Brute Force:

Brute Force is the straight forward approach in solving the problem, usually directly based on problem statement and definition of concept involved.

Eg: Algorithms like,

- Finding maximum element in list
- Sorting
- Searching
- String Matching etc

Disadvantages:

- rarely efficient
- Some algorithms - slow
- Not very creative

Bubble Sort:

It is a sorting algorithm that compares adjacent elements of the list and exchange them if they are out of order. By doing this, repeatedly. we end up bubbing the largest element to last position on the list. The next pass bubbles second largest element and so on. The sorted result is obtained after n-1 passes.

In the algorithm, the outerloop indicates the pass number and innerloop indicates the no of comparisons to be done for every pass.

```
Algorithm BubbleSort(A[0...n-1])
//Sorts a given array by bubblesort
//Input: An array A[0...n-1] of orderable
         elements
//Output: Array A[0...n-1] of sorted elements

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]
```

Sort Elements 2, 3, 1, 4, 5

Pass 1

1st	2	3	1	4	5
2nd	2	1	3	4	5
3rd	2	1	3	4	5
4th	2	1	3	4	5

Pass 2

1st	2	1	3	4	| 5
2nd	1	2	3	4	| 5
3rd	1	2	3	4	| 5

Pass 3

1st	1	2	3	| 4	5
2nd	1	2	3	| 4	5

Pass 4

1st	1	2	| 3	4	5

Sort : 89, 45, 68, 90, 29, 34, 17

Pass 1:

89 $\xleftrightarrow{Y}$	45	68	90	29	34	17
45	89 $\xleftrightarrow{Y}$	68	90	29	34	17
45	68	89 $\xleftrightarrow{N}$	90	29	34	17
45	68	89	90 $\xleftrightarrow{Y}$	29	34	17
45	68	89	29	90 $\xleftrightarrow{Y}$	34	17
45	68	89	29	34	90 $\xleftrightarrow{Y}$	17
45	68	89	29	34	17	| 90

Pass 2:

45 $\xleftrightarrow{N}$	68	89	29	34	17	| 90
45	68 $\xleftrightarrow{N}$	89	29	34	17	| 90
45	68	89 $\xleftrightarrow{Y}$	29	34	17	| 90
45	68	29	89 $\xleftrightarrow{Y}$	34	17	| 90
45	68	29	34	89 $\xleftrightarrow{Y}$	17	| 90
45	68	29	34	17	| 89	90

Pass 3:

45 $\xleftrightarrow{N}$	68	29	34	17	| 89	90
45	68 $\xleftrightarrow{Y}$	29	34	17	| 89	90

45 29 68 ⟷ 34 17 | 89 90

45 29 34 68 ⟷ 17 | 89 90

45 29 34 17 | 68 89 90

Pass 4:

45 ⟷ 29 34 17 | 68 89 90

29 45 ⟷ 34 17 | 68 89 90

29 34 45 ⟷ 17 | 68 89 90

29 34 17 | 45 68 89 90

Pass 5:

29 ⟷ 34 17 | 45 68 89 90

29 34 ⟷ 17 | 45 68 89 90

29 17 | 34 45 68 89 90

Pass 6:

29 ⟷ 17 | 34 45 68 89 90

17 29 34 45 68 89 90 Sorted list

Analysis

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

$$= \sum_{i=0}^{n-2} n-2-i-0+1$$

$$= \sum_{i=0}^{n-2} n-1-i$$

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

$$= n-1 \sum_{i=0}^{n-2} 1 - [0+1+2+3+ \dots + n-2]$$

$$= (n-1)[n-2-0+1] - \frac{(n-2)(n-1)}{2}$$

$$= (n-1)^2 - \frac{(n-1)(n-2)}{2}$$

$$= (n-1)\left[n-1 - \frac{(n-2)}{2}\right]$$

$$= (n-1)\left[\frac{2n-2-n+2}{2}\right]$$

$$= \frac{n(n-1)}{2}$$

$$C(n) \in \Theta(n^2)$$

Selection Sort:

