# Data Structures and Algorithms

University Computer Science · Vocabulary in context

| Word | Example |
|---|---|
| **data structures and algorithms** | Good **data structures and algorithms** make programs fast. |
| **Big-O** /bɪg əʊ/ | Linear search is **Big-O** of n: the time grows with the length of the list. n |
| **Big-Θ** | **Big-Θ** gives the exact growth rate, such as $\Theta(n^2)$. $\Theta(n^2)$ |
| **array** /əˈreɪ/ | An **array** stores many values under one name. |
| **linked list** /lɪŋkt lɪst/ | In a **linked list**, each item points to the next one. |
| • **node** /nəʊd/ | Each **node** holds a value and a pointer. |
| **stack** /stæk/ | The last item put on the **stack** is the first one taken off. |
| **queue** /kjuː/ | A **queue** is first in, first out. |
| **hash function** /hæʃ ˈfʌŋkʃn/ | A **hash function** turns a key into a position in the table. |
| **collision** /kəˈlɪʒn/ | A **collision** happens when two computers send at the same time. |
| **chaining** /ˈtʃeɪnɪŋ/ | With **chaining**, each slot in the hash table holds a list of keys. |
| **open addressing** | With **open addressing**, a key that collides moves to the next free slot. |



exercises.lol exercises.com.cn

| Word | Example |
|---|---|
| **load factor** /ləʊd ˈfæktə/ | The **load factor** is the number of keys divided by the number of slots. |
| **clustering** | **Clustering** happens when many keys pile up in neighbouring slots. |
| **binary search tree** /ˈbaɪnəri sɜːtʃ triː/ | In a **binary search tree**, smaller values go left and bigger values go right. |
| **heap** /hiːp/ | malloc() takes memory from the **heap**. malloc() |
| **min-heap** | In a **min-heap**, the smallest value is always at the top. |
| **adjacency list** /əˈdʒeɪsənsi lɪst/ | An **adjacency list** stores each node with a list of its neighbours. |
| **adjacency matrix** | An **adjacency matrix** is a table of 0s and 1s showing which nodes are joined. 0 1 |
| **inversion** /ɪnˈvɜːʃn/ | In [3, 1], the pair (3, 1) is an **inversion**: the order is wrong. [3, 1] (3, 1) |
| **insertion sort** /ɪnˈsɜːʃn sɔːt/ | **Insertion sort** puts each new item in the right place. |
| **merge sort** /mɜːdʒ sɔːt/ | **Merge sort** splits the list in half, sorts each half and joins them. |
| **linear search** /ˈlɪnɪə sɜːtʃ/ | A **linear search** checks each item one by one. |
| **binary search** /ˈbaɪnəri sɜːtʃ/ | **Binary search** halves the list each time. |