CDS2003: Data Structures and Object-Oriented Programming

Lecture: Algorithm Analysis

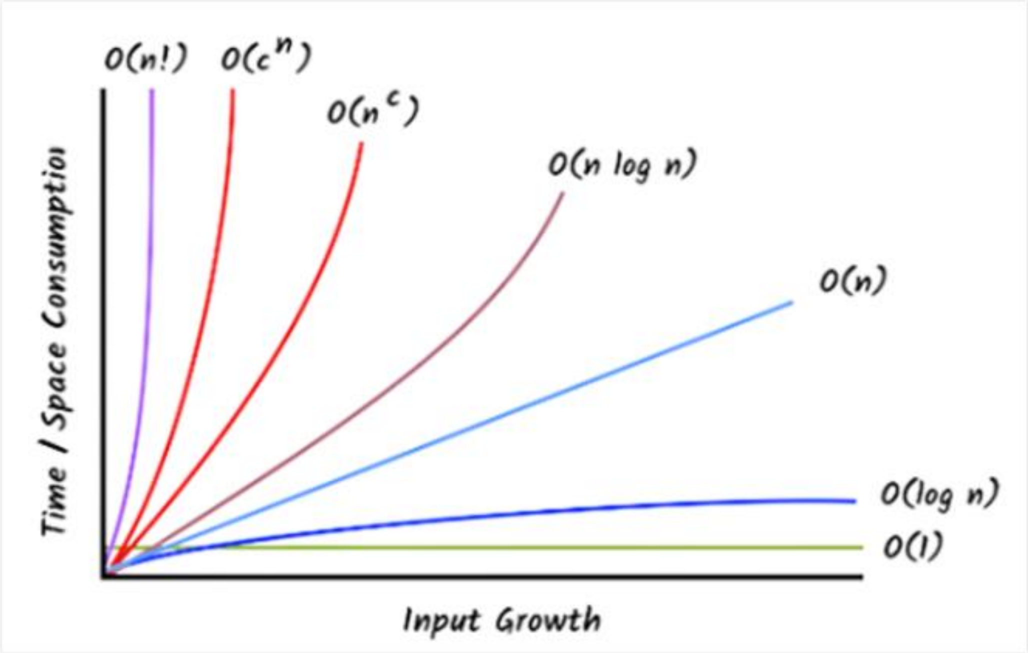
Review

- Complexity
  - Time complexity
  - Space complexity
- General rules for deriving time complexity
  - Each statement takes a constant time to run
  - Consecutive statements
  - If-Else statement
  - Loops: product of the number of iterations and the time complexity for an iteration
  - Nested loops
- Three cases in algorithm analysis
  - Worst-case
  - Average-case
  - Best-case

Time complexity

- Polynomial time (easy)
  - Constant complexity $O(1)$
  - Logarithmic complexity $O(\log n)$
  - Square root complexity $O(\sqrt{n})$
  - Linear complexity $O(n)$
  - N-LogN complexity $O(n \log n)$
  - Quadratic complexity $O(n^2)$
  - Polynomial complexity $O(n^c)$
- Super-polynomial time (hard)
  - Exponential complexity $O(c^n)$
  - Factorial complexity $O(n!)$ or $O(n^n)$

Note that $c > 1$ is a constant.



[1] https://www.scholarhat.com/tutorial/datastructures

Complexity theory

- Problem: sorting a list of n elements in increasing order

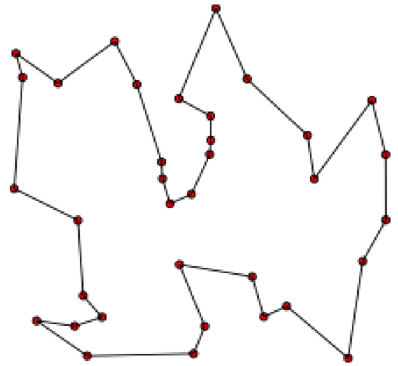
- Solution 1: selection sort algorithm
  - Repeatedly selecting the smallest element from the unsorted portion of the list and swapping it with the first element of the unsorted part until the entire list is sorted.
  - Time complexity $O(n^2)$