Selection sort works by scanning entire list for minimum element and exchange it with the first element, putting the smallest element in its position in sorted list. After n-1 passes, the list is sorted.

```
Algorithm SelectionSort(A[0...n-1])
 //Sorts given array by selection sort
 // Input: An array A[0...n-1] of orderable elements
 // output: Array A[0..n-1] sorted in ascending order.

   for i <- 0 to n-2 do
       min <- i
       for j <- i+1 to n-1 do
           if A[j] < A[min]   min <- j
       swap A[i] and A[min]
```

Sorting Example:

89 45 68 90 29 34 17

17 | 45 68 90 29 34 89

17 29 | 68 90 45 34 89

17 29 34 | 90 45 68 89

17 29 34 45 ¦ 90 68 89

17 29 34 45 68 ¦ 90 89

17 29 34 45 68 89 90 → Sorted

Analysis:

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

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

$$= \sum_{i=0}^{n-2} n-i-1$$

$$= \sum_{i=0}^{n-2} n-1 - \sum_{i=0}^{n-2} i$$

$$= n-1 \sum_{i=0}^{n-2} 1 - [0+1+2+\dots+n-2]$$

$$= (n-1)[n-2-0+1] - \frac{(n-1)(n-2)}{2}$$

$$= (n-1)^2 - \frac{(n-1)(n-2)}{2}$$

$$= (n-1)\left[n-1 - \frac{(n-2)}{2}\right]$$

$$= (n-1)\left[\frac{2n-\cancel{2}-n+\cancel{2}}{2}\right]$$

$$= \frac{n(n-1)}{2}$$

$$C(n) \in \Theta(n^2)$$

Advantages of Selection Sort:

1) Straight forward approach.
2) Simple to understand.

Disadvantages of Selection Sort:

1) Time complexity is $\Theta(n^2)$ and hence not efficient
2) Many efficient sorting Techniques are present whose time efficiency is nlogn

Sequential Search :

This algorithm compares successive elements of a given list with a search key until either a match is encountered or the list is exhausted without finding a match.

One simple modification - if search key is appended to the end of the list, the search will be successful and hence we can eliminate a check for end of array in each iteration of algorithm.

```
Algorithm : SequentialSearch2 (A[0...n], K)
// Implements sequential search with search key
// as sentinel
// I/P : An array A of n elements and search key K
// O/P : Index of element in A[0...n-1] if found, else
// returns -1

   A[n] ← K
    i ← 0
   while A[i] ≠ K do
      i ← i + 1
   if  i < n  return i
   else return -1.
```

Brute Force String Matching:

Given a string called text with 'n' characters and another string called pattern with 'm' characters, where m <= n, it is required to find the pattern string in the text string. If the search is successful, return the position of 1st occurence of 'pattern' string in 'text' string, otherwise return -1.

Design:

Align the pattern string against the text string's first m characters and compare the characters of pattern string with characters of text string from left to right. If there is match in all characters of pattern string in text string, return the position of occurrence and algorithm terminates.

If there is a mismatch, the pattern string is shifted by one position to the right and the process repeats.

Eg:

i = 0 1 2 3 4 5 6 7 8
H O W _ C R U E L - text

C R U E L - pattern
j = 0 1 2 3 4

compare, i and j H & C - no match

move pattern to right

0 1 2 3 4 5 6 7 8
H O W _ C R U E L
C R U E L
j=0 1 2 3 4

C ≠ O. Pattern shifted to right.

i=0 1 2 3 4 5 6 7 8
H O W _ C R U E L
C R U E L
j=0 1 2 3 4

C ≠ W. Pattern shifted to right

0 1 2 3 4 5 6 7 8
H O W _ C R U E L
C R U E L
0 1 2 3 4

C ≠ _. Pattern shifted to right.

