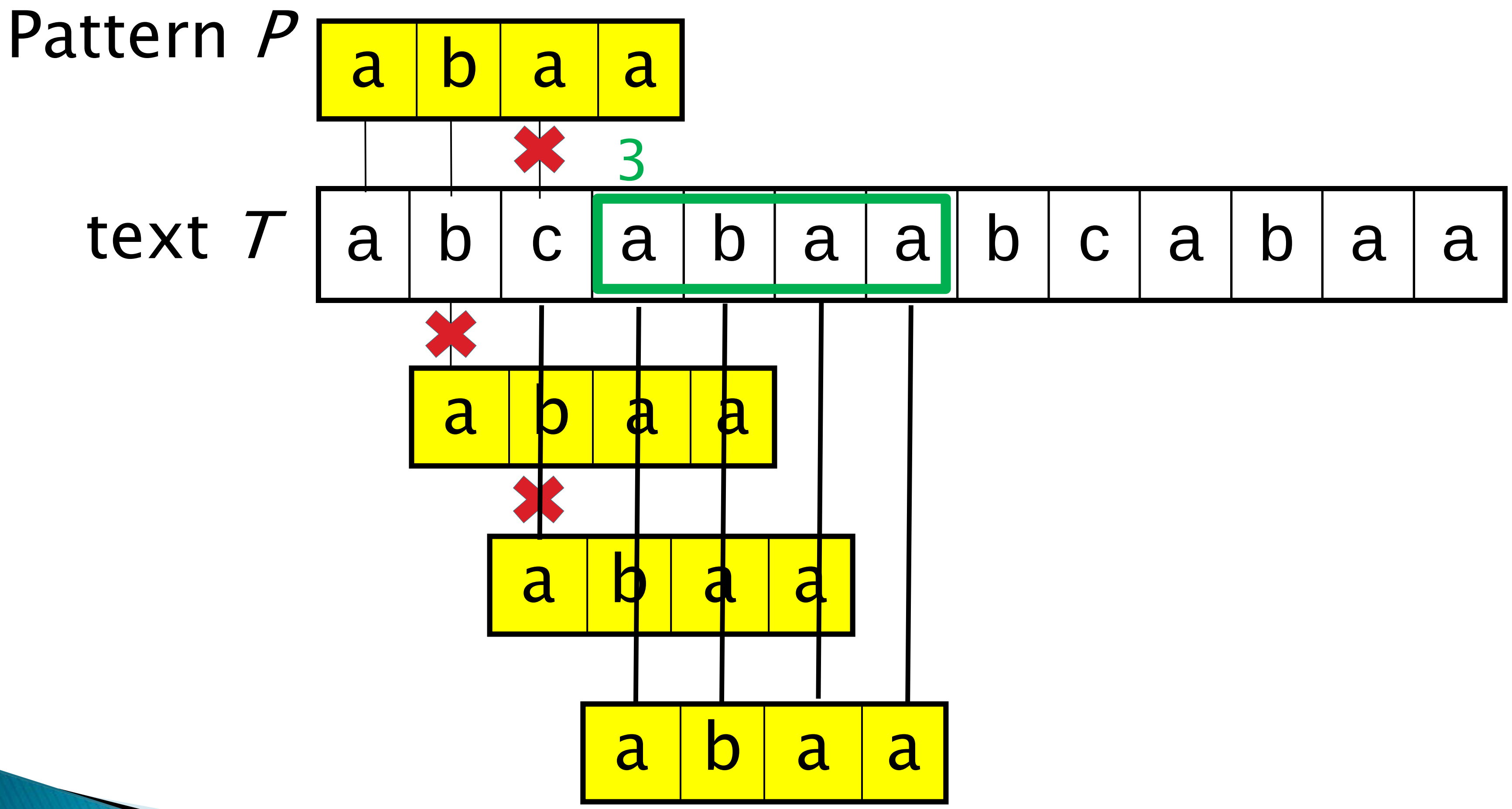
Input:

- *pattern:* A string of *m* characters to search for
- *text:* A longer string of *n* characters to search in

Problem:

Find a substring in the text that matches the pattern

The String Matching Problem



Solution (in words)

