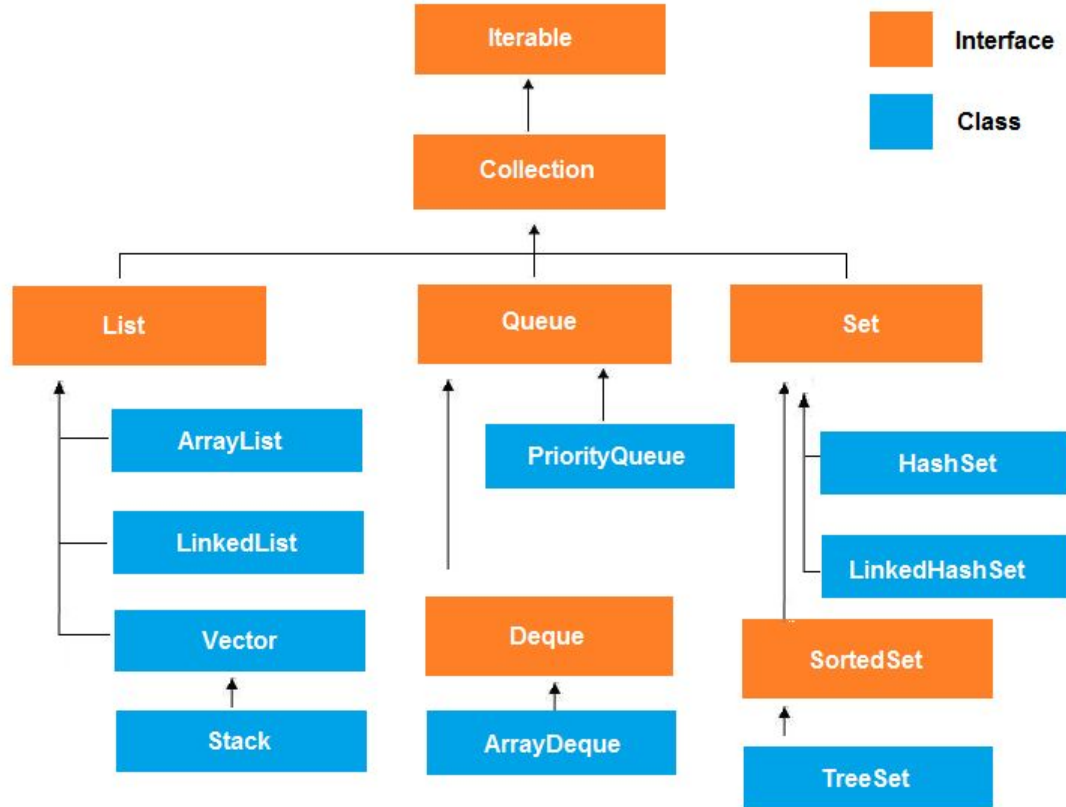


Collection



Collections

Collections trong java là một khuôn khổ cung cấp một kiến trúc để lưu trữ và thao tác tới nhóm các đối tượng. Tất cả các hoạt động mà bạn thực hiện trên một dữ liệu như tìm kiếm, phân loại, chèn, xóa,... có thể được thực hiện bởi Java Collections.

The screenshot shows the Tiki.vn website during a 'BACK 2 SCHOOL' promotion. The main banner features a family scene with a child and a 'Giá Tốt' (Good Price) tag. Below the banner, there are several promotional tiles and product categories. The left sidebar lists various product categories, and the right sidebar shows more product categories and a 'Giỏ hàng' (Shopping Cart) icon.

BACK 2 SCHOOL **Giá Tốt** **HÓT THÊM QUÀ**

Tiki App - Ưu đãi ngay trên tay | Phiếu quà tặng | Khuyến Mãi HOT | Tiki Global - Mua hàng từ nước ngoài | Đặt vé máy bay online | Bán hàng cùng Tiki

TIMI.VN | SẢN PHẨM SẴN HÀNG KỊCH LIỆT | Tìm sản phẩm, danh mục hay thương hiệu mong muốn... | Tìm kiếm

