Introduction to Algorithms

Subhabrata Samajder

IIIT, Delhi
Summer Semester,
25^{th} April, 2022

About Myself

Name: Subhabrata Samajder

Research Interests:

- Lattice based cryptography
- Statistical aspects of symmetric key cryptanalysis
- Broadcast Encryption
- Blockchain
- e-Voting
- Random graphs

About Myself

Name: Subhabrata Samajder

E-mail: *subhabrata@iiitd.ac.in*

Office: B-505 (R & D Block)

Office Hours: By appointment

Course Webpage: On Google classroom

- Class Code: s34dfee

Academic Integrity Policy

- Anyone caught cheating or copying will be penalised.
- Plagiarism cases will be *dealt strictly*.
- Take this opportunity to stay away from plagiarism forever.

Grading plan: Tentative Grading Components

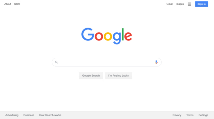
Components	Number	Weightage
MidSem Theory	1	25%
MidSem Lab	1	10%
EndSem Theory	1	25%
EndSem Lab	1	10%
Lab	11 to 13	20%
Quiz and/or Homework	≥ 4	10%

Introduction to Algorithms

Algorithms: In Our Daily Lives

Cooking

Traffic Lights



Google Search

Sorting Vinyl Records

Work Commute

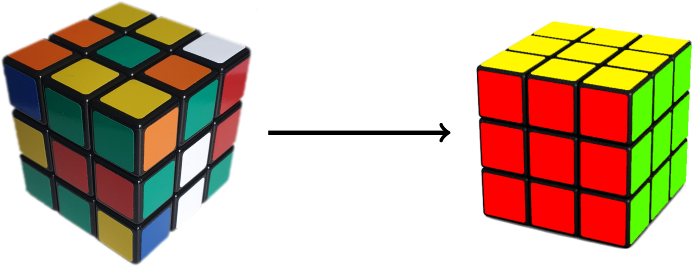
Online Shopping

Algorithms: Rubik's Cube

Solve:

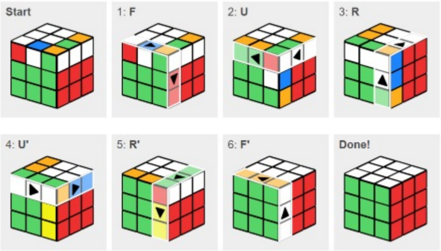
Algorithms: Rubik's Cube

Solve:



Algorithms: Rubik's Cube

Solve:



Introduction of Algorithms

"What is an algorithm?"

Introduction of Algorithms

"What is an algorithm?"

Intuitive answer: It is a finite sequence of elementary operations with the objective of performing some (computational) task.

Introduction of Algorithms

"What is an algorithm?"

Intuitive answer: It is a finite sequence of elementary operations with the objective of performing some (computational) task.

Elementary Operations

"How elementary is 'elementary?"

Elementary Operations

"How elementary is 'elementary?"

The elementary operations that we will consider will be at a higher level and include arithmetic and logical operations.

Finiteness

Algorithms: Finiteness $\Rightarrow$ an algorithm must stop.

- **Example:** Compute $(a + b) * (c + d)$

$$t_1 = a + b; \quad t_2 = c + d; \quad t_3 = t_1 * t_2.$$

Finiteness

Algorithms: Finiteness $\Rightarrow$ an algorithm must stop.

- **Example:** Compute $(a + b) * (c + d)$

$$t_1 = a + b; \quad t_2 = c + d; \quad t_3 = t_1 * t_2.$$

Computational Method: A procedure that has all of the characteristics of an algorithm except that it possibly *lacks finiteness*.

- **Example:** $\mathrm{while}(1)\{\}$

Sequence

$$t_1 = a + b; \quad t_2 = c + d; \quad t_3 = t_1 * t_2.$$

- We emphasise on the sequential nature of the procedure.

- Any permutation of this sequence *does not* give the same desired output.
  - **Example:** $t_3 = t_1 * t_2; \quad t_2 = c + d; \quad t_1 = a + b$, is not the same as the algorithm above.

Finite Sequence

$$t_1 = a + b; \quad t_2 = c + d; \quad t_3 = t_1 * t_2.$$

Note: Sometimes different orderings of the operations may give rise to the *same* result.

- **Example:**

$$\begin{array}{lllllllll} t_1 & = & a + b; & t_2 & = & c + d; & t_3 & = & t_1 * t_2; \text{ and} \\ t_1 & = & c + d; & t_2 & = & a + b; & t_3 & = & t_1 * t_2. \end{array}$$

Finite Sequence

$$t_1 = a + b; \quad t_2 = c + d; \quad t_3 = t_1 * t_2.$$

Note: Sometimes different orderings of the operations may give rise to the *same* result.