- Solution 2: brute-force search
  - Enumerating all permutations of the list of n elements and finding the one in increasing order
  - Time complexity $O(n!)$

- From the complexity analysis of algorithms to that of problems

Complexity theory

- A decision problem is one whose answer is either “yes” or “no”
- Many problems can be converted to a decision problem.
- Problem 1: Travelling salesman problem
  - Given a list of cities and the distances between each pair of cities
  - What is the shortest possible route that visits each city exactly once and returns to the origin city?
  - Is there route of length no more than l that visits each city exactly once and returns to the origin city?



- Problem 2: Greatest common divisor problem
  - Is the greatest common divisor of given two integers no less than k?

[1] https://en.wikipedia.org/wiki/Travelling_salesman_problem

Complexity theory – Class P

- Definition and features
  - Containing all decision problems for which there exists a deterministic Turing machine that leads to the “YES/NO” answer in polynomial time.
  - P means “polynomial time.”
  - The set of all decision problems that can be solved in polynomial time
- Greatest common divisor problem
  - Is the greatest common divisor of given two integers no less than k?
- Sorting and searching problems
- …

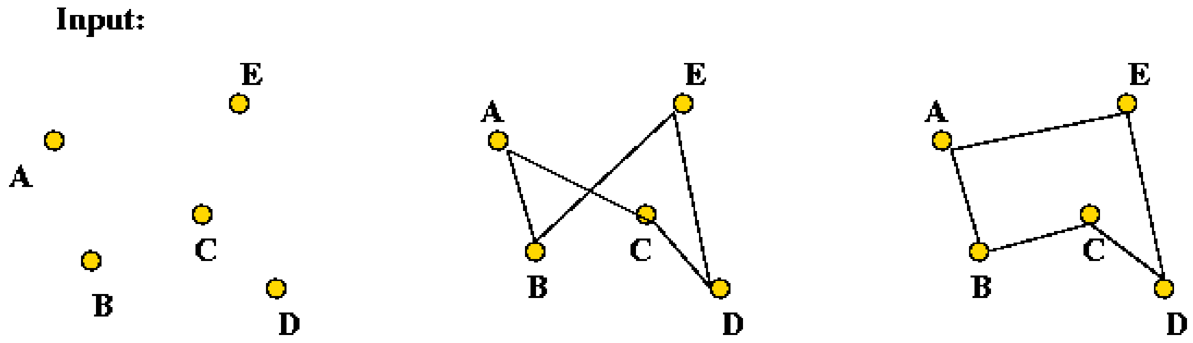
Complexity theory – Class NP

- Definition and features
  - Containing all decision problems for which there exists a deterministic Turing machine that can verify the correctness of a YES solution in polynomial time.
  - What is a YES solution to a decision problem?
    - An instance that helps give a YES answer to the decision problem
  - NP means “non-deterministic polynomial time,” but it is not necessarily known if they can be solved in polynomial time.

- NP problems hold significant importance in computer science.
  - Highlighting the difference between problems that can be solved quickly (Class P) and those that can only be verified quickly
  - Driving the development of new algorithms and heuristics
  - Representing many real-world optimization challenges
  - Providing deep theoretical insights into the nature of computation and complexity

Travelling salesman problem (TSP)

- Given: a list of cities and the distances between each pair of cities
- Is there route of length no more than l that visits each city exactly once and returns to the origin city?
- A YES solution is a route whose length is no more than l and the route visits each city exactly once and returns to the origin city.