Theo dõi đơn hàng | Thông báo của tôi | Đăng nhập Tài khoản | Giỏ hàng 0

DANH MỤC SẢN PHẨM | Bạn muốn giao hàng tới đâu? | Sản phẩm bạn đã xem | 2-tiếng nhận hàng Hàng chục nghìn sản phẩm | Tất cả sản phẩm 100% chính hãng | 30 ngày đổi trả dễ dàng

Điện Thoại - Máy Tính Bảng | Tivi - Thiết bị nghe nhìn | Phụ Kiện - Thiết Bị Số | Laptop - Thiết bị IT | Máy Ảnh - Quay Phim | Điện Gia Dụng - Điện Lạnh | Nhà Cửa Đời Sống | Hàng Tiêu Dùng - Thực Phẩm | Đồ chơi, Mẹ & Bé | Làm Đẹp - Sức Khỏe | Thời trang - Phụ kiện | Thể Thao - Dã Ngoại | Xe Máy, Ô tô, Xe Đạp | Sách, VPP & Quà Tặng | Voucher - Dịch Vụ - Thẻ Cào

Bạn ơi hãy chọn địa chỉ giao hàng của bạn để có trải nghiệm mua sắm tốt nhất nhé

2 BACK 2 SCHOOL **Giá Tốt** **HÓT THÊM QUÀ**

XEM NGAY

CƠ LỘC GIÁ SỐC KANGAROO | ƯU ĐÃI ĐẾN 50%+ | Coupon giảm thêm 10% | ĐỒNG HỒ NAM DÂY DA TIMEX METROPOLITAN | GIÁ CHỈ 1.377.000đ

SỮA TẮM JOHNSON'S BABY CHỨA SỮA VÀ YÊN MẠCH | GIẢM ĐẾN 20% | GIẢM SỐC - ĐẬP TAN NẮNG HÈ | GIẢM THÊM 20% | Cho đơn hàng Sách giao tại Hà Nội

Khuyến mãi Tã sữa LỚN NHẤT NĂM | COUPON GIẢM THÊM ĐẾN 130K | 01/07 - 31/07/2018 | XEM NGAY

SẢM TẠO GIÁ TỐT | Trả góp 0% 12 tháng Bảo hành chính hãng tại Apple Việt Nam | SẢM MÁY ẢNH VI VU HÈ | TRẢ GÓP 0% | Nhập mã giảm thêm đến 1.000.000đ

Set Interface

Set (tập hợp) là kiểu dữ liệu mà bên trong nó mỗi phần tử chỉ xuất hiện duy nhất một lần (tương tự như tập hợp trong toán học vậy) và tập hợp chưa được sắp xếp

TP Hồ Chí Minh (5)

Hà Nội (1)

HCM, Bình Dương, Long An, Đồng Nai (1)

SỐ LƯỢNG SIM

2 Sim (186)

1 Sim (6)

DUNG LƯỢNG PIN

Từ 3.500 mAh đến 5.000 mAh (148)

Từ 2.500 mAh đến 3.500 mAh (92)

Dưới 2.500 mAh (31)

Trên 5.000 mAh (9)

KÍCH THƯỚC MÀN HÌNH

Trên 5.5 inch (247)

Từ 5 inch đến 5.5 inch (58)

Từ 4.5 inch đến 5 inch (23)

Đang cập nhật (3)

Dưới 4.5 inch (1)

ĐỘ PHÂN GIẢI MÀN HÌNH

2K (2560 x 1440 pixels) (4)

Full HD+ (1080 x 2246 pixels) (2)

HD+ (720 x 1560 Pixels) (11)


Full HD (1920 x 1080 pixels) (14)


Retina (750 x 1334 pixels) (6)

2K+ (1440 x 2960 Pixels) (2)

Full HD+ (1080 x 2248 pixels) (4)

Xem sản phẩm giao từ nước ngoài - [Tim hiểu](#)







TIKINOW | Điện Thoại Xiaomi Redmi 7 (2GB/16GB) - Hàng...

2.390.000 đ -20%

2.990.000 đ

★★★★★ (13 nhận xét)






TIKINOW | Điện Thoại Samsung Galaxy Note 10 Plus...


24.969.000 đ -7%

26.990.000 đ

★★★★★ (30 nhận xét)

Xem sản phẩm hỗ trợ TikiNOW giao nhanh - [Tim hiểu](#)






TIKINOW | Điện Thoại iPhone 6s Plus 32GB VN/A - Hàng...