- **Example:**

$$\begin{aligned} t_1 &= a + b; \quad t_2 = c + d; \quad t_3 = t_1 * t_2; \text{ and} \\ t_1 &= c + d; \quad t_2 = a + b; \quad t_3 = t_1 * t_2. \end{aligned}$$

- **Note:** $t_1 = a + b$ and $t_2 = c + d$ can be executed independently of each other.
- **Single computing unit (a processor):** *Sequential execution*.
- **Two computing unit:** Can be executed simultaneously!

Finite Sequence

$$t_1 = a + b; \quad t_2 = c + d; \quad t_3 = t_1 * t_2.$$

Note: Sometimes different orderings of the operations may give rise to the *same* result.

- **Example:**

$$\begin{aligned} t_1 &= a + b; \quad t_2 = c + d; \quad t_3 = t_1 * t_2; \text{ and} \\ t_1 &= c + d; \quad t_2 = a + b; \quad t_3 = t_1 * t_2. \end{aligned}$$

- **Note:** $t_1 = a + b$ and $t_2 = c + d$ can be executed independently of each other.
- **Single computing unit (a processor):** *Sequential execution*.
- **Two computing unit:** Can be executed simultaneously! Give rise to the area of *parallel algorithms*.

Finite Sequence

$$t_1 = a + b; \quad t_2 = c + d; \quad t_3 = t_1 * t_2.$$

Note: Sometimes different orderings of the operations may give rise to the *same* result.

- **Example:**

$$\begin{array}{rclrclrcl} t_1 &=& a+b; & t_2 &=& c+d; & t_3 &=& t_1 * t_2; \text{ and} \\ t_1 &=& c+d; & t_2 &=& a+b; & t_3 &=& t_1 * t_2. \end{array}$$

- **Note:** $t_1 = a + b$ and $t_2 = c + d$ can be executed independently of each other.
- **Single computing unit (a processor):** *Sequential execution*.
- **Two computing unit:** Can be executed simultaneously! Give rise to the area of *parallel algorithms*.

We would only be concentrating on sequential algorithms!!

Inputs and Output

Recall: The purpose of an algorithm is to perform some task.

Inputs and Output

Recall: The purpose of an algorithm is to perform some task.

Inputs: Can take *several* inputs.

- *Example:* a, b, c, d.

Output: Algorithms produce *an* output.

- *Example:* $(a + b) * (c + d)$.

Inputs and Output

Recall: The purpose of an algorithm is to perform some task.

Inputs: Can take *several* inputs.

- *Example:* a, b, c, d.

Output: Algorithms produce *an* output.

- *Example:* $(a + b) * (c + d)$.

- The relation of the output to the input defines the *computational task* of the algorithm.

Inputs and Output

Recall: The purpose of an algorithm is to perform some task.

Inputs: Can take *several* inputs.

- *Example:* a, b, c, d.

Output: Algorithms produce *an* output.

- *Example:* $(a + b) * (c + d)$.
- The relation of the output to the input defines the *computational task* of the algorithm.
- **Simplest case:** Binary valued output (*Decision problem*).

Inputs and Output

- **Simplest case:** Binary valued output (*Decision problem*).

  Example: Searching Problem

  - **I/P:** A list L of integer values and another value v.
  - **Question:** Does $v \in L$?
  - **O/P:** '*yes*' if $v \in L$; else it returns '*no*'.

Note: *Decision problems appear rather simple but much of the sophistication of the area of algorithms can be discovered by studying such algorithms*!!

Resources of an Algorithm

'Efficient algorithms?

- **High level view:** Efficiency $\equiv$ requiring little 'resources'.

Resources of an Algorithm

'Efficient algorithms?

- **High level view:** Efficiency $\equiv$ requiring little 'resources'.

- Here, resources $\equiv$ the time of execution and the space required by the algorithm.

Resources of an Algorithm

- **Time:** # steps required by the algorithm to produce its output.
  - **Assumption:** Each elementary operation requires *unit time*.
  - $\therefore$ # steps $=$ time required by the algorithm.

Resources of an Algorithm

- **Time:** $\#$ steps required by the algorithm to produce its output.
  - **Assumption:** Each elementary operation requires *unit time*.
  - $\therefore$ $\#$ steps $=$ time required by the algorithm.
- **Space:** $\#$ *temporary* variables.

Resources of an Algorithm

- **Time:** # steps required by the algorithm to produce its output.
  - **Assumption:** Each elementary operation requires *unit time*.
  - $\therefore$ # steps = time required by the algorithm.

- **Space:** # *temporary* variables.

  Example: Our algorithm for finding $(a+b)*(c+d)$.
  - Temporary variables $= t_1, t_2$ and t_3.
  - $\therefore$ space required is 3.

