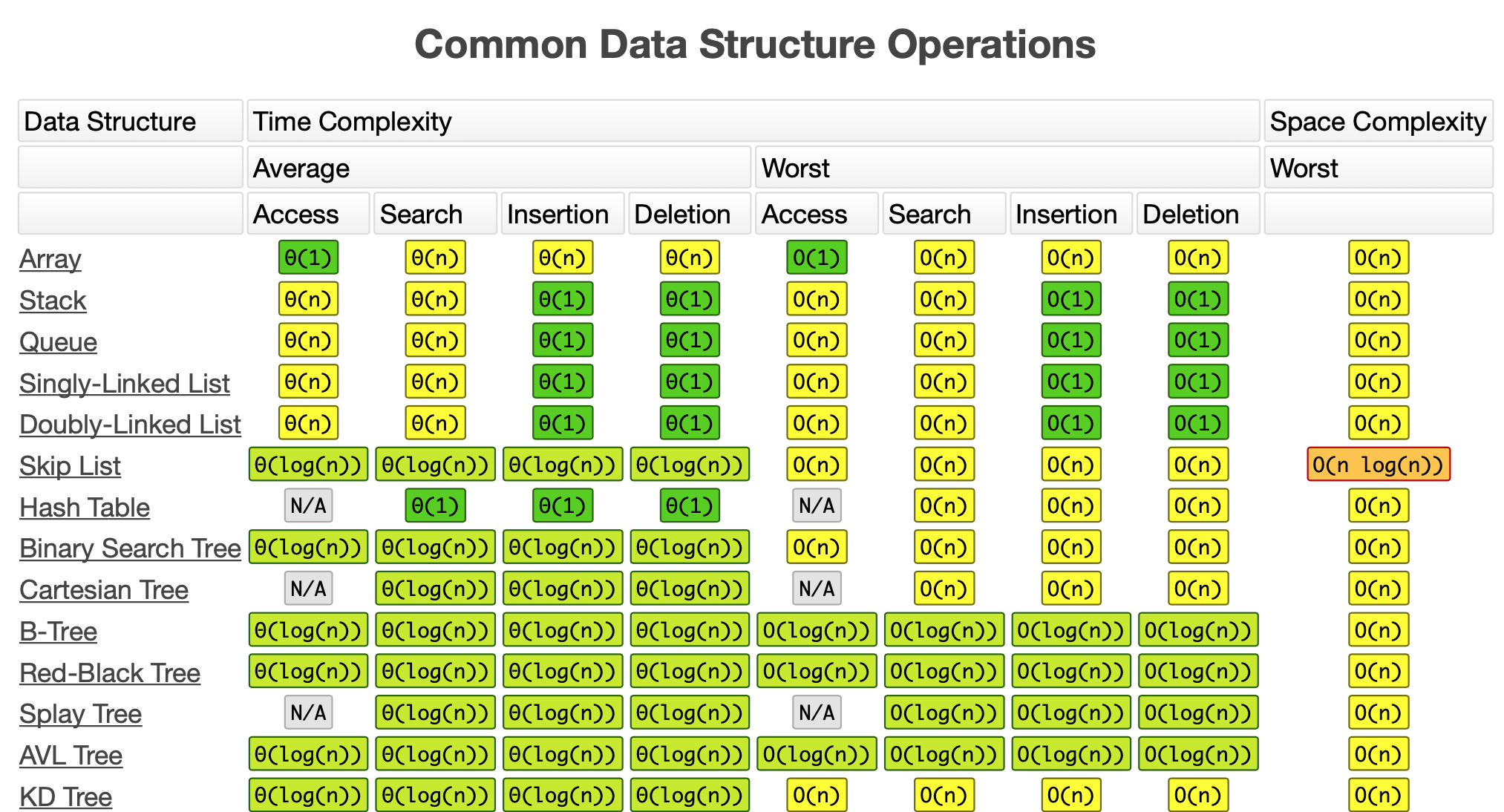
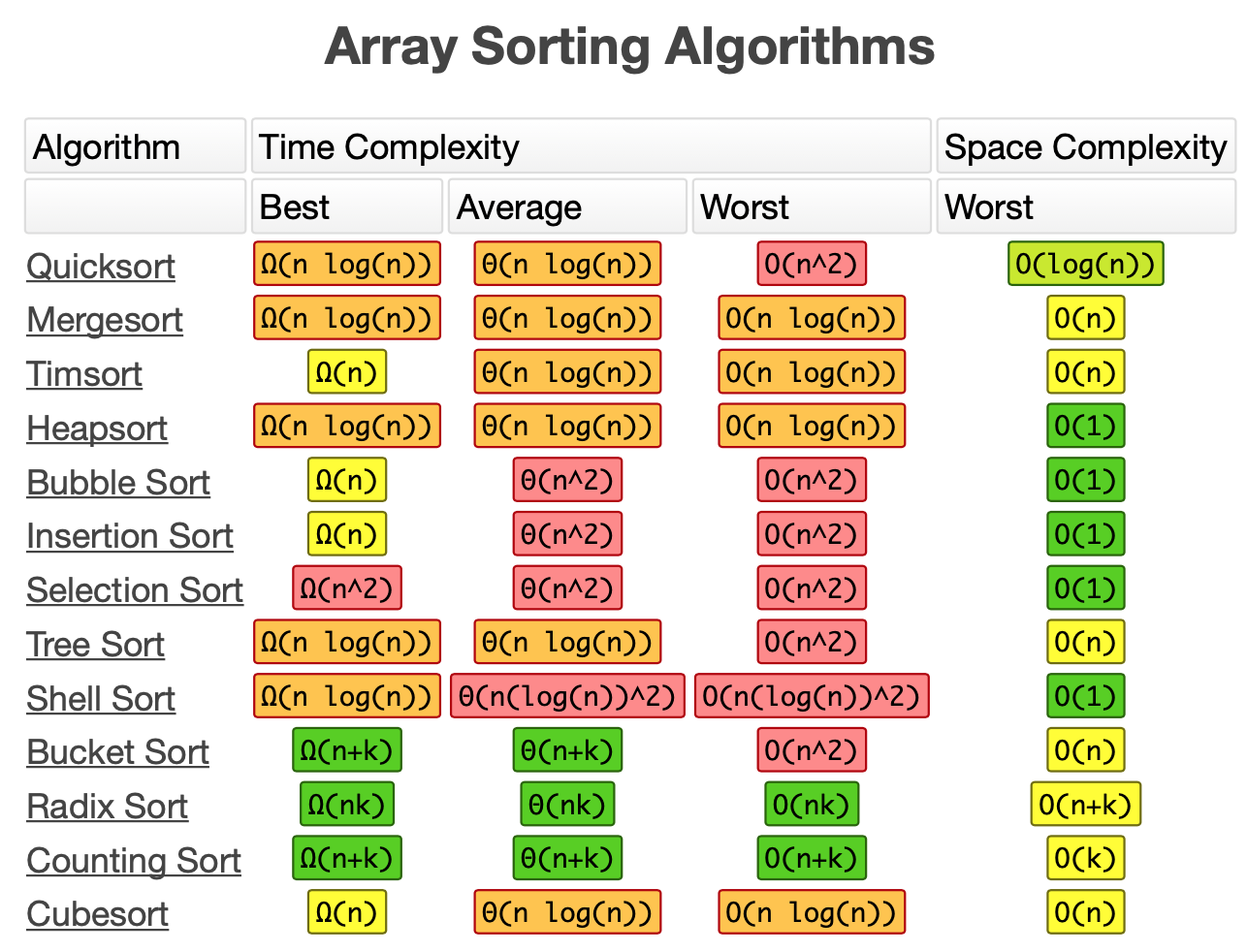
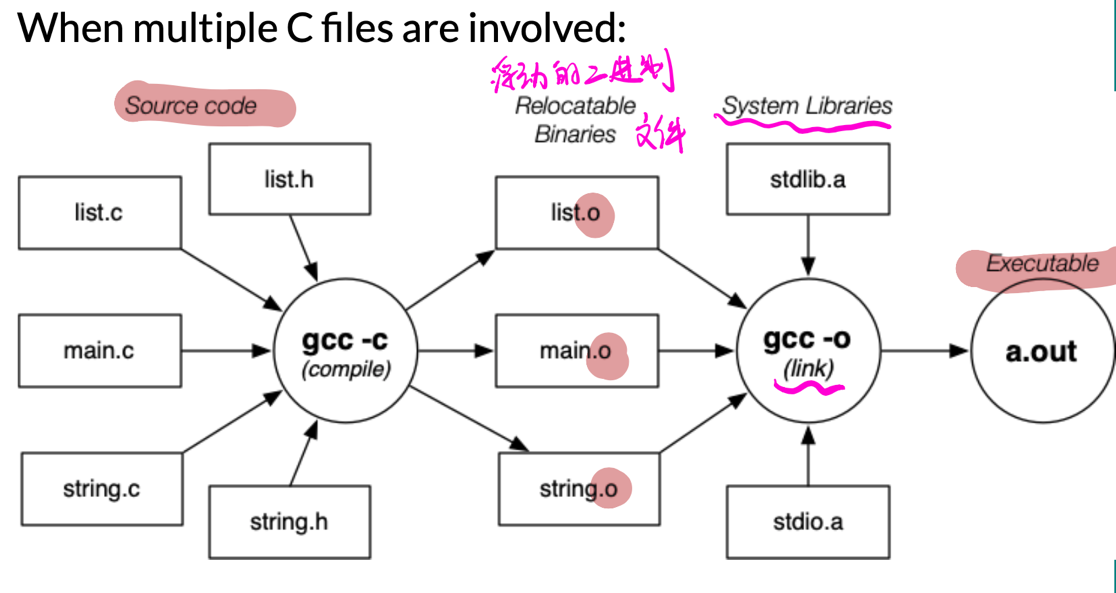
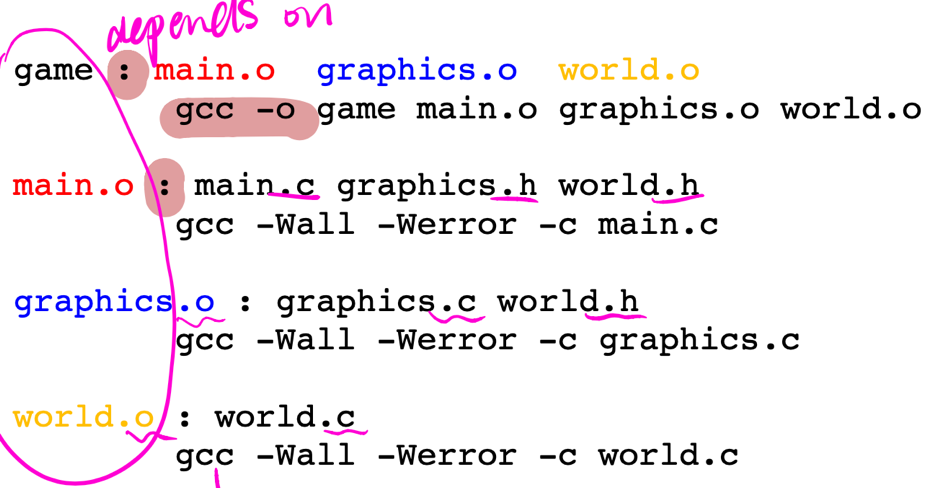
1. Running time: <https://www.bigocheatsheet.com>