8.250.000 đ -25%

10.990.000 đ

Trả góp 0% chỉ 687.500 đ/tháng

★★★★★ (168 nhận xét)







TIKINOW | Điện Thoại Nokia 5.1 Plus - Hàng Chính Hãng

2.690.000 đ -46%


4.990.000 đ


★★★★★ (205 nhận xét)







TIKINOW | Điện Thoại Samsung Galaxy A10 (32GB/2GB) -...







TIKINOW | Điện Thoại Meizu C9 - Hàng Chính Hãng





Điện Thoại Samsung Galaxy A20 (32GB/3GB) -...





TIKINOW | Điện Thoại OPPO A3s (16GB/2GB) - Hàng Chính...

Các điện thoại không được trùng nhau

List Interface

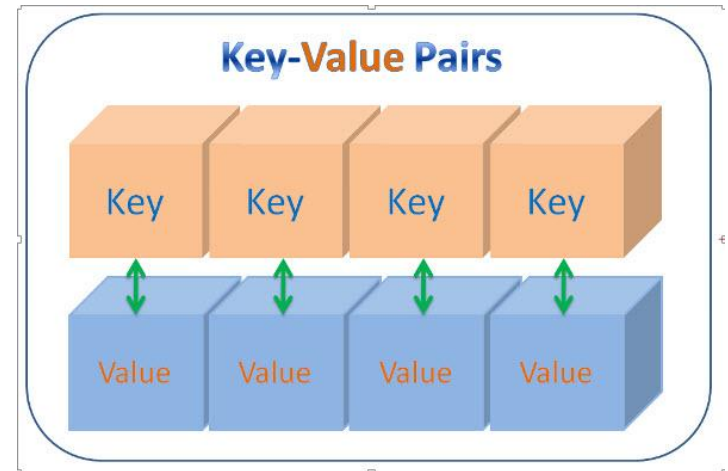
List (danh sách) là cấu trúc dữ liệu tuyến tính trong đó các phần tử được sắp xếp theo một thứ tự xác định. Cho phép các phần tử được trùng lặp nhau

Home	Add a Product	Import ▾	Product List 34	Orders	Settings ▾	Reports ▾	My Supplier Page	Back to Home Page
Orders								
Autoincrement ID	From	To	Status ▾	Filter	Export to CSV			
Order #	Purchased On	Bill to Name	Ship to Name	Subtotal	Income	Status	Actions	
#000000028	2017-04-12 10:29:28	Supplier Demo	Supplier Demo	\$45.00	\$40.50	Pending	View	Ship
#000000027	2017-03-31 11:50:22	Veronica Costello	Veronica Costello	\$120.00	\$108.00	Complete	View	Ship
#000000026	2017-03-31 11:35:41	Veronica Costello	Veronica Costello	\$122.00	\$109.80	Complete	View	Ship
#000000025	2017-03-31 11:33:01	Veronica Costello	Veronica Costello	\$20.00	\$18.00	Complete	View	Ship
#000000020	2017-03-30 14:06:19	Supplier Demo	Supplier Demo	\$45.00	\$40.50	Pending	View	Ship
#000000015-1	2017-03-14 15:17:43	Veronica Costello	Veronica Costello	\$10.00	\$9.00	Complete	View	Ship
#000000015	2017-03-14 15:14:49	Veronica Costello	Veronica Costello	\$10.00	\$9.00	Canceled	View	
#000000014	2017-03-14 14:40:32	Garion Bracken	Garion Bracken	\$10.00	\$9.00	Complete	View	Ship
#000000012	2017-03-01 07:17:56	Supplier Demo	Supplier Demo	\$30.00	\$27.00	Complete	View	Ship