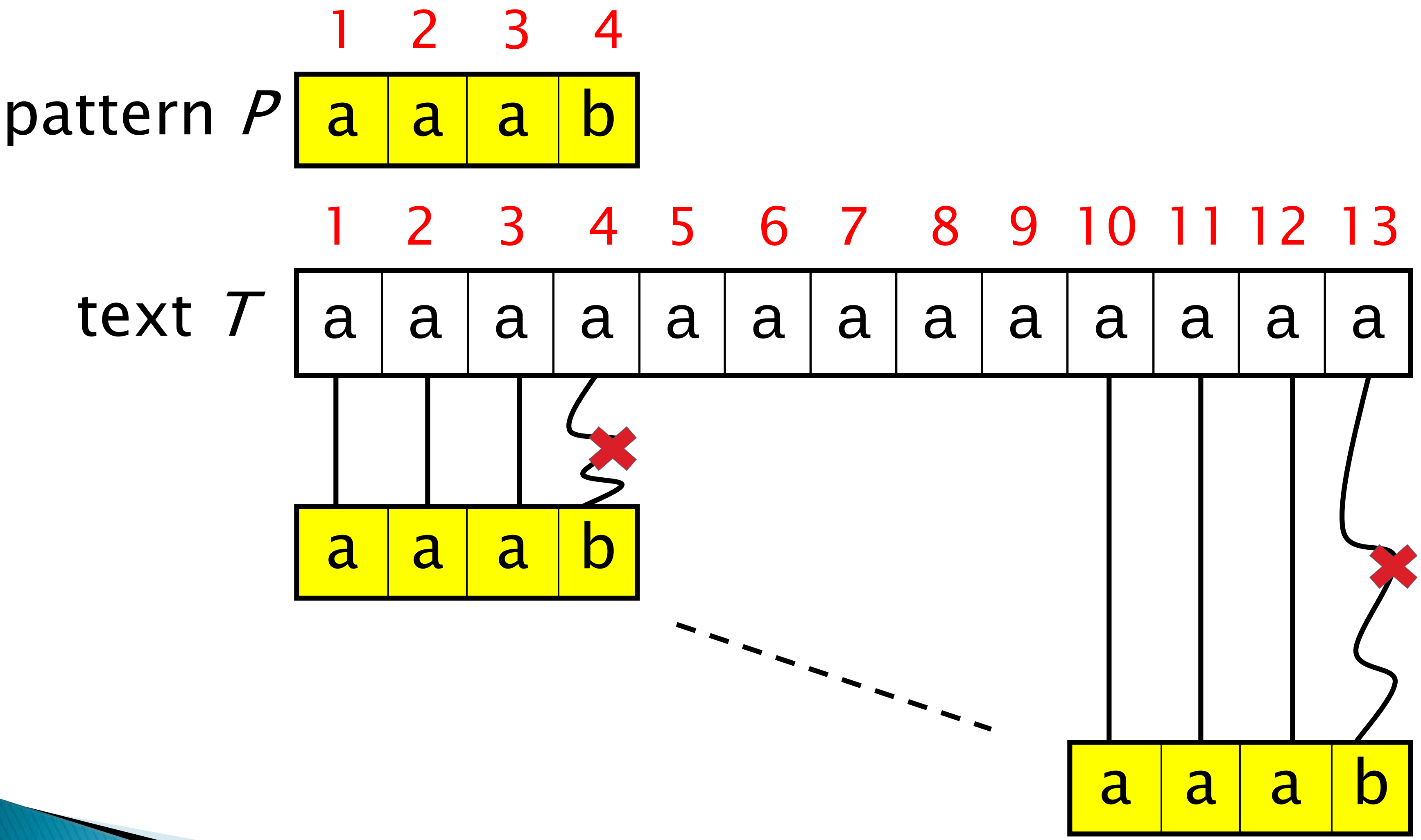
<u>Brute-force algorithm</u>

1. Align pattern at beginning of text
2. Moving from left to right, compare each character of pattern to the corresponding character in text until
   - all characters are found to match (successful search); or
   - a mismatch is detected
3. While pattern is not found and the text is not yet exhausted, realign pattern one position to the right and repeat Step 2

