Data Structure operations and time complexity:

Arrays:

Set, Check element at a particular index: O(1)

Searching: O(n) if array is unsorted and O(log n) if array is sorted and something like a binary search is used,

As pointed out by Aivean, there is no Delete operation available on Arrays. We can symbolically delete an element by setting it to some specific value, e.g., -1, 0, etc. depending on our requirements

Similarly, insert for arrays is basically Set as mentioned in the beginning

--

Linked List:

Inserting: O (1), if done at the head, O(n) if anywhere else since we have to reach that position by traversing the linked list linearly.

Deleting: O (1), if done at the head, O(n) if anywhere else since we have to reach that position by traversing the linked list linearly.

Searching: O(n)

--

Stack:

Push**:** O(1)

Pop**:** O(1)

Top**:** O(1)

Search **(Something like lookup, as a special operation):** O(n)

--

Queue/Deque/Circular Queue:

Insert**:** O(1)

Remove**:** O(1)

Size**:** O(1)

--

Binary Search Tree:

Insert, delete and search**: Average case:** O(log n)**, Worst Case:** O(n)

--

HashMap/Hashtable/HashSet:

Insert/Delete**:** O(1) **amortized**

Re-size/hash: O(n)

Contains: O(1)

Heap/PriorityQueue (min/max):

Find Min/Find Max: O(1)

Insert: O(log n)

Delete Min/Delete Max: O(log n)

Extract Min/Extract Max: O(log n)

Lookup, Delete (if at all provided): O(n), we will have to scan all the elements as they are not ordered like BST

The data structure is a particular way of organizing data in a computer. The developer must choose the appropriate data structure for better performance. If the developer chooses a bad data structure, the system does not perform well. This article explains each data structure's advantages and usage.

Linked List

The linked list is a data structure that links each node to the next node. The developer can use the linked list in the following use cases.

- When the developer needs constant time for insertion and deletion.
- When the data dynamically grows.
- Do not access random elements from the linked list.
- Insert the element in any position of the list.

Circular Linked List

A circular linked list is a linked list in which the link field of the tail node link to the head node. The developer can use a circular linked list in the following use cases.

- Develop the buffer memory.
- Represent a deck of cards in a game.
- Browser cache allows hitting the BACK button.
- Implement the Most Recently Used (MRU) list.
- Undo functionality in Photoshop or Word.

Doubly Linked List

Doubly linked is a data structure in which each node contains data and two links. One link point to the previous node and another link point to the next node. The developer can use a doubly linked list in the following uses cases.

- Easier to delete the node from the doubly linked list.
- It can be iterated in reverse order without recursion implementation.
- Insert or remove from double-linked lists faster.

Stack

The stack is a last-in, first-out data structure. The developer can use the stack in the following use cases.

- Expression evaluation and syntax parsing.
- Finding the correct path in a maze using backtracking.
- Runtime memory management.
- Recursion function.

Queue

The queue is a first in, first-out (FIFO) data structure. The developer can use Queue in the following use cases.

- Use a queue when the developer wants an order.
- Processed in First In First Out order.
- If the developer wants to add or remove both ends, they can use the queue or a double-ended queue.

Binary Tree

A binary tree is a tree data structure in which each node has at most two child nodes. The developer can use Binary Tree in the following use cases.

- Find the name in the phone book.
- Sorted traversal of the tree.
- Find the next closest element.
- Find all elements less than or greater than a certain value.

Binary Search Tree

A binary search tree is a tree data structure in which the root node is less than or equal to the left subtree and greater than or equal to the right subtree. The developer can use Binary Search Tree in the following use cases.

- Binary Search Trees are memory-efficient.
- Use when the data need to be sorted.
- Search can be done for a range of values.
- Height balancing helps to reduce the running time.

Heap

A heap is a specialized tree-based abstract data type that satisfies the heap property. The developer can use Heap in the following use cases.

- Implement Priority Queue.
- whenever the developer wants quick access to the largest (or smallest) item.
- Good for selection algorithms (finding the min or max).
- Operations tend to be faster than for a binary tree.
- Heap sort sorting methods being in-place and with no quadratic worst-case scenarios.
- Graph algorithms are using heaps as internal traversal data structures, the run time will be reduced by polynomial order.