Ví dụ như Veronica

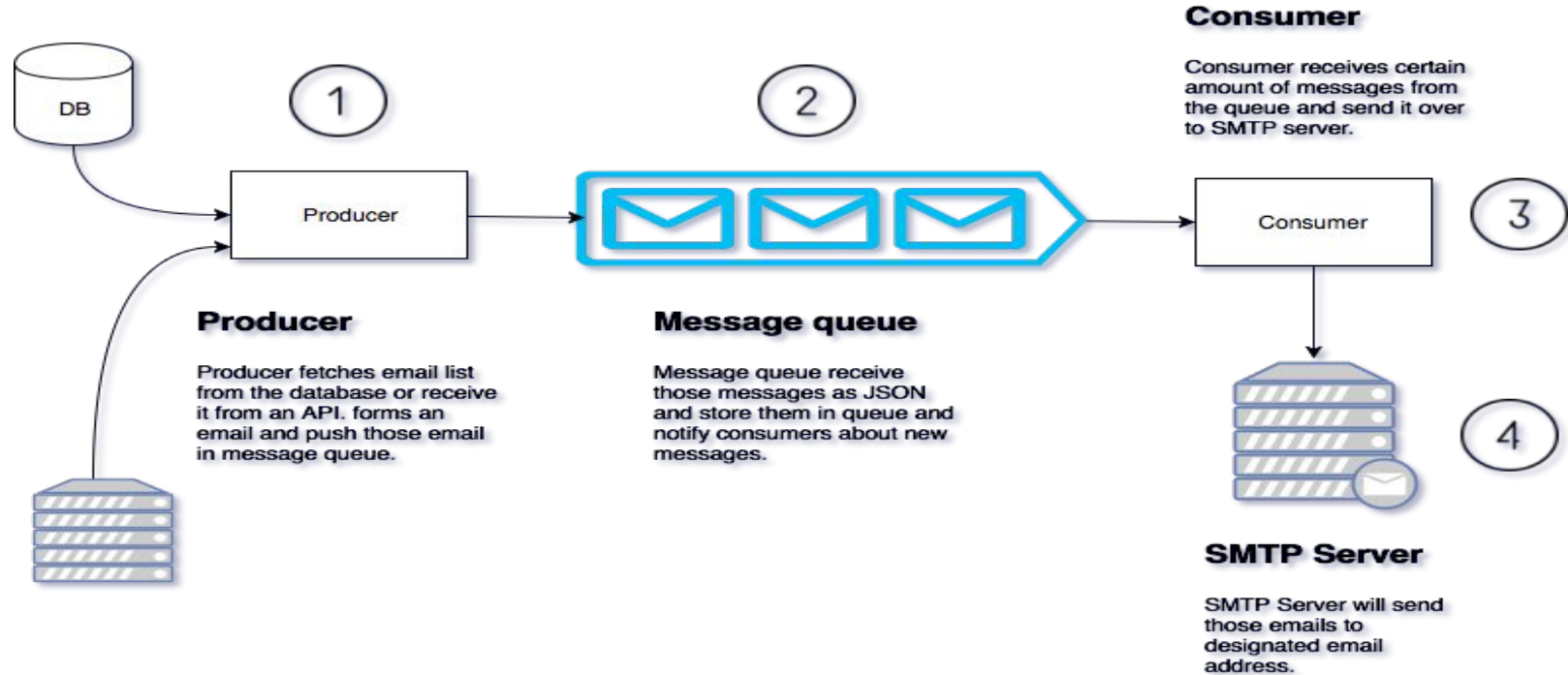
Map Interface

Map (đồ thị/ánh xạ) là kiểu dữ liệu cho phép ta quản lý dữ liệu theo dạng **cặp key-value**, trong đó **key là duy nhất và tương ứng với 1 key là một giá trị value**. Map Interface không kế thừa từ Collection Interface mà đây là 1 interface độc lập với các phương thức của riêng mình.



Queue Interface

Queue (hàng đợi) là kiểu dữ liệu nổi tiếng với kiểu vào ra FIFO (first-in-first-out hay vào trước ra trước)



Performance Big O

Khái niệm Big O hoặc với tên gọi khác trong tiếng Việt là “độ phức tạp của thuật toán” là thuật ngữ thường dùng để chỉ khoảng thời gian tiêu hao để chạy một thuật toán. Các lập trình viên thường sử dụng Big O như một phương tiện để so sánh mức độ hiệu quả của nhiều cách xử lý khác nhau cho cùng một vấn đề.

