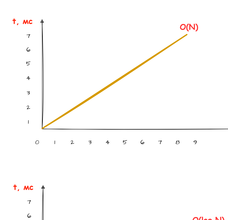
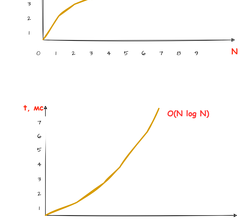
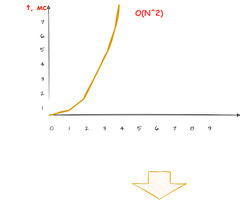
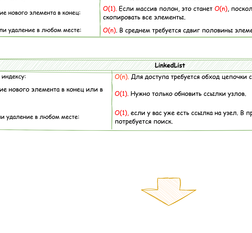
Big O notation - оценка сложности алгоритма. Показывает худший случай времени выполнения алгоритма в зависимости от размера входных данных









ArrayList	
Доступ по индексу:	O(1). ArrayList мгновенно находит элемент с любым индексом.
Добавление нового элемента в конец:	O(1). Если массив полон, это станет O(n), поскольку придется создать новый массив и скопировать все элементы.
Вставка или удаление в любом месте:	O(n). В среднем требуется сдвиг половины элементов, поэтому это O(n).

LinkedList	
Доступ по индексу:	O(n). Для доступа требуется обход цепочки связей.
Добавление нового элемента в конец или в начало:	O(1). Нужно только обновить ссылки узлов.
Вставка или удаление в любом месте:	O(1), если у вас уже есть ссылка на узел. В противном случае это O(n), так как вам потребуется поиск.

HashSet	
Вставка, удаление или поиск:	O(1) в среднем. Если возникают коллизии, это может упасть до O(n).

TreeSet	
Вставка, удаление или поиск:	O(log(n)). Бинарное дерево поиска обеспечивает логарифмическую скорость.

LinkedHashSet	
Вставка, удаление или поиск:	O(1) в среднем. LinkedHashSet работает почти так же, как и HashSet, но он также поддерживает двунаправленный связный список, позволяющий итерацию с порядком вставки. Однако при коллизиях это может упасть до O(n).

HashMap	
Вставка, удаление или поиск:	O(1) в среднем. Как и в случае HashSet, при коллизиях эффективность может упасть до O(n).

TreeMap	
Вставка, удаление или поиск:	O(log(n)). TreeMap реализует SortedMap и использует структуру данных дерева. Это гарантирует, что операции вставки, удаления и поиска будут занимать логарифмическое время.

LinkedHashMap	
Вставка, удаление или поиск:	O(1) в среднем. LinkedHashMap работает подобным образом HashMap, но поддерживает двунаправленный связный список для сохранения порядка вставки. Также как и у HashMap и HashSet, эффективность при поиске может снизиться до O(n) в случае коллизий.

	get	add	contains	next	remove(0)	iterator.remove
ArrayList	O(1)	O(1)	O(n)	O(1)	O(n)	O(n)
LinkedList	O(n)	O(1)	O(n)	O(1)	O(1)	O(1)
CopyOnWrite-ArrayList	O(1)	O(n)	O(n)	O(1)	O(n)	O(n)

	add	contains	next	notes
HashSet	O(1)	O(1)	O(h/n)	h is the table capacity
LinkedHashSet	O(1)	O(1)	O(1)	
CopyOnWriteArraySet	O(n)	O(n)	O(1)	
EnumSet	O(1)	O(1)	O(1)	
TreeSet	O(log n)	O(log n)	O(log n)	
ConcurrentSkipListSet	O(log n)	O(log n)	O(1)	

	get	containsKey	next	Notes
HashMap	O(1)	O(1)	O(h/n)	h is the table capacity
LinkedHashMap	O(1)	O(1)	O(1)	
IdentityHashMap	O(1)	O(1)	O(h/n)	h is the table capacity
EnumMap	O(1)	O(1)	O(1)	
TreeMap	O(log n)	O(log n)	O(log n)	
ConcurrentHashMap	O(1)	O(1)	O(h/n)	h is the table capacity
ConcurrentSkipListMap	O(log n)	O(log n)	O(1)	

	offer	peek	poll	size
PriorityQueue	O(log n)	O(1)	O(log n)	O(1)
ConcurrentLinkedQueue	O(1)	O(1)	O(1)	O(n)
ArrayBlockingQueue	O(1)	O(1)	O(1)	O(1)
LinkedBlockingQueue	O(1)	O(1)	O(1)	O(1)
PriorityBlockingQueue	O(log n)	O(1)	O(log n)	O(1)
DelayQueue	O(log n)	O(1)	O(log n)	O(1)
LinkedList	O(1)	O(1)	O(1)	O(1)
ArrayDeque	O(1)	O(1)	O(1)	O(1)
LinkedBlockingDeque	O(1)	O(1)	O(1)	O(1)

"h/n" - это среднее количество элементов в каждом «ящике» хэш-таблицы