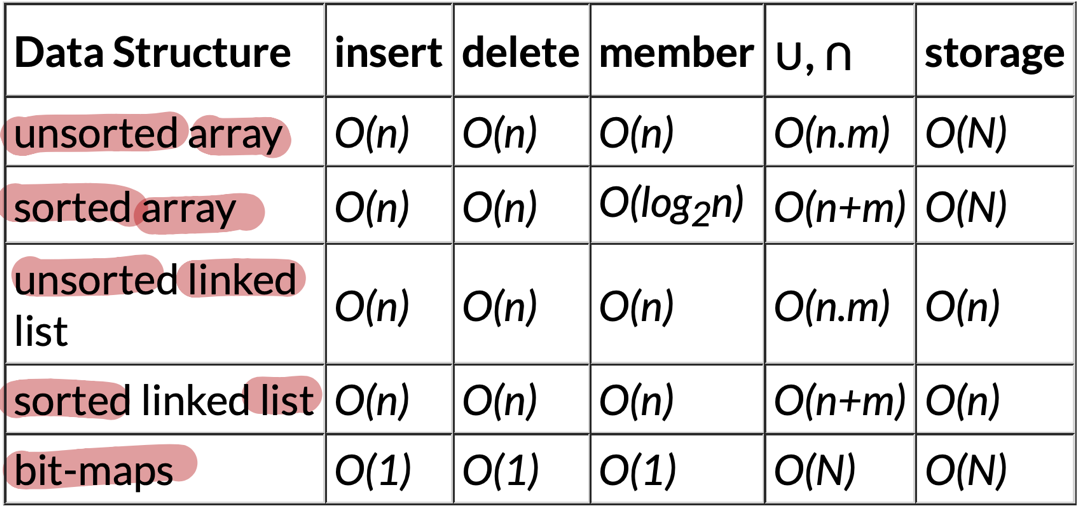


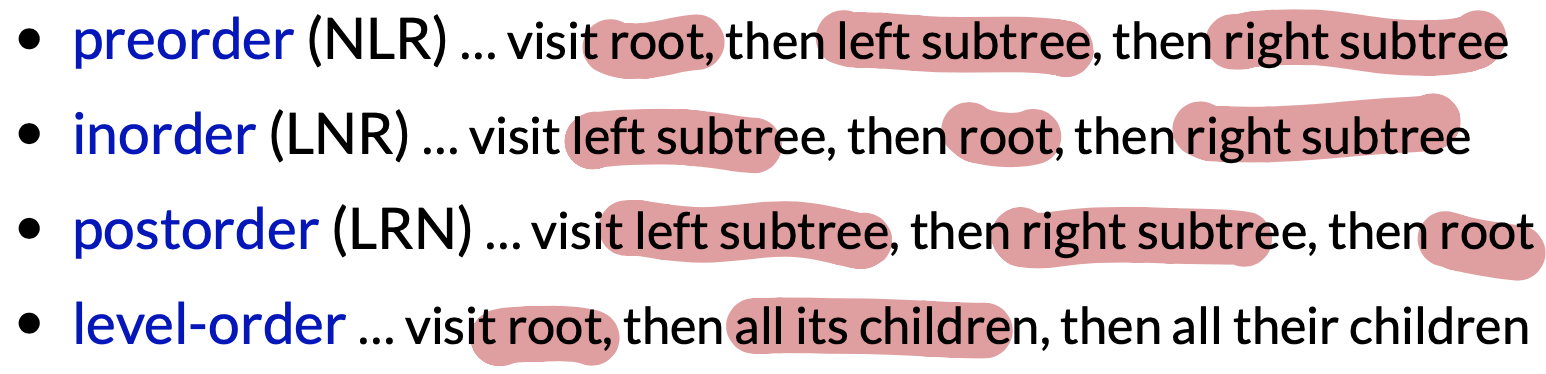
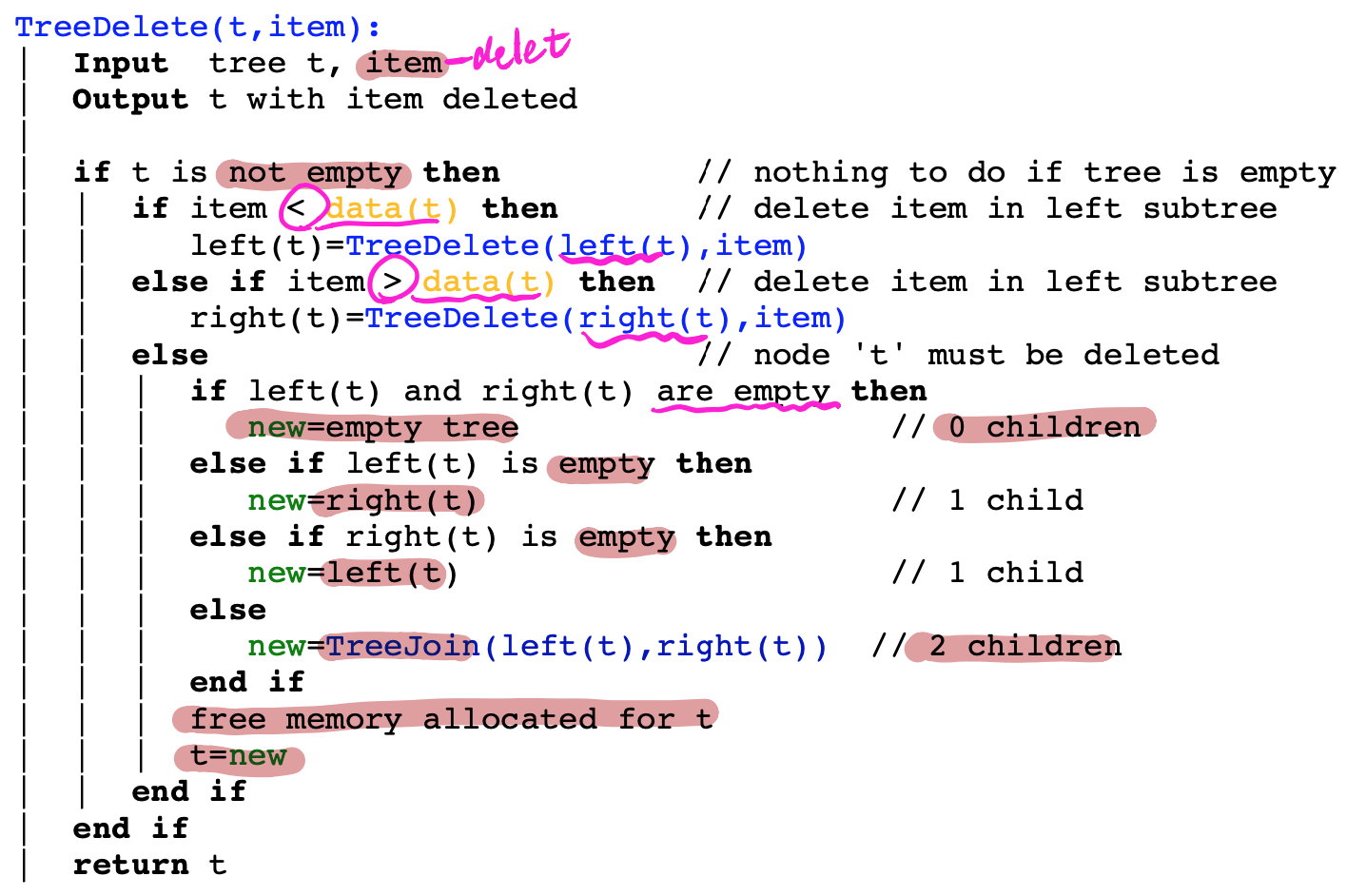
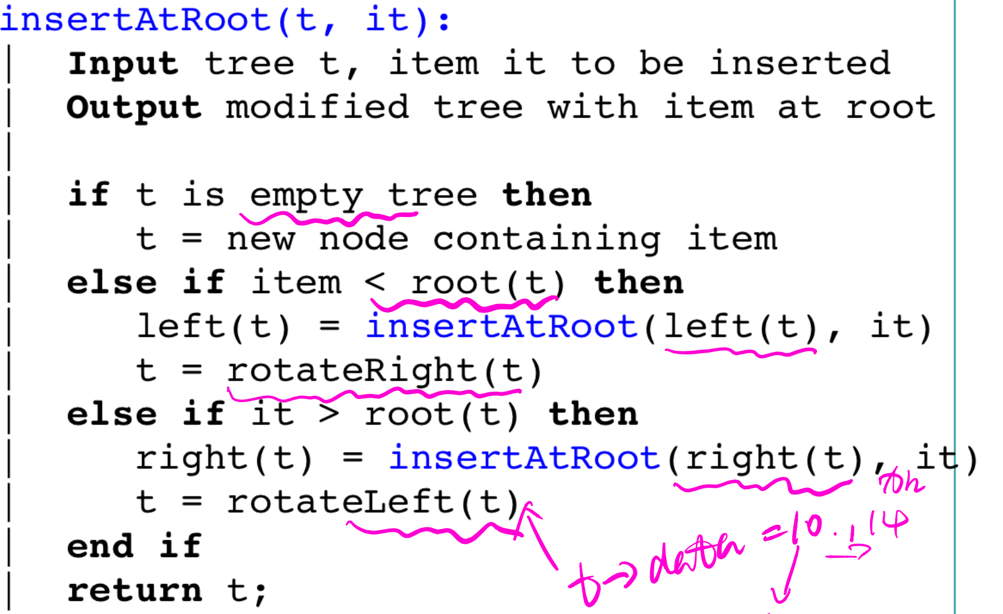