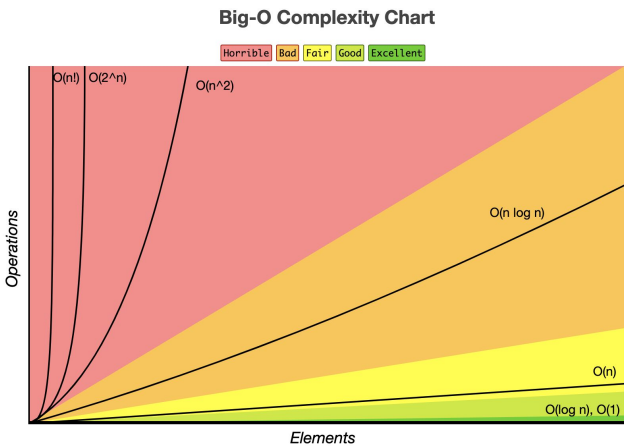
Khái niệm Big O có thể đúc kết thành một câu ngắn gọn như sau: Thời gian chạy nhanh như thế nào, còn tùy thuộc vào giá trị đầu vào -input, vì giá trị input sẽ lớn dần lên khi chương trình chạy.

Performance Big O

Giá trị bất biến: Rõ ràng là Big O chỉ giải quyết vấn đề dựa trên tối ưu hóa phương thức code, bỏ qua các giá trị bất biến như đơn vị thời gian, size tiêu hao thực của bộ nhớ. Điều này rất bất lợi trong thời đại công nghệ phần cứng đang phát triển nhanh như vũ bão, tôi đang sử dụng một chiếc máy tính đời cổ để chạy một chương trình tốn đến 5 giờ, tôi miệt mài tốn công sức nghiên cứu tối ưu cách viết code để cải thiện chương trình chạy nhanh hơn 30 phút, thì cũng không bằng tôi ra tiệm sửa máy và nâng cấp phần cứng chiếc máy lên, khi đó máy tôi chạy chương trình nhanh hơn tận 4 tiếng.

Phương án tối ưu hóa code nóng vội: Đôi khi sử dụng phương thức Big O để tối ưu hóa code có thể tiết kiệm được thời gian chạy chương trình và giảm tiêu hao bộ nhớ, nhưng đồng thời sẽ khiến code khó đọc hoặc rất tốn thời gian suy nghĩ để viết. Đối với những lập trình viên trẻ, họ nên học cách viết code một cách tường minh, rõ ràng, dễ đọc, dễ hiểu và dễ bảo dưỡng dù chương trình họ viết ra có thể không được tiết kiệm tài nguyên như các lập trình viên lành nghề khác.

Nói như vậy không có nghĩa là các lập trình viên trẻ chẳng cần phải quan tâm gì đến Big O, vì bất kỳ một lập trình viên nào (dù trẻ, lành nghề, chuyên gia hay mới bước vào nghề) cũng luôn phải chú ý cân bằng các yếu tố thời gian chạy chương trình, thời gian chạy thuật toán, bộ nhớ tiêu hao, khả năng bảo dưỡng, tính tường minh của những dòng code.