String Matching (pseudocode)

```
ALGORITHM  BruteForceStringMatch($T[0..n-1]$, $P[0..m-1]$)
    //Implements brute-force string matching
    //Input: An array $T[0..n-1]$ of $n$ characters representing a text and
    //          an array $P[0..m-1]$ of $m$ characters representing a pattern
    //Output: The index of the first character in the text that starts a
    //            matching substring or $-1$ if the search is unsuccessful
    for $i \leftarrow 0$ to $n - m$ do
        $j \leftarrow 0$
        while $j < m$ and $P[j] = T[i + j]$ do
            $j \leftarrow j + 1$
        if $j = m$ return $i$
    return $-1$
```

Analysis: Worst-case Example



Worst-case Analysis

- There are m comparisons for each shift in the worst case (inner loop)
- There are $n-m+1$ shifts (outer loop)
- So, the worst-case running time is:
  $O((n-m+1)m)$
- In the example on previous slide, we have $(13-4+1)4$ comparisons in total

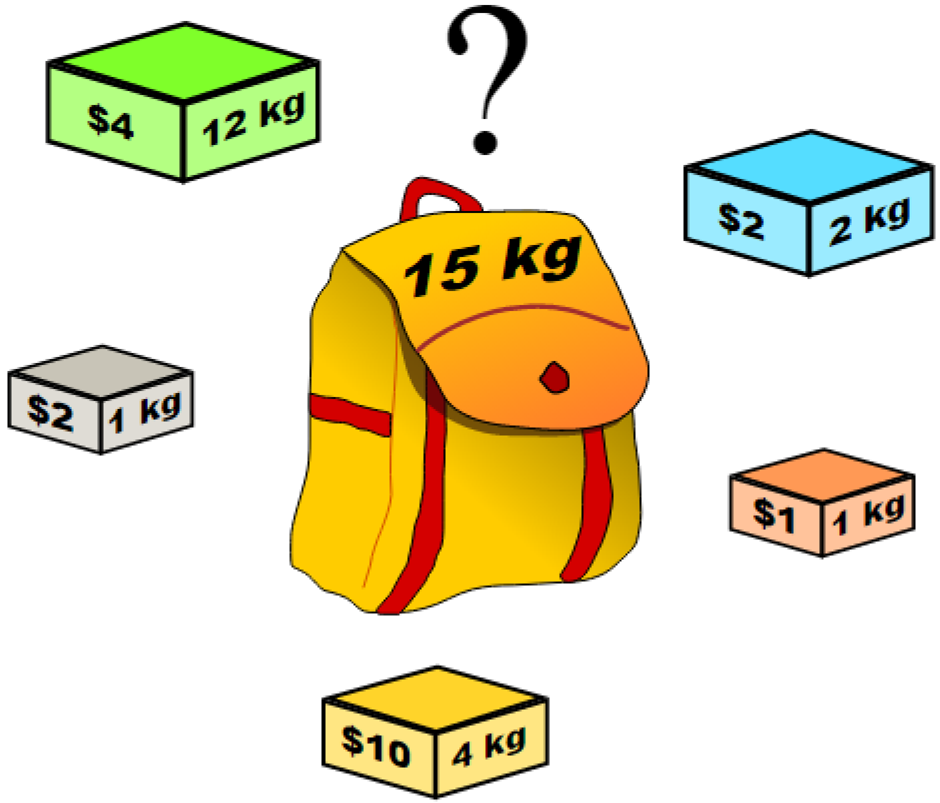
Brute Force

1. Sorting problem
   - Selection sort
   - Bubble sort
2. String matching
3. Optimization problem
   - Knapsack problem
   - Assignment Problem

Brute Force for optimization problems

- Generate a list of all potential solutions to the problem in a systematic manner
- Evaluate potential solutions one by one, disqualifying infeasible ones, and keeping track of the best one found so far
- When search ends, announce the solution(s) found