0 1 2 3 4 5 6 7 8
H O W _ C R U E L
C R U E L
0 1 2 3 4

All characters match. Hence return 4.

Text[4] compared with Pattern [0]

Text [4+1] = P[1]

Text [4+2] = P [2]   ∴ Text [i+j] = P[j]

⋮   i ranges from 0 to n-m

Text [4+4] = P[4]

BRUTE FORCE STRING MATCHING PROBLEM

Search for pattern NOT in text NOBODY_NOTICED_HIM

length of Text $n = 18$, length of pattern $m = 3$ $n > m$ [Proceed with matching

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
N	O	B	O	D	Y	_	N	O	T	I	C	E	D	_	H	I	M
= N	= O	≠ T															
	≠ N	O	T														
		≠ N	O	T													
			≠ N	O	T												
				≠ N	O	T											
					≠ N	O	T										
						≠ N	O	T									
							= N	= O	= T								

2 Matches successful. T ≠ B
Move pattern 1 position to right
N ≠ D. Move pattern 1 pos to right
N ≠ B. Move pattern 1 pos to right
N ≠ O. Move pattern 1 pos to right
N ≠ D. Move pattern 1 pos to right
N ≠ Y. Move pattern 1 pos to right
N ≠ _ move pattern 1 pos to right
3 character Match successful.
Hence pattern found in text string at position 7.

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 of Text
// An array P[0 ... m-1] of m characters of pattern
// Output : Index of first character in the text for
// successful match , else -1

```
for i ← 0 to n-m do
    j ← 0
    while j < m and T[i+j] = P[j] do
        j ← j+1
    if j = m return i
return -1
```

Analysis:

Best case:

Best case occurs if pattern is present at beginning of text.

Hence no. of comparisons required would be m.

$\therefore \quad C(n) \in \Omega(m)$

Worst case:

Basic operation - comparison.

Let the comparison be executed for every iteration in the loop.

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

$$= \sum_{i=0}^{n-m} m-1-0+1$$

$$= \sum_{i=0}^{n-m} m$$

$$= m \sum_{i=0}^{n-m} 1$$

$$= m(n-m-0+1)$$

$$= mn - m^2 + m$$

$$C(n) \in O(mn)$$

Average Case:

In an average, for random texts, the no. of comparisons would be $n + m$, i.e linear.

$$C(n) \in \Theta(n+m)$$

$$C(n) \in \Theta(n)$$

Exhaustive Search:

Exhaustive Search is a brute force approach to combinatorial problems. We find each and every element of problem domain and select only those satisfying the constraints.

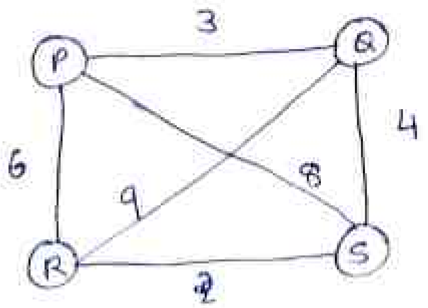
Travelling Salesman Problem (TSP):

Given 'n' cities a salesperson starts at a specified city visiting all $n-1$ cities only once, and return to the city from where he has started.

The objective of this algorithm is to find the route through the cities that minimize the cost, thereby maximizing the profit.

How to model the travelling salesperson Problem:

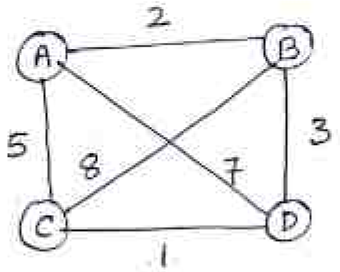
Problem can be modelled as undirected weighted graph. The vertices of the graph represents various cities. The weights associated with edges will represent the cost involved while travelling from one city to another.



This problem can be stated as the problem of finding the shortest Hamiltonian circuit. Hamiltonian circuit is a cycle that passes through all the vertices of the graph exactly once.