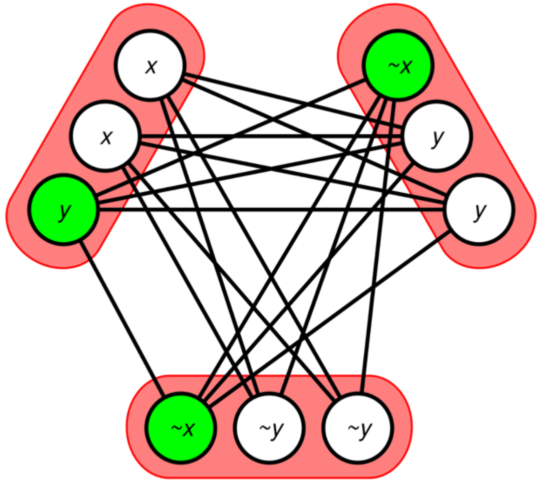


Cities	A	B	C	D	E
A	0	*	*	*	*
B	*	0	*	*	*
C	*	*	0	*	*
D	*	*	*	0	*
E	*	*	*	*	0

[1] https://www2.seas.gwu.edu/~simhaweb/champalg/tsp/tsp.html

Boolean satisfiability problem (SAT)

- Given: a Boolean formula
- Does there exist an assignment (Ture or False) to the variables such that the formula is satisfied?
- A YES solution is an assignment to the variables such that the formula is satisfied.
- Example: $(x \vee x \vee y) \wedge (\neg x \vee \neg y \vee \neg y) \wedge (\neg x \vee y \vee y\ )$



[1] https://en.wikipedia.org/wiki/Boolean_satisfiability_problem#/

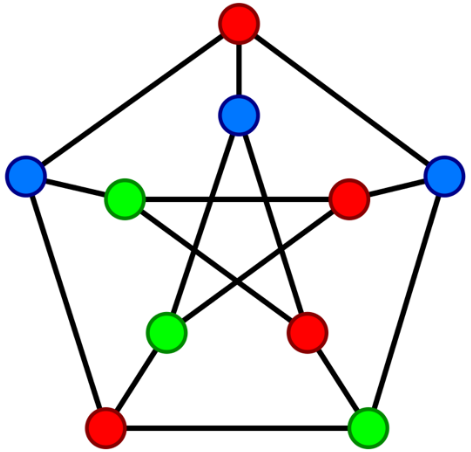
Knapsack problem

- Given: a set of items, with given values and sizes/weights/volumes
- Given: a container with a maximum capacity
- Does there exist a subset of items that can be put into the container and the total value is no less than v?
- A YES solution is a subset of items whose total size does not exceed the container capacity and whose total value is no less than v.

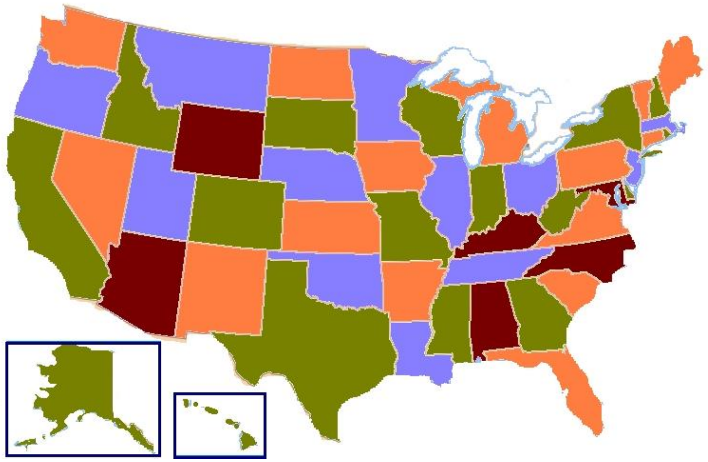
[1] https://developers.google.com/optimization/pack/knapsack

Vertex coloring problem

- Given: a graph
- Is there a way of coloring the vertices of a graph with at most k colors such that no two adjacent vertices are of the same color?
- For some special graphs, the answers can be clear.
  - Four color theorem: it would never take more than four colors to color the map such that no two neighbouring regions were the same color.



A proper vertex coloring with three colors.



A map of the United States and coloring divisions with four colors.