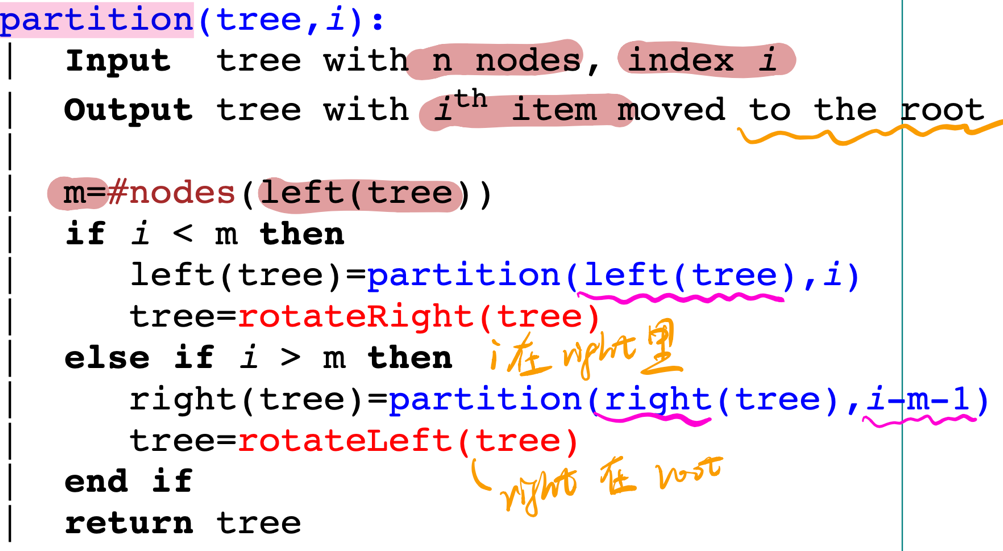
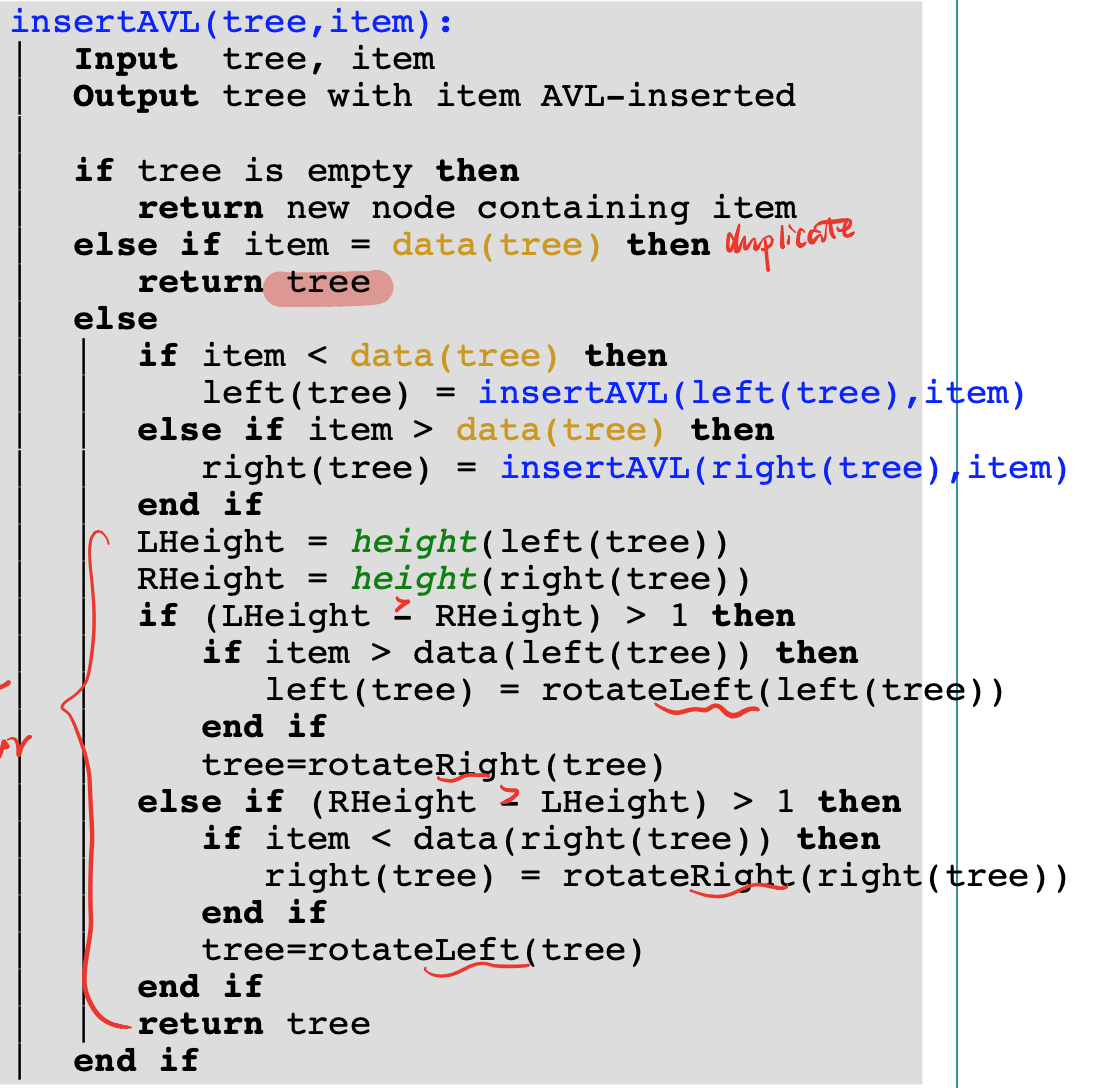
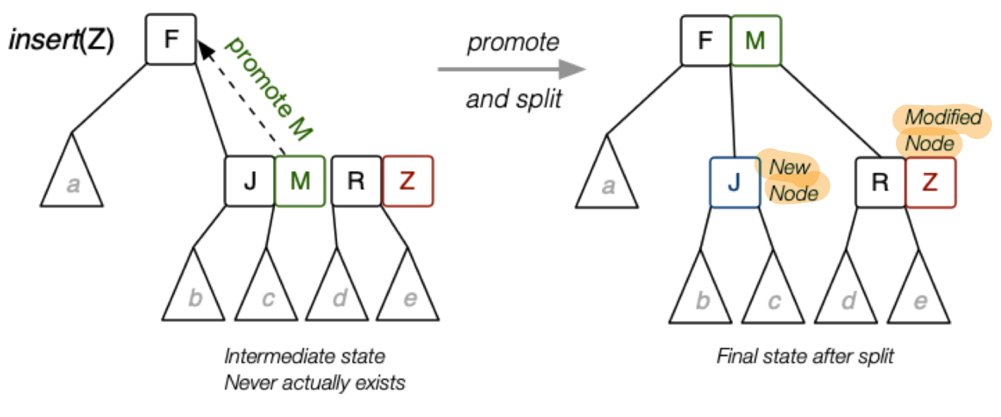
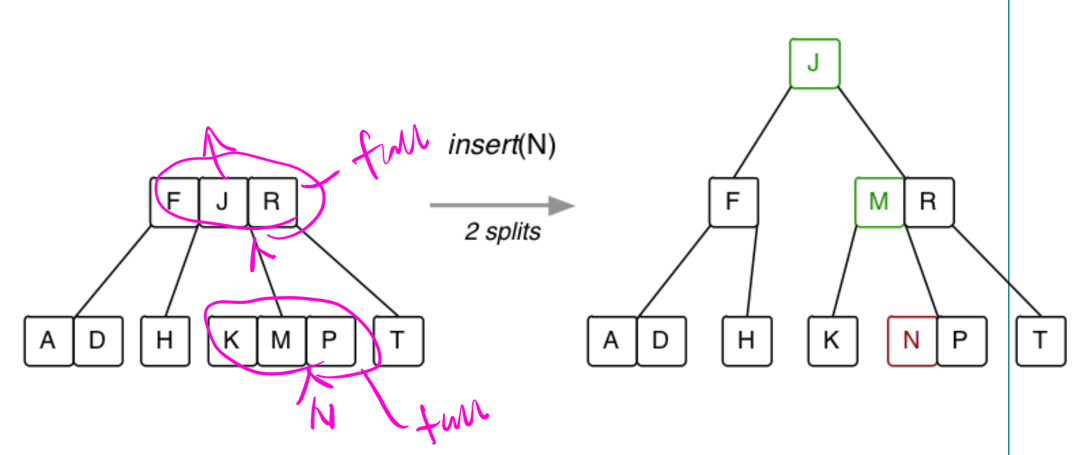