Knapsack Problem



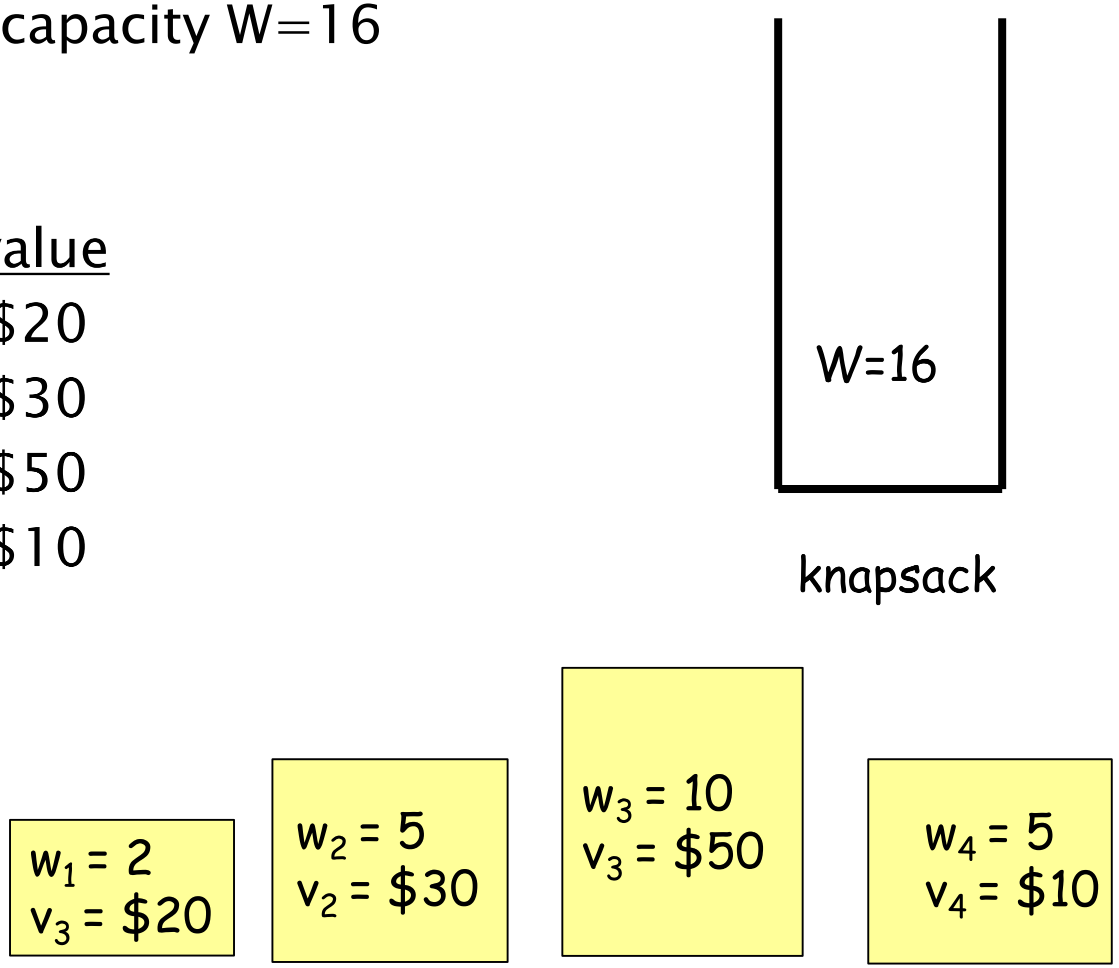
Knapsack Problem

- Input:
  - weights: $w_1 \quad w_2 \; \dots \; w_n$
  - values: $v_1 \quad v_2 \; \dots \; v_n$
  - a knapsack of capacity W
- Goal:
  - Find most valuable subset of the items that fit into the knapsack

Knapsack Problem

Example: Knapsack capacity W=16

item	weight	value
1	2	\$20
2	5	\$30
3	10	\$50
4	5	\$10



Knapsack Problem

- Generate all possible subsets of the n items
- Compute total weight of each subset
- Identify feasible subsets
- Find the subset of the largest value

Knapsack Example

Subset	Total weight	Total value
{1}	2	$20
{2}	5	$30
{3}	10	$50
{4}	5	$10
{1,2}	7	$50
{1,3}	12	$70
{1,4}	7	$30
{2,3}	15	$80
{2,4}	10	$40
{3,4}	15	$60
{1,2,3}	17	not feasible
{1,2,4}	12	$60
{1,3,4}	17	not feasible
{2,3,4}	20	not feasible
{1,2,3,4}	22	not feasible

item	weight	value
1	2	$20
2	5	$30
3	10	$50
4	5	$10

Efficiency?

Need to generate *all subsets*. For n items, there are 2^n subsets. So this is a $O(2^n)$ algorithm.