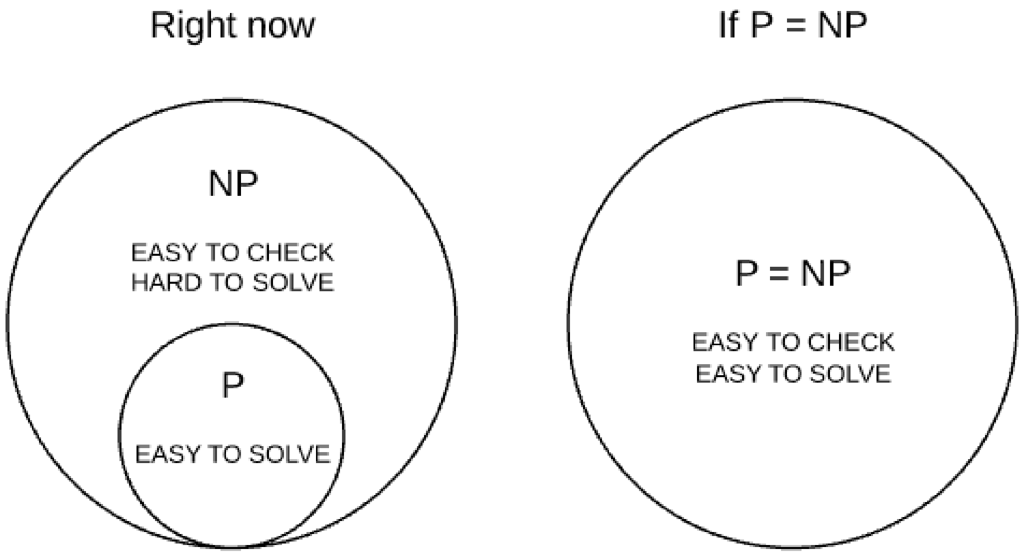
[1] https://en.wikipedia.org/wiki/Graph_coloring

Complexity theory – P versus NP

- P is a subset of NP, but is the reverse true?
  - If the solution to a problem is easy to check for correctness, must the problem be easy to solve?

- An open problem: Whether P ≠ NP or P = NP
  - One of the seven Millennium Prize Problems selected by the Clay Mathematics Institute, each of which carries a US$1,000,000 prize for the first correct solution.
  - P ≠ NP is widely believed.
  - If P = NP, what would the world be like?

- There are problems harder than NP.



[1] https://medium.com/@bilalaamir/p-vs-np-problem-in-a-nutshell-dbf08133bec5
[2] https://en.wikipedia.org/wiki/P_versus_NP_problem

Complexity theory – Class co-NP

- Definition and features
  - Containing all decision problems for which there exists a deterministic Turing machine that can verify the correctness of a NO solution in polynomial time.

- P is a subset of co-NP

- NP versus Co-NP
  - A problem is Co-NP if and only if its complement is in NP, and vice versa.

- Checking a prime number (P, NP, and co-NP)
- Checking a composite number (P, co-NP, and NP)

- Integer factorization (Both NP and co-NP)
  - For natural numbers n and k, does n have a factor smaller than k besides 1?
  - A Yes solution $(n = 27, k = 4)$ and a NO solution $(n = 35, k = 4)$
  - Unknown whether the integer factorization problem belongs to Class P

Pseudo-polynomial algorithm

- A pseudo-polynomial algorithm is an algorithm whose worst-case time complexity is polynomial in the numeric value of input (not number of inputs).

- Checking a prime number:
  - The size of input is $\lceil \log n \rceil$.
  - Exponential time complexity

```python
def isPrime(n):
    # Pre-condition: n is a nonnegative integer
    # Return True if n is prime and False otherwise
    k = 2
    while k*k <= n:
        if n % k == 0:
            return False
        k = k + 1
    return True

print(isPrime(12))
print(isPrime(17))
```

An algorithm of exponential time complexity.

Checking a prime number (PRIMES)

- P, NP, and co-NP
- Agrawal, Kayal, and Saxena showed that PRIMES is in P.

Annals of Mathematics, **160** (2004), 781–793

PRIMES is in P

By Manindra Agrawal, Neeraj Kayal, and Nitin Saxena*

Abstract

We present an unconditional deterministic polynomial-time algorithm that determines whether an input number is prime or composite.