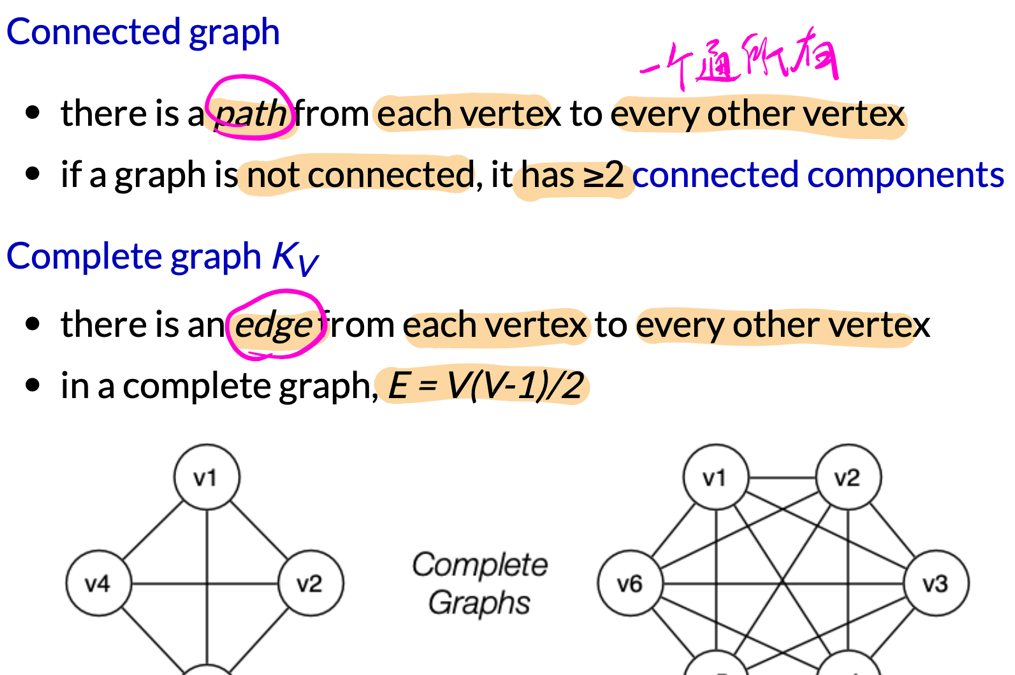


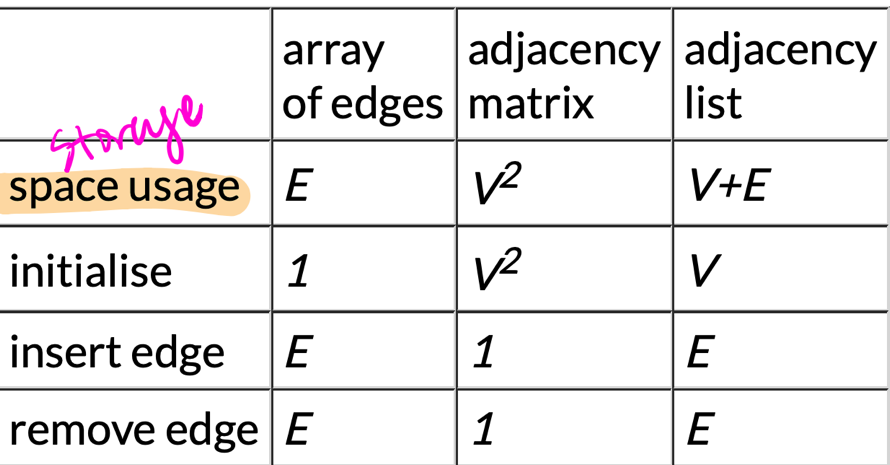
1. -E runs a \*.c file through the pre-processor and shows the result on stdout
2. -c compiles a \*.c file to a relocatable \*.o file
3. -o specifies the name of the result file for some gcc step
4. -g incorporates information into a \*.o for the debugger
5. -O3 compiles using the optimizer to make the machine code more efficient