Hashing

Hash table is a data structure used to implement an associative array, a structure that can map keys to values. The developer can use a Hash table in the following use cases.

- Constant time operation.
- Inserts are generally slow, reads are faster than trees.
- Hashing is used so that searching a database can be done more efficiently.
- Internet routers use hash tables to route the data from one computer to another.
- The Internet search engine uses a hash functions effectively.

Graph

The graph is an abstract data type that is meant to implement the graph and directed graph concepts from mathematics. The developer can use Graph in the following use cases.

- Networks have many uses in the practical side of graph theory.
- Finding the shortest path between the cities.
- Solve the maze game.
- Find the optimized route between the cities.

Red-Black Tree

Red–black tree is a binary search tree with an extra bit of data per node, its color, which can be either red or black. The developer can use Red-Black Tree in the following use cases.

- Java TreeMap and C++ map implemented using Red Block Tree.
- Computational Geometry Data structures.
- Scheduler applications.

Array

The array is a data structure to store the same type of elements continuously. The developer can use an array in the following use cases.

- Need access to the elements using the index.
- Know the size of the array before defining the memory.
- Speed when iterating through all the elements in the sequence.
- The array takes less memory compare than a linked list.

Matrix

Matrix is a data structure that stores the data using rows and columns. The developer can use Matrix in the following use cases.

- Matrix arithmetic in graphic processing algorithms.
- Represent the graph.
- Represent quadratic forms and linear algebra solution.

B-Tree

B-tree is a tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions in logarithmic time. The developer can use B-Tree in the following use cases.

- File systems.
- Database operations.

Splay Tree

A splay tree is a self-adjusting binary search tree with the additional property that recently accessed elements are quick to access again. The developer can use Splay Tree in the following use cases.

- When the developer wants to access to the recent data easily.
- Allow duplicate items.
- Simple implementation and take less memory.
- When the application deals with a lot of data, use the splay-tree.

AVL Tree

AVL tree, the shape of the tree is constrained at all times such that the tree shape is balanced. The height of the tree never exceeds O(log n). The developer can use AVL Tree in the following use cases.

- When the developer wants to control the tree height outside -1 to 1 range.
- Fast looking element.

Trie

A Trie (digital tree and sometimes radix tree or prefix tree), is an ordered tree data structure that is used to store a dynamic set or associative array where the keys are usually strings. The developer can use Trie in the following use cases.