Complexity theory

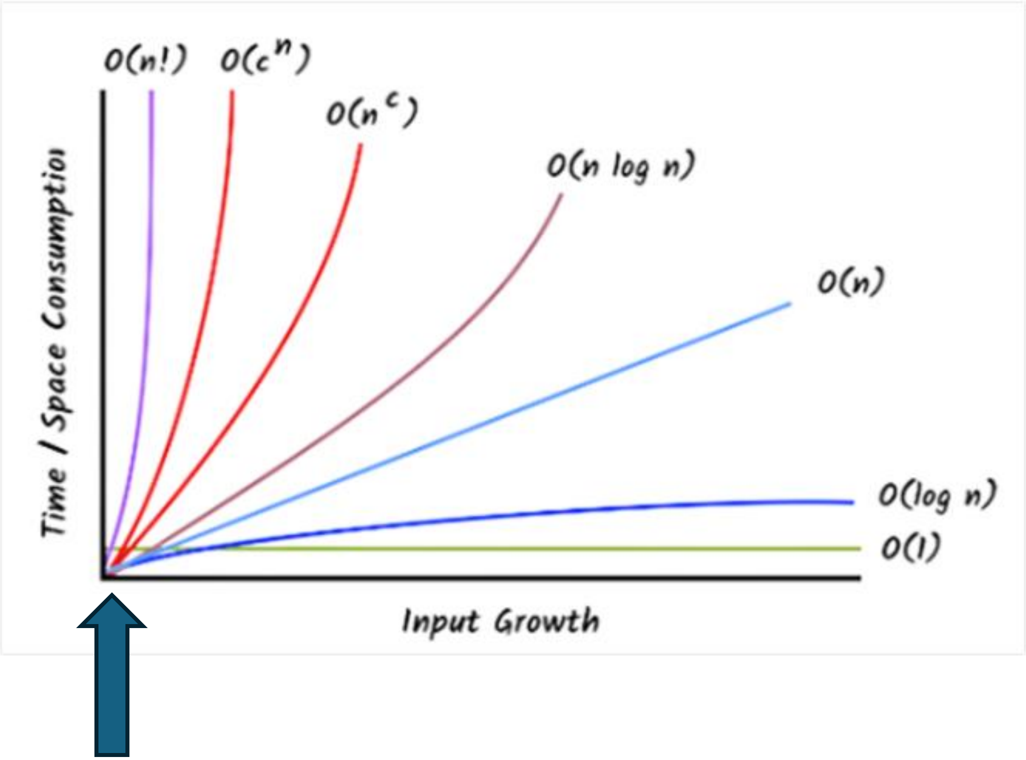
- NP-hard
  - Every problem in NP can be reduced to it in polynomial time
  - At least as hard as the hardest problem in NP
  - Example: the halting problem (Will the program halt when executed with this input, or will it run forever?)

- NP-complete
  - NP and NP-hard
  - Every problem in NP can be reduced to it in polynomial time
  - If one could solve an NP-complete problem in polynomial time, then one could solve any NP problem in polynomial time.
  - Example: the decision problem version of the TSP
  - The halting problem is not NP-complete since it does not belong to Class NP.

- So far, we cannot find polynomial algorithms to solve NP-hard problems.