TOUR	LENGTH
$P \rightarrow q \rightarrow R \rightarrow S \rightarrow P$	$3 + 9 + 2 + 8 = 22$
$P \rightarrow Q \rightarrow S \rightarrow R \rightarrow P$	$3 + 4 + 2 + 6 = 15$ - Optimal
$P \rightarrow R \rightarrow Q \rightarrow S \rightarrow P$	$6 + 9 + 4 + 8 = 27$
$P \rightarrow R \rightarrow S \rightarrow Q \rightarrow P$	$6 + 2 + 4 + 3 = 15$ - Optimal
$P \rightarrow S \rightarrow R \rightarrow Q \rightarrow P$	$8 + 2 + 9 + 3 = 22$
$P \rightarrow S \rightarrow Q \rightarrow R \rightarrow P$	$8 + 4 + 9 + 6 = 27$

HW



optimal - 11

$a \rightarrow b \rightarrow d \rightarrow c \rightarrow a$

$a \rightarrow c \rightarrow d \rightarrow b \rightarrow a$

Analysis:

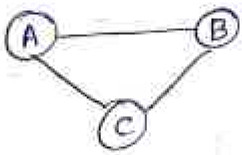
Let n denote no. of cities to visit

If $n = 2$

$a \rightarrow b \rightarrow a$

1 way $= (2-1)!$

If $n = 3$,

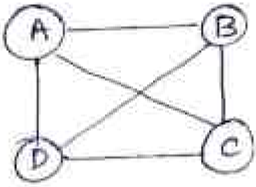


$a \rightarrow b \rightarrow c \rightarrow a$

$a \rightarrow c \rightarrow b \rightarrow a$

2 ways $= (3-1)!$

If $n = 4$



$a \rightarrow b \rightarrow c \rightarrow d \rightarrow a$

$a \rightarrow b \rightarrow d \rightarrow c \rightarrow a$

$a \rightarrow c \rightarrow b \rightarrow d \rightarrow a$

$a \rightarrow c \rightarrow d \rightarrow b \rightarrow a$

$a \rightarrow d \rightarrow c \rightarrow b \rightarrow a$

$a \rightarrow d \rightarrow b \rightarrow c \rightarrow a$

6 ways $(4-1)!$

In general, for n cities, no of routes will be $(n-1)!$

$\therefore \quad f(n) \in \Theta(n!)$

Knapsack Problem:

Given 'n' items of known weight $w_1, w_2, w_3, \ldots, w_n$ and values $v_1, v_2, v_3, \ldots, v_n$ and knapsack capacity W, we need to find the most valuable subset of items that fit into the knapsack.

Eg: Theef wanting to steal most valuable items that fit his knapsack.

Knapsack Problem:

W = 10

Item	Weights	Values
1	7	$42
2	3	$12
3	4	$40
4	5	$25

Subset	Total Weight	Total Value (Profit)
ϕ	0	$0
{1}	7	$42
{2}	3	$12
{3}	4	$40
{4}	5	$25
{1, 2}	10	$54
{1, 3}	11	Not feasible
{1, 4}	12	Not feasible
{2, 4}	8	$37
{2, 3}	7	$52
{3, 4}	9	$65 ✓
{1, 2, 3}	14	Not feasible
{1, 2, 4}	15	Not feasible
{1, 3, 4}	16	Not feasible
{2, 3, 4}	12	Not feasible
{1, 2, 3, 4}	19	Not feasible

Feasible subset with max value = {3, 4}

Profit = $65

Problems (HW):

$W = 40$ [capacity]

no. of items $n = 3$

$\{w_1, w_2, w_3\} = \{20, 25, 10\}$

$\{v_1, v_2, v_3\} = \{30, 40, 35\}$

Analysis

The total no. of subsets obtained from 3 elements is 8. In general, give 'n' objects total no. of subsets will be 2^n.