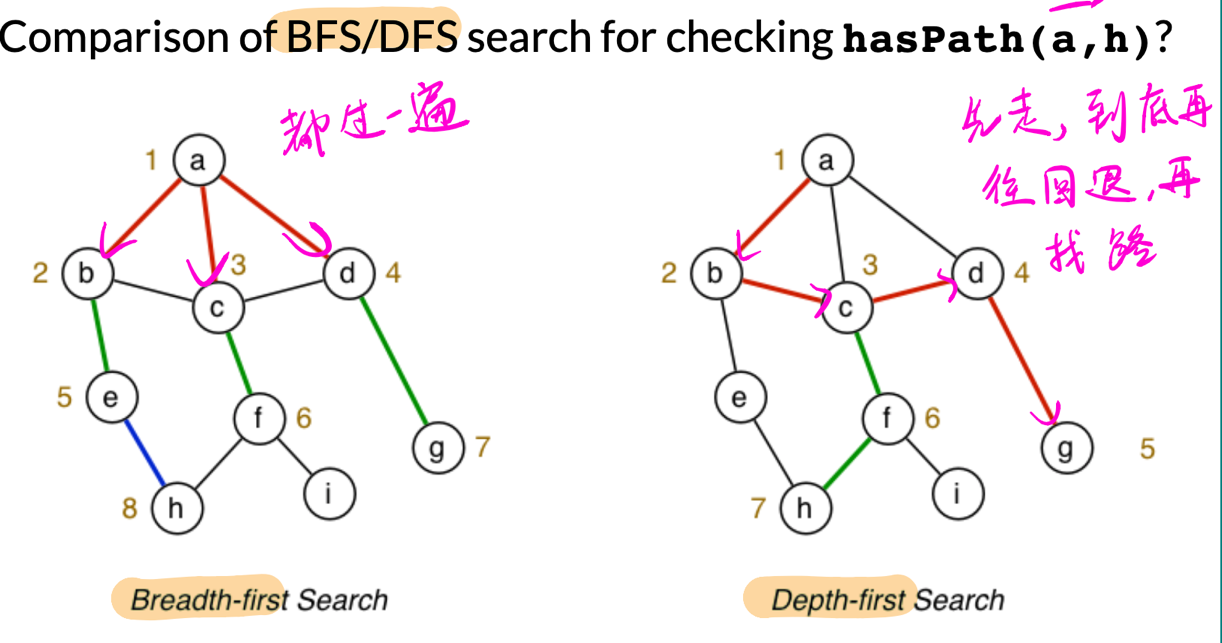


* **Binary Search Trees:**
  + **Ordered: max(left subtree) < root < min(right subtree)**
  + **Perfectly-balanced tree: #nodes(left subtree) = #nodes(right subtree)**
  + **Height-balanced tree: height(left subtree) = height(right subtree)**
  + 
  + **Delete: **
  + **Insert at root: **
  + **Partition: **
* **Splay Trees:** 
  + **insertion-at-root method**
  + **worst-case search cost O(n)**
* **AVL Trees:** 
  + **abs(height(left)-height(right)) <= 1**
  + **insert: **
* **2-3-4 Trees:**
  + ****
  + ****

**Graph**

****

****

****

****

**Hamiltonian path problem:**

find a simple path connecting two vertices v,w in graph G such that the path includes each vertex exactly once

If v = w, then we have a **Hamiltonian circuit**

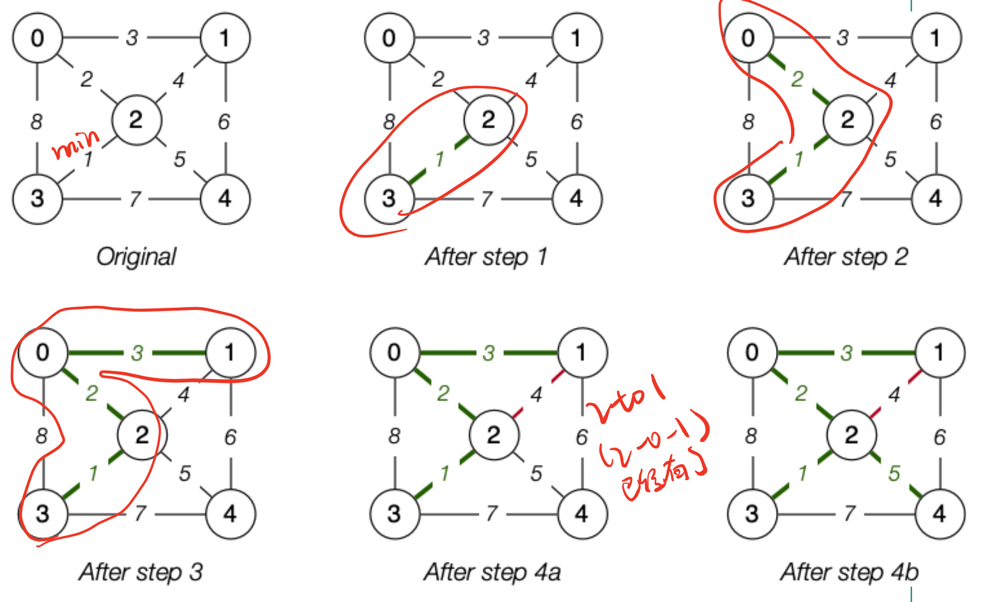
**Euler path problem:**

find a path connecting two vertices v,w in graph G such that the path includes each edge exactly once

(note: the path does not have to be simple ⇒ can visit vertices more than once)