How to deal with an NP-hard problem

- Using super-polynomial algorithms to solve it when the input size is small



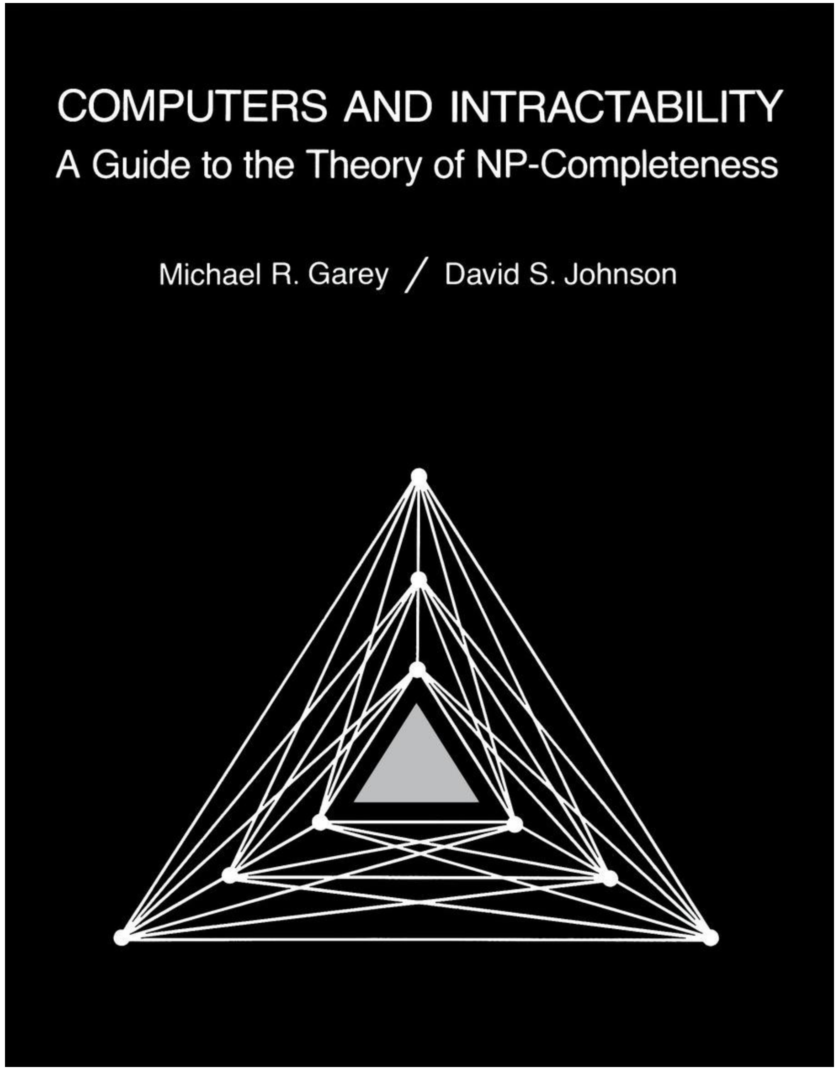
How to deal with an NP-hard problem

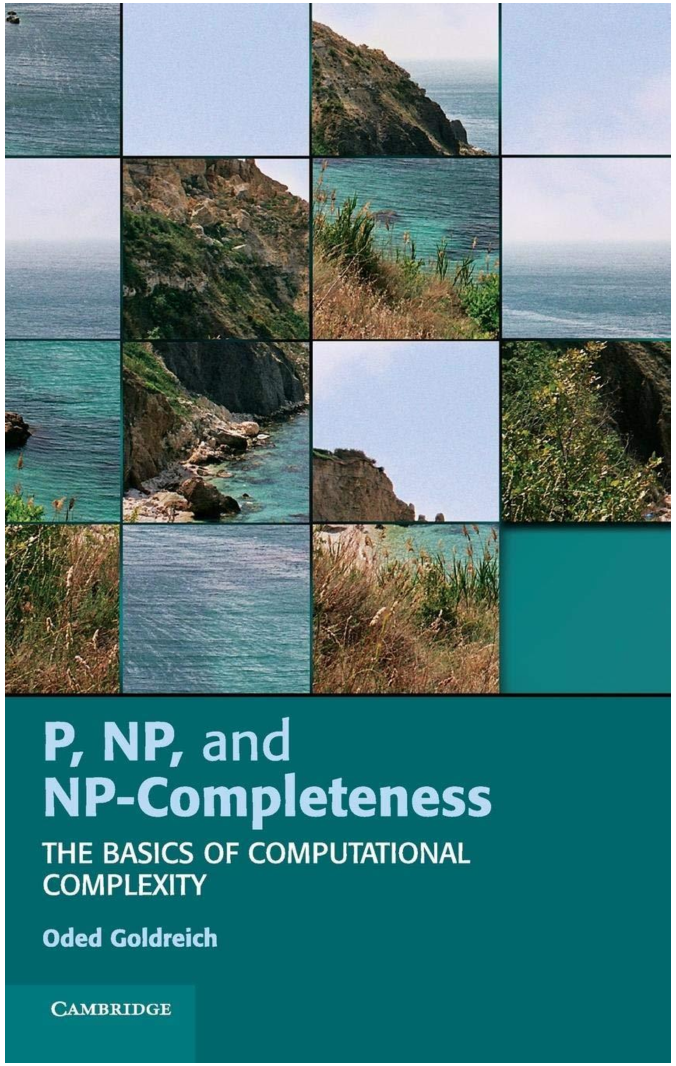
- Finding sub-optimal solution
  - Sacrificing accuracy
  - Approximation algorithms with guaranteeing the quality of the solution
  - Heuristic algorithms without guaranteeing the quality of the solution