Resources of an Algorithm

- **Time:** # steps required by the algorithm to produce its output.
  - **Assumption:** Each elementary operation requires *unit time*.
  - $\therefore$ # steps = time required by the algorithm.
- **Space:** # *temporary* variables.

  Example: Our algorithm for finding $(a + b) * (c + d)$.
  - Temporary variables = t_1, t_2 and t_3.
  - $\therefore$ space required is 3.
- **Other resources:** For example, power consumption is important for battery operated devices.

Size of input(s)

- **Intuitively:** Time taken by an algorithm will depend on the size(s) of its input(s).

Size of input(s)

- **Intuitively:** Time taken by an algorithm will depend on the size(s) of its input(s).

  Example: Consider our searching problem.

  - $\uparrow$ in size of the list $\Rightarrow$ algorithm takes more time.

Size of input(s)

- **Intuitively:** Time taken by an algorithm will depend on the size(s) of its input(s).

  Example: Consider our searching problem.

  - $\uparrow$ in size of the list $\Rightarrow$ algorithm takes more time.

- Thus, one has to factor in the size(s) of the input(s) while talking about algorithmic efficiency.

Size of input(s)

- **Intuitively:** Time taken by an algorithm will depend on the size(s) of its input(s).

  Example: Consider our searching problem.

  - $\uparrow$ in size of the list $\Rightarrow$ algorithm takes more time.

- Thus, one has to factor in the size(s) of the input(s) while talking about algorithmic efficiency.

- **Note:** Set of all possible inputs is *typically infinite*.

Size of input(s)

- **Intuitively:** Time taken by an algorithm will depend on the size(s) of its input(s).

  Example: Consider our searching problem.

  - $\uparrow$ in size of the list $\Rightarrow$ algorithm takes more time.

- Thus, one has to factor in the size(s) of the input(s) while talking about algorithmic efficiency.

- **Note:** Set of all possible inputs is *typically infinite*.

- **Size of inputs:** A *function* from the *set of all possible inputs to* $\mathbb{Z}^+$.

Size of input(s)

- **Intuitively:** Time taken by an algorithm will depend on the size(s) of its input(s).

  Example: Consider our searching problem.

  - $\uparrow$ in size of the list $\Rightarrow$ algorithm takes more time.

- Thus, one has to factor in the size(s) of the input(s) while talking about algorithmic efficiency.

- **Note:** Set of all possible inputs is *typically infinite*.

- **Size of inputs:** A *function* from the *set of all possible inputs to* $\mathbb{Z}^+$.

- Fixing a positive integer n fixes the set of all inputs of size n and this is a typically a *finite set*.

Size of input(s)

- **Note:** The set of all possible inputs depend on the algorithm and so does the size function.

Size of input(s)

- **Note:** The set of all possible inputs depend on the algorithm and so does the size function.

Example:

- *Search Problem:* $|L|$.
- *Arithmetic Problem?*

Size of input(s)

- **Note:** The set of all possible inputs depend on the algorithm and so does the size function.

Example:

- *Search Problem:* $|L|$.
- *Arithmetic Problem:*
  - Additions: 2
  - Multiplications: 1
  - Time: $2\times$Cost of Additions $+ 1\times$Cost of Multiplication

Size of input(s)

- **Note:** The set of all possible inputs depend on the algorithm and so does the size function.

Example:

- *Search Problem:* $|L|$.
- *Arithmetic Problem:* $\max\{\log_2 a, \log_2 b, \log_2 c, \log_2 d\}$.
  - Additions: 2
  - Multiplications: 1
  - Time: $2\times$Cost of Additions $+$ $1\times$Cost of Multiplication

Runtime Function of an Algorithm

$t(n)$: # steps required by the algorithm on an input of size n.

Runtime Function of an Algorithm

$t(n)$: $\#$ steps required by the algorithm on an input of size n.

Note:

- $\#$ steps can vary across two different inputs of size n.

Runtime Function of an Algorithm

$t(n)$: # steps required by the algorithm on an input of size n.

Note:

- # steps can vary across two different inputs of size n.
- $\therefore$ given n, one *cannot define a unique $t(n)$* such that the algorithm requires *exactly $t(n)$ steps* on any input of size n.

Two Ways to Tackle this Problem

1. **Worst-case time complexity:** $t(n)$ is the *maximum* of the different numbers of steps that the algorithm requires for different inputs of size n.

Two Ways to Tackle this Problem

1. **Worst-case time complexity:** $t(n)$ is the *maximum* of the different numbers of steps that the algorithm requires for different inputs of size n.
    - May not present a proper picture of the performance.

Two Ways to Tackle this Problem

1. **Worst-case time complexity:** $t(n)$ is the *maximum* of the different numbers of steps that the algorithm requires for different inputs of size n.
   - May not present a proper picture of the performance.
   - May take a rather long time only for a few inputs of size n.

