空間利用率

⇒ 浪費 (hash → collision handling)

static → 固定，已知 size
dynamic 動態 continuous
) 宣告方式

Array

Data type
amount

位置 ⇒ index
→ continuous, 固定

Array → integer

Random access
↓
O(1)

sort → O(n²) O(nlogn)... because of different 演算法
Insert
delete
linkup
) time, space complexity

O(n) ⇒ sequential search → sorting → O(nlogn)

binary search

BST (binary search tree)

if BST imbalance (skewed, deg vote)
⇒ find middle integer (中位數)

Linked List

→ access O(1)

use pointer

→ link traverse
↳ traverse issue (can't get element by index)

time complexity
- search O(n)
- insert O(1)
- delete O(1)

doubly / circular

Stack / Queue

- FIFO → 先進先出 排隊
- FILO → 先進後出 程式 function call

array
linked list

Queue array (知道裡面狀態)

front tail

stack

array ☆
linked list

push O(1)
pop = O(1)
top
Head

Hash

⇒ array + linked list

- search, sequential speed up → Open Addressing ⇒ Random Access

空間利用率 α = 0.75 best condition

Tree

Sequential (Linked List) → Hierarchy (Tree)

Level
0 (height, de)
1
2
3

root
fan-out (degree)
sibling
leaf node

Tree → Binary Tree → Binary Search Tree
↓ degree limitation
↓ child and parent related

Tree

general tree → binary tree → binary search tree

固定 fan-out

Method: Left child - Right sibling

complete binary tree

Heap
↳ priority Queue

tree traverse
- preorder
- inorder
- postorder

Graph G(V, E)

adjacent Matrix
adjacent List

DBE
BBG

⟷ 雙向紀錄