- Quantum computer (Quantum Turing machine)
  - Many quantum algorithm for NP-hard problems are still theoretical and require further development and testing.

[1] https://quantumai.google/discover/whatisqc

Additional materials





Individual assignment

Notice

- Mandatory
  - P01: Please check whether the following statement is true or false:

    a. $5\,n + 10\,n^2 = O(n^2)$
    b. $n \log n + 4\,n = O(n)$
    c. $\log(n^2) + 4\log(\log n) = O(\log n)$
    d. $12\,n^{1/2} + 3 = O(n^2)$
    e. $3^n + 11\,n^2 + n^{20} = O(2^n)$

  - P02: Please list the functions in ascending order of their growth rates.

    $\log^2 n \qquad 2^{\sqrt{n}} \qquad 5\log\log n \qquad n^4 \qquad 7\sqrt{n} \qquad 2\log^3 n$

  - P03: Please give the time complexity of the following algorithms using the big-O notation.

```python
def my_function(n):
    if n == 1:
        return

    for i in range(1, n+1):
        # Inner loop executes only one
        # time due to break statement.
        for j in range(1, n+1):
            print("*", end="")
            break

my_function(5)  # Example: calling the function with n=5
#this code is contributed by Monu Yadav.
```

Individual assignment

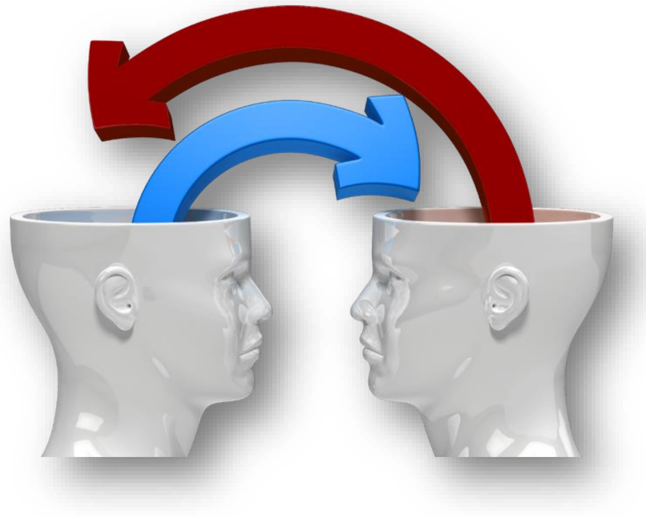
- Mandatory
  - P04: Please give the time complexity of the following algorithms using the big-O notation.

```python
def function(n):
    i = 1  # Initialize i to 1
    s = 1  # Initialize s to 1

    # Loop until the sum of consecutive integers exceeds n
    while s <= n:
        i += 1  # Increment i
        s += i  # Add i to the sum
        print("*", end="")  # Print '*' without a newline

# Example value of n
n = 10
function(n)  # Call the function
```

- Optional
  - P05: Please give the time complexity of the following algorithms using the big-O notation

```python
def myFunction(n):
    for i in range(n):
        for j in range(i, i * i):
            if j % i == 0:  # Check if j is divisible by i
                for k in range(j):
                    print("*", end="")  # Print '*' to the console without newline
        print()  # Move to the next line after printing '*' for each j

# Example usage
myFunction(5)
```

Discussion



Q & A!