Two Ways to Tackle this Problem

1. **Worst-case time complexity:** $t(n)$ is the *maximum* of the different numbers of steps that the algorithm requires for different inputs of size n.
   - May not present a proper picture of the performance.
   - May take a rather long time only for a few inputs of size n.

   Example: Quick Sort.

Two Ways to Tackle this Problem

1. **Worst-case time complexity:** $t(n)$ is the *maximum* of the different numbers of steps that the algorithm requires for different inputs of size n.
   - May not present a proper picture of the performance.
   - May take a rather long time only for a few inputs of size n.

     Example: Quick Sort.
   - Labelling such an algorithm as inefficient is inappropriate.

Two Ways to Tackle this Problem

1. **Worst-case time complexity:** $t(n)$ is the *maximum* of the different numbers of steps that the algorithm requires for different inputs of size n.
   - May not present a proper picture of the performance.
   - May take a rather long time only for a few inputs of size n.

     Example: Quick Sort.
   - Labelling such an algorithm as inefficient is inappropriate.

2. **Average-case time complexity:** Considers the average case behaviour of the algorithm.
   - For each n, the set of all inputs of size n is assumed to be finite.
   - Define a *uniform distribution* on this set.
   - Then the time function $T(n)$ becomes a *random variable*.
   - *Average-case time complexity* $= E[T(n)]$ (function of n).

Runtime Function of an Algorithm (Cont.)

- We Will mostly focus on the worst-case time complexity.

Runtime Function of an Algorithm (Cont.)

- We Will mostly focus on the worst-case time complexity.
- Analogously, one can also formulate the worst-case and average-case *space* required by an algorithm.

Arithmetic Problem

f(a, b, c, d):

$$
\begin{aligned}
t_1 &= a + b \\
t_2 &= c + d \\
t_3 &= t_1 * t_2
\end{aligned}
$$

return t_3

Arithmetic Problem

f(a, b, c, d):

$$
\begin{aligned}
t_1 &= a + b \\
t_2 &= c + d \\
t_3 &= t_1 * t_2 \\
\text{return} &\quad t_3
\end{aligned}
$$

- **Basic operation:** 2 Addition and 1 multiplication

Arithmetic Problem

f(a, b, c, d):

$$\begin{aligned} t_1 &= a + b \\ t_2 &= c + d \\ t_3 &= t_1 * t_2 \\ \text{return} &\quad t_3 \end{aligned}$$

- **Basic operation:** 2 Addition and 1 multiplication
  - Depends on the size of integers a, b, c and d.
  - The sizes of a, b, c and d can vary.
  - Assume that $n = \max\{\lceil \log_2 a \rceil, \lceil \log_2 b \rceil, \lceil \log_2 c \rceil, \lceil \log_2 d \rceil\}$.
  - Adding two n-bit integers take time proportional to n.
  - Multiplying two n-bit integers take time proportional to $n^{\log_2 3}$.

Arithmetic Problem

$$
\begin{aligned}
&\mathrm{f}(a, b, c, d):\\
&\quad t_1 \;=\; a + b\\
&\quad t_2 \;=\; c + d\\
&\quad t_3 \;=\; t_1 * t_2\\
&\quad \mathrm{return} \quad t_3
\end{aligned}
$$

- **Basic operation:** 2 Addition and 1 multiplication
    - Depends on the size of integers a, b, c and d.
    - The sizes of a, b, c and d can vary.
    - Assume that $n = \max\{\lceil \log_2 a \rceil, \lceil \log_2 b \rceil, \lceil \log_2 c \rceil, \lceil \log_2 d \rceil\}$.
    - Adding two n-bit integers take time proportional to n.
    - Multiplying two n-bit integers take time proportional to $n^{\log_2 3}$.
- **Size of input:** n.

Arithmetic Problem

f(a, b, c, d):

$$
\begin{aligned}
t_1 &= a + b \\
t_2 &= c + d \\
t_3 &= t_1 * t_2 \\
\text{return} \quad & t_3
\end{aligned}
$$

- **Basic operation:** 2 Addition and 1 multiplication
  - Depends on the size of integers a, b, c and d.
  - The sizes of a, b, c and d can vary.
  - Assume that $n = \max\{\lceil \log_2 a \rceil, \lceil \log_2 b \rceil, \lceil \log_2 c \rceil, \lceil \log_2 d \rceil\}$.
  - Adding two n-bit integers take time proportional to n.
  - Multiplying two n-bit integers take time proportional to $n^{\log_2 3}$.
- **Size of input:** n.
- **Time complexity:** proportional to $n^{\log_2 3}$.

Books Consulted

1. Chapter 2 of *A Course on Cooperative Game Theory* by Satya R. Chakravarty, Palash Sarkar and Manipushpak Mitra.

2. *Introduction to Algorithms: A Creative Approach* by Udi Manber.

Thank You for your kind attention!