
DSA
CHEATSHEET
Swipe

Arrays & Strings

Stores data elements based on an sequential, most commonly 0 based, index.

Time Complexity

- **Indexing:** Linear array: O(1), Dynamic array: O(1)
- **Search:** Linear array: O(n), Dynamic array: O(n)
- **Optimized Search:** Linear array: O(log n), Dynamic array: O(log n)
- **Insertion:** Linear array: n/a, Dynamic array: O(n)

Bonus:

- type[] name = {val1, val2, ...}
- Arrays.sort(arr) -> O(n log(n))
- Collections.sort(list) -> O(n log(n))
- int digit = '4' - '0' -> 4
- String s = String.valueOf('e') -> "e"
- (int) 'a' -> 97 (ASCII)
- new String(char[] arr ['a','e'] -> "ae"
- (char) ('a' + 1) -> 'b'
- Character.isLetterOrDigit(char) -> true/false
- new ArrayList<>(anotherList); -> list w/ items
- StringBuilder.append(char||String)

Linked List

Stores data with nodes that point to other nodes.

Time Complexity

- **Indexing:** O(n)
- **Search:** O(n)
- **Optimized Search:** O(n)
- **Append:** O(1)
- **Prepend:** O(1)
- **Insertion:** O(n)

Linked List

Stores data with nodes that point to other nodes.

Time Complexity

- **Indexing:** O(n)
- **Search:** O(n)
- **Optimized Search:** O(n)
- **Append:** O(1)
- **Prepend:** O(1)
- **Insertion:** O(n)

HashTable

Stores data with key-value pairs.

Time Complexity

- **Indexing:** O(1)
- **Search:** O(1)
- **Insertion:** O(1)

Bonus:

- {1, -1, 0, 2, -2} into map

HashMap {-1, 0, 2, 1, -2} -> any order

LinkedHashMap {1, -1, 0, 2, -2} -> insertion order

TreeMap {-2, -1, 0, 1, 2} -> sorted

- Set doesn't allow duplicates.
- map.getOrDefaultValue(key, default value)

Stack/Queue/Deque

Stack	Queue	Deque	Heap
Last In First Out	First In Last Out	Provides first/last	Ascending Order
push(val)	offer(val)	offer(val)	offer(val)
pop()	poll()	poll()	poll()
peek()	peek()	peek()	peek()

Implementation in Java:

- Stack<E> stack = new Stack();
- Queue<E> queue = new LinkedList();
- Deque<E> deque = new LinkedList();
- PriorityQueue<E> pq = new PriorityQueue();

DFS & BFS Big O Notation

	Time	Space
DFS	O(E+V)	O(Height)
BFS	O(E+V)	O(Length)

V & E -> where V is the number of vertices and E is the number of edges.

Height -> where h is the maximum height of the tree.

Length -> where l is the maximum number of nodes in a single level.