Brute Force

1. Sorting problem
   - Selection sort
   - Bubble sort
2. String matching
3. Optimization problems
   - Knapsack problem
   - Assignment Problem

Assignment Problem

- this is a classic optimization problem
- there are *n people* who need to be assigned *n jobs*, and there is a (possibly different) cost for each person to do each job
- the problem is to *find the combination of people and jobs* that has the minimum (or maximum) overall cost

Assignment Problem

	Job 1	Job 2	Job 3	Job 4
Person 1	9	2	7	8
Person 2	6	4	3	7
Person 3	5	8	1	8
Person 4	7	6	9	4

for example, this means that the cost is "8" for person 3 to do job 2

Goal:
find the combination of people and jobs that has the minimum overall cost

Assignment Problem

- possible solution 1: <1, 2, 3, 4>

	Job 1	Job 2	Job 3	Job 4
Person 1	$\textcircled{9}$	2	7	8
Person 2	6	$\textcircled{4}$	3	7
Person 3	5	8	$\textcircled{1}$	8
Person 4	7	6	9	$\textcircled{4}$

- in this case the total cost is 9+4+1+4 = 18

Assignment Problem

- Another possible solution: <2, 1, 3, 4>

	Job 1	Job 2	Job 3	Job 4
Person 1	9	[✓ 2]	7	8
Person 2	[✓ 6]	4	3	7
Person 3	5	8	[✓ 1]	8
Person 4	7	6	9	[✓ 4]

in this case the total cost is 6+2+1+4 = 13

... and another possible solution: <2, 3, 1, 4>

	Job 1	Job 2	Job 3	Job 4
Person 1	9	[✓ 2]	7	8
Person 2	6	4	[✓ 3]	7
Person 3	[✓ 5]	8	1	8
Person 4	7	6	9	[✓ 4]

in this case the total cost is 5+2+3+4 = 14

Assignment Problem

Brute Force:

- Check every combination of assignments
- Calculate the total cost
- Find the combination with minimum cost

Assignment Problem (algorithm)

- We have to *generate all possible permutations* of the job assignments, and consider the cost of each one
- Here is a brute force algorithm to solve it:

```
for each permutation P of job assignments
   totalcost ← sum of the job costs for P
   if totalcost < mincost
     mincost ← totalcost
     minperm ← P
return minperm
```

Efficiency?

Need to generate *all permutation*. For n jobs, there are n! permutation. So this is a O(n!) algorithm.

Comments on brute force

- Brute force (Exhaustive-search algorithms) run in a realistic amount of time <u>only on very small instances</u>

- In many cases, exhaustive search or its variation is the only known way to get exact solution

B.F. Strengths and Weaknesses

- Strengths
  - wide applicability
  - simplicity
  - yields reasonable algorithms for some important problems
    - matrix mult.
    - sorting
    - searching
    - string matching

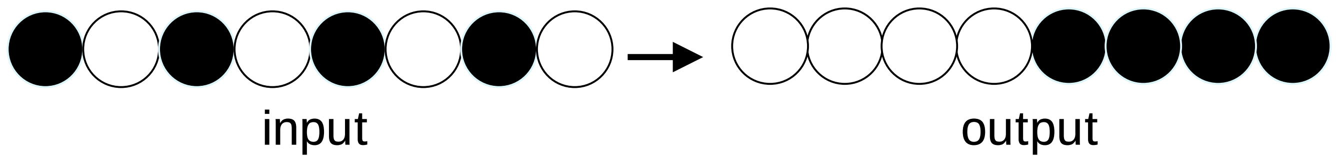
- Weaknesses
  - rarely yields efficient algorithms
  - some brute-force algorithms are unacceptably slow
  - not as constructive as some other design techniques

Example

- Consider the following problem:

You have 1 row of n disks of 2 colors, n dark and n light. They alternate dark, light, dark, light, dark, and so on. You want to get all the dark disks on the right hand side and all the light disks on the left.

The only moves you are allowed to make are those that interchange the position of two neighboring disks.



(a) Design an algorithm that solves this problem
(b) Analyze the efficiency of your algorithm.

Try it/homework

1. Chapter 3.1, page 102, questions 4, 8, 11
2. Chapter 3.1, page 107, question 5, 8