Even in the best and worst case, total no of subsets generated will be 2^n. So, time complexity is $f(n) \in \Theta(2^n)$

Assignment Problem:

In this problem, there are n people who need to be assigned to n jobs, one person per job, i.e each person is assigned to exactly one job and each job is assigned to exactly one person. The cost of assigning i^{th} person to j^{th} job is given by cost matrix $C[i, j]$, for every pair $i, j = 1, 2, 3, \ldots, n$.

The problem is to find an assignment with minimum cost.

Consider cost matrix,

$$C = \begin{array}{c} \\ P1 \\ P2 \\ P3 \\ P4 \end{array} \begin{array}{c} \begin{array}{cccc} J1 & J2 & J3 & J4 \end{array} \\ \begin{bmatrix} 9 & 2 & 7 & 8 \\ 6 & 4 & 3 & 7 \\ 5 & 8 & 1 & 8 \\ 7 & 6 & 9 & 4 \end{bmatrix} \end{array}$$

we can describe feasible solution to this problem as a n-tuple $< j_1, j_2, \ldots, j_n >$ in which each j_i indicates job assigned to i^{th} person.

$<1, 2, 3, 4>$ - $9 + 4 + 1 + 4 = 18$

$<1, 2, 4, 3>$ - $9 + 4 + 8 + 9 = 30$

$<1, 3, 2, 4>$ - $9 + 3 + 8 + 4 = 24$

$<1, 3, 4, 2>$ - $9 + 3 + 8 + 6 = 26$

$<1, 4, 2, 3>$ - $9 + 7 + 8 + 9 = 33$

$<1, 4, 3, 2>$ - $9 + 7 + 1 + 6 = 23$

$\langle 2, 1, 3, 4 \rangle$	-	$2 + 6 + 1 + 4 = 13$ ✓
$\langle 2, 1, 4, 3 \rangle$	-	$2 + 6 + 8 + 9 = 25$
$\langle 2, 3, 1, 4 \rangle$	-	$2 + 3 + 5 + 4 = 14$
$\langle 2, 3, 4, 1 \rangle$	-	$2 + 3 + 8 + 7 = 20$
$\langle 2, 4, 1, 3 \rangle$	-	$2 + 7 + 5 + 9 = 23$
$\langle 2, 4, 3, 1 \rangle$	-	$2 + 7 + 1 + 7 = 17$
$\langle 3, 1, 2, 4 \rangle$	-	$7 + 6 + 8 + 4 = 25$
$\langle 3, 1, 4, 2 \rangle$	-	$7 + 6 + 8 + 6 = 27$
$\langle 3, 2, 1, 4 \rangle$	-	$7 + 4 + 5 + 4 = 20$
$\langle 3, 2, 4, 1 \rangle$	-	$7 + 4 + 8 + 7 = 26$
$\langle 3, 4, 1, 2 \rangle$	-	$7 + 7 + 5 + 6 = 25$
$\langle 3, 4, 2, 1 \rangle$	-	$7 + 7 + 8 + 7 = 29$
$\langle 4, 1, 2, 3 \rangle$	-	$8 + 6 + 8 + 9 = 31$
$\langle 4, 1, 3, 2 \rangle$	-	$8 + 6 + 1 + 6 = 21$
$\langle 4, 2, 1, 3 \rangle$	-	$8 + 4 + 5 + 9 = 26$
$\langle 4, 2, 3, 1 \rangle$	-	$8 + 4 + 1 + 7 = 20$
$\langle 4, 3, 1, 2 \rangle$	-	$8 + 3 + 5 + 6 = 22$
$\langle 4, 3, 2, 1 \rangle$	-	$8 + 3 + 8 + 7 = 26$

Analysis:

The total no. of permutations obtained from 4 x 4 cost matrix was $24 = 4!$. Hence, the no. of permutations for 'n' persons to be assigned to 'n' jobs will be $n!$

The no. of permutations in the best and worst case will be $n!$ Hence, the time complexity is,

$$f(n) \in \Theta(n!)$$