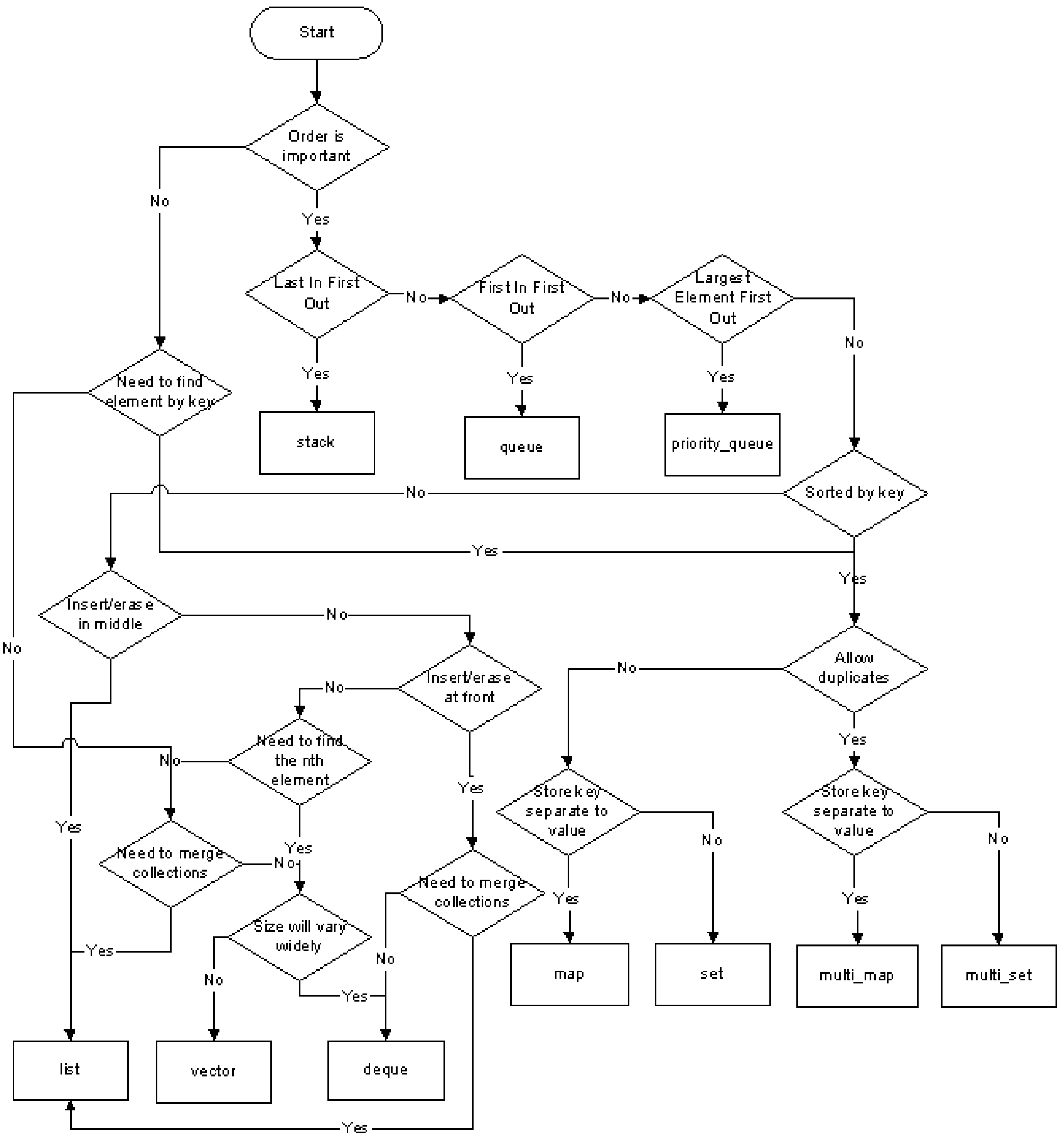
- Fixed dictionary and want to look up quickly.
- Require less storage for a large dictionary.
- Matching sentences during string matching.
- Predictable O(k) lookup time where k is the size of the key.
- Lookup can take less than k time if it's not there.
- Supports ordered traversal.
- No need for a hash function.
- Deletion is straightforward.

Minimum Spanning Tree

A spanning tree of that graph is a subgraph that is a tree and connects all the vertices together. A minimum spanning tree (MST) or minimum weight spanning tree is then a spanning tree with weight less than or equal to the weight of every other spanning tree. The developer can use Minimum Spanning Tree in the following use cases.

- Describe financial markets.
- Handwriting recognition of mathematical expressions.
- Image registration and segmentation.
- Constructing trees for broadcasting in computer networks.

We discussed different data structures and uses cases to choose the appropriate data structure. When the candidate attends the technical coding interview or uses the application programming interface in software development, the candidate must choose the correct data structure. If the candidate uses the incorrect data structure, it may work. But the programs may fail with more data or with the different use cases.


Start
Order is important
No
Yes
Last In First Out
No
Yes
First In First Out
No
Yes
Largest Element First Out
Yes
No
stack
queue
priority_queue
Need to find element by key
Sorted by key
No
Yes
Yes
Insert/erase in middle
No
Allow duplicates
No
Insert/erase at front
No
No
Yes
Need to find the nth element
No
Yes
Store key separate to value
Store key separate to value
Yes
No
No
Yes
Need to merge collections
No
Yes
Need to merge collections
Size will vary widely
Yes
Yes
Yes
No
No
Yes
map
set
multi_map
multi_set
list
vector
deque
Yes