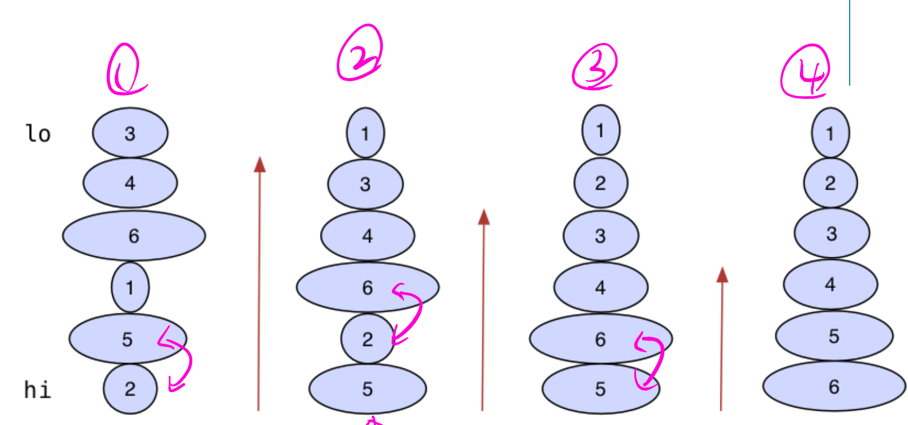
If v = w, the we have an **Euler circuit**

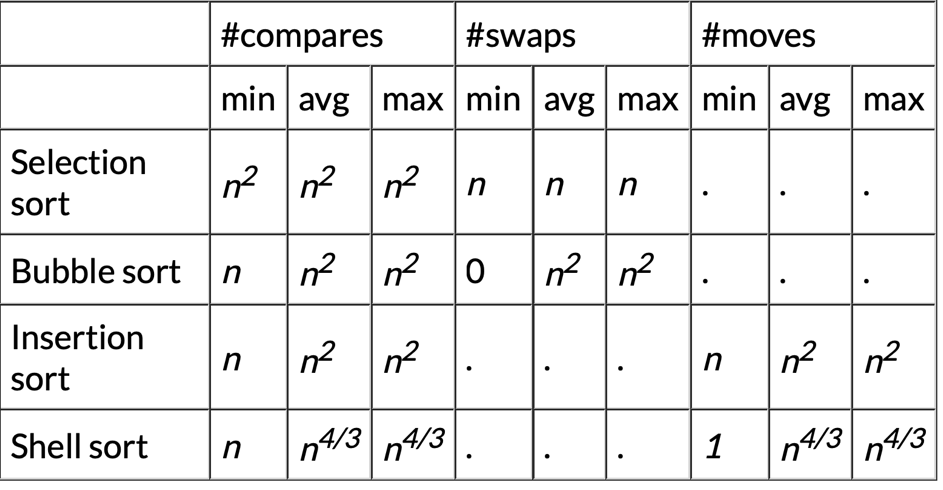
**Minimum Spanning Trees**

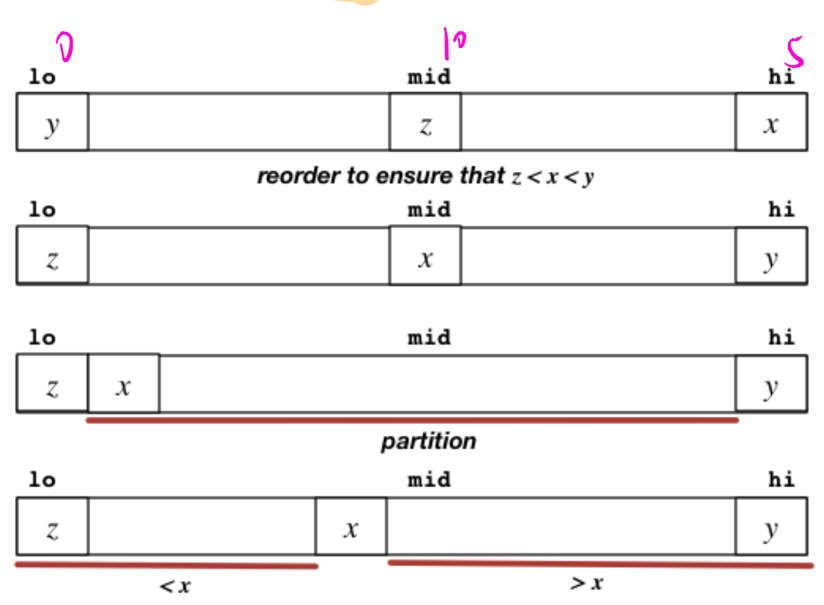
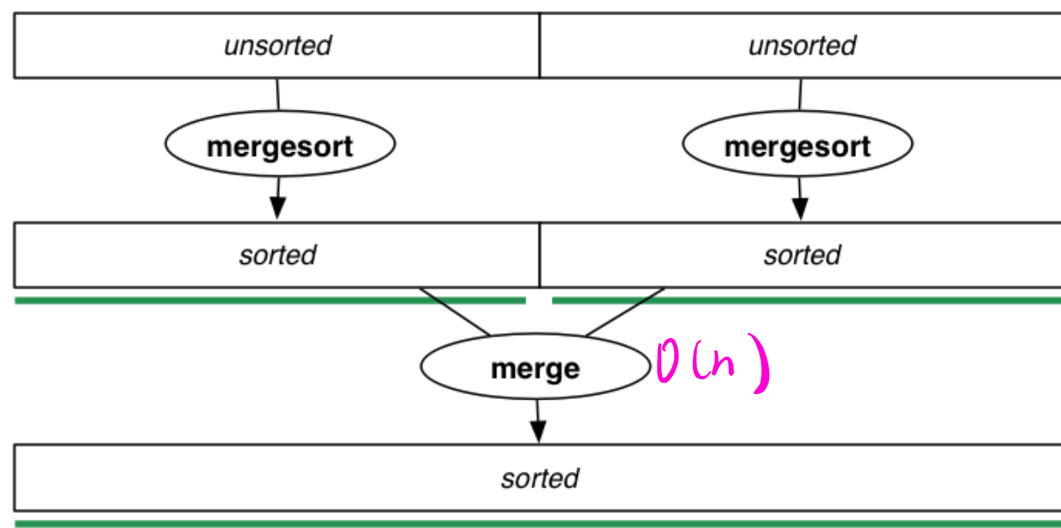
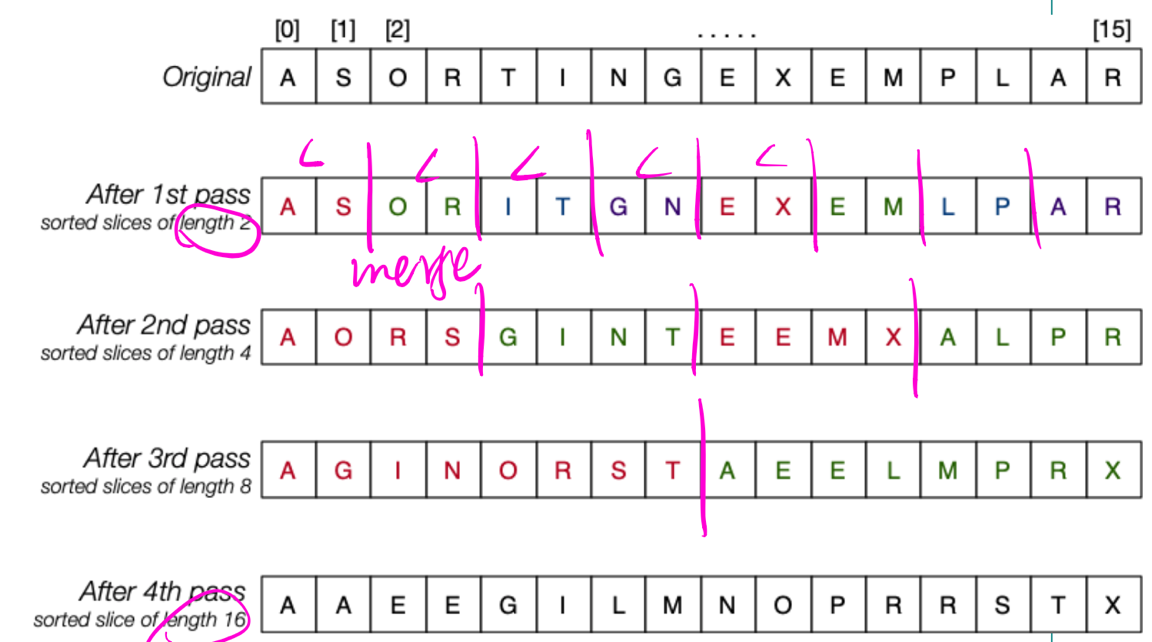
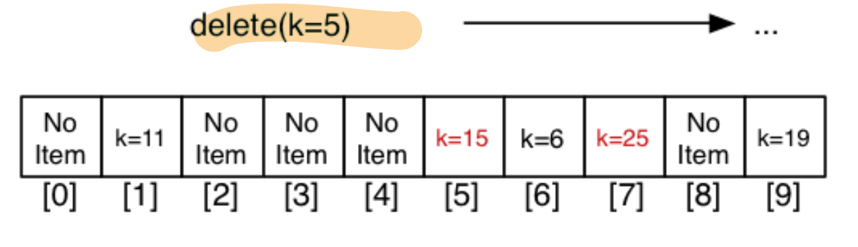
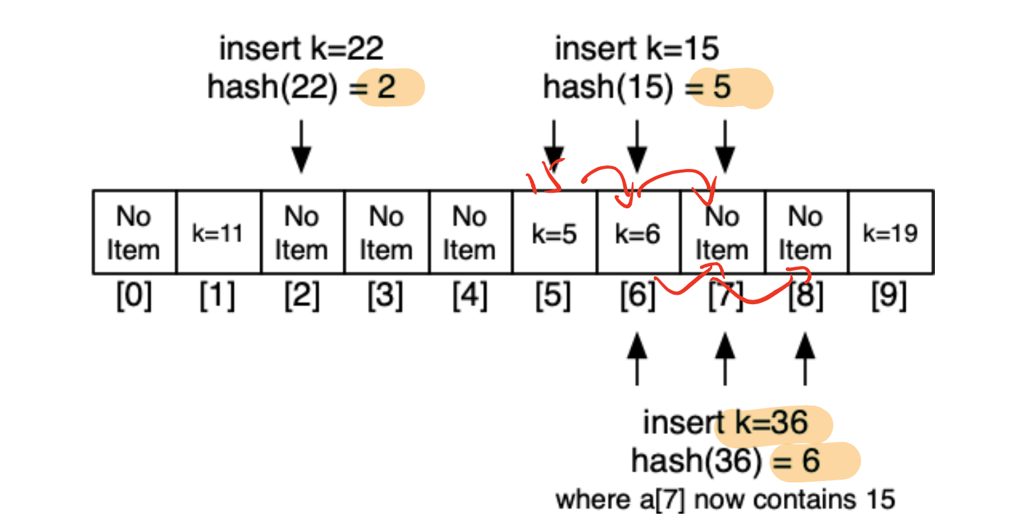
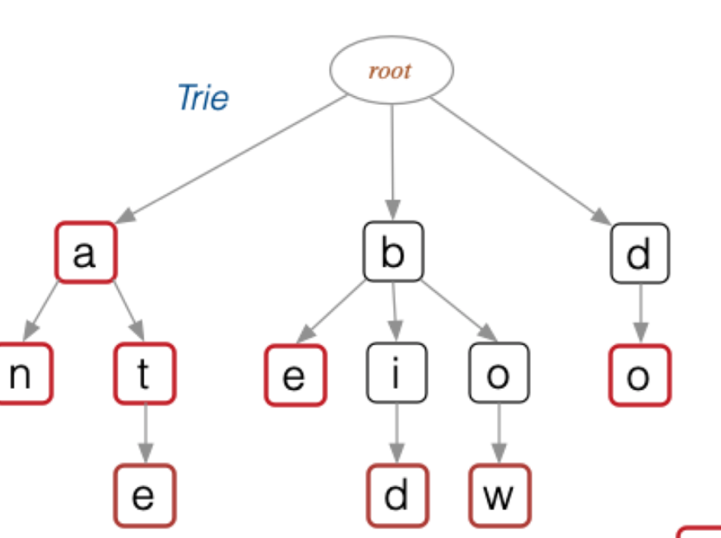
* 1. edges in G are not directed
  2. Kruskal's Algorithm: 
  3. Prim's Algorithm: 

**Shortest Path Algorithms**: Dijkstra's Algorithm

**O(n2) Sorting Algorithms**

1. selection sort ... simple, non-adaptive sort-find the smallest element, put it into first array slot; find second smallest element,
2. bubble sort ... simple, adaptive sort: ****
3. insertion sort ... simple, adaptive sort: ****
4. shellsort ... improved version of insertion sort: ****

****

1. **Quicksort:**
   1. little benefit from partitioning when size < 5
2. **Mergesort:**
   1. 
   2. Disadvantage over quicksort: need extra storage O(N)
3. **Bottom-up Mergesort**
   1. 
4. **Hashing**
   1. search cost in hash table is O(1)
   2. Linear: 
   3. Double hashing
5. **Tries: **
6. **Heaps:** 
   1. Insertion cost = O(logN), Deletion cost = O(logN)
   2. 