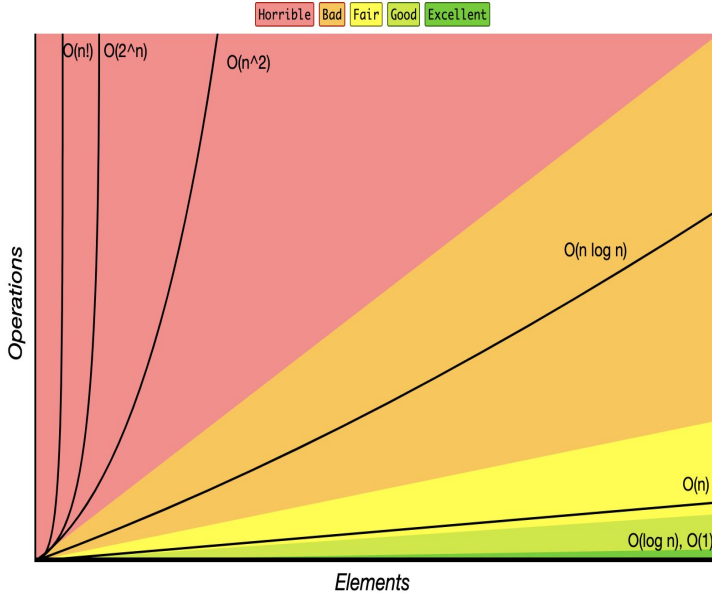


Common Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
<u>Array</u>	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>Stack</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Queue</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Singly-Linked List</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Doubly-Linked List</u>	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$	$\theta(n)$	$\theta(1)$	$\theta(1)$	$\theta(n)$
<u>Skip List</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n \log(n))$
<u>Hash Table</u>	N/A	$\theta(1)$	$\theta(1)$	$\theta(1)$	N/A	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>Binary Search Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>Cartesian Tree</u>	N/A	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	N/A	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$
<u>B-Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$
<u>Red-Black Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$
<u>Splay Tree</u>	N/A	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	N/A	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$
<u>AVL Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$
<u>KD Tree</u>	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(\log(n))$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$	$\theta(n)$

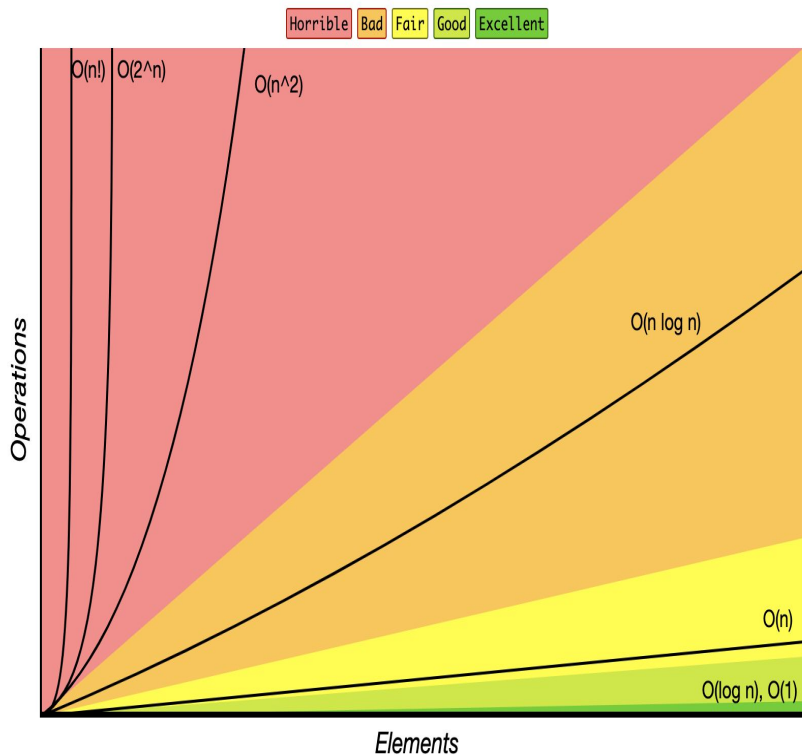
Array Sorting Algorithms

Big-O Complexity Chart



Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
<u>Quicksort</u>	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n^2)$	$O(\log(n))$
<u>Mergesort</u>	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n \log(n))$	$O(n)$
<u>Timsort</u>	$\Omega(n)$	$\theta(n \log(n))$	$O(n \log(n))$	$O(n)$
<u>Heapsort</u>	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n \log(n))$	$O(1)$
<u>Bubble Sort</u>	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$	$O(1)$
<u>Insertion Sort</u>	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$	$O(1)$
<u>Selection Sort</u>	$\Omega(n^2)$	$\theta(n^2)$	$O(n^2)$	$O(1)$
<u>Tree Sort</u>	$\Omega(n \log(n))$	$\theta(n \log(n))$	$O(n^2)$	$O(n)$
<u>Shell Sort</u>	$\Omega(n \log(n))$	$\theta(n(\log(n))^2)$	$O(n(\log(n))^2)$	$O(1)$
<u>Bucket Sort</u>	$\Omega(n+k)$	$\theta(n+k)$	$O(n^2)$	$O(n)$
<u>Radix Sort</u>	$\Omega(nk)$	$\theta(nk)$	$O(nk)$	$O(n+k)$
<u>Counting Sort</u>	$\Omega(n+k)$	$\theta(n+k)$	$O(n+k)$	$O(k)$
<u>Cubesort</u>	$\Omega(n)$	$\theta(n \log(n))$	$O(n \log(n))$	$O(n)$

Big-O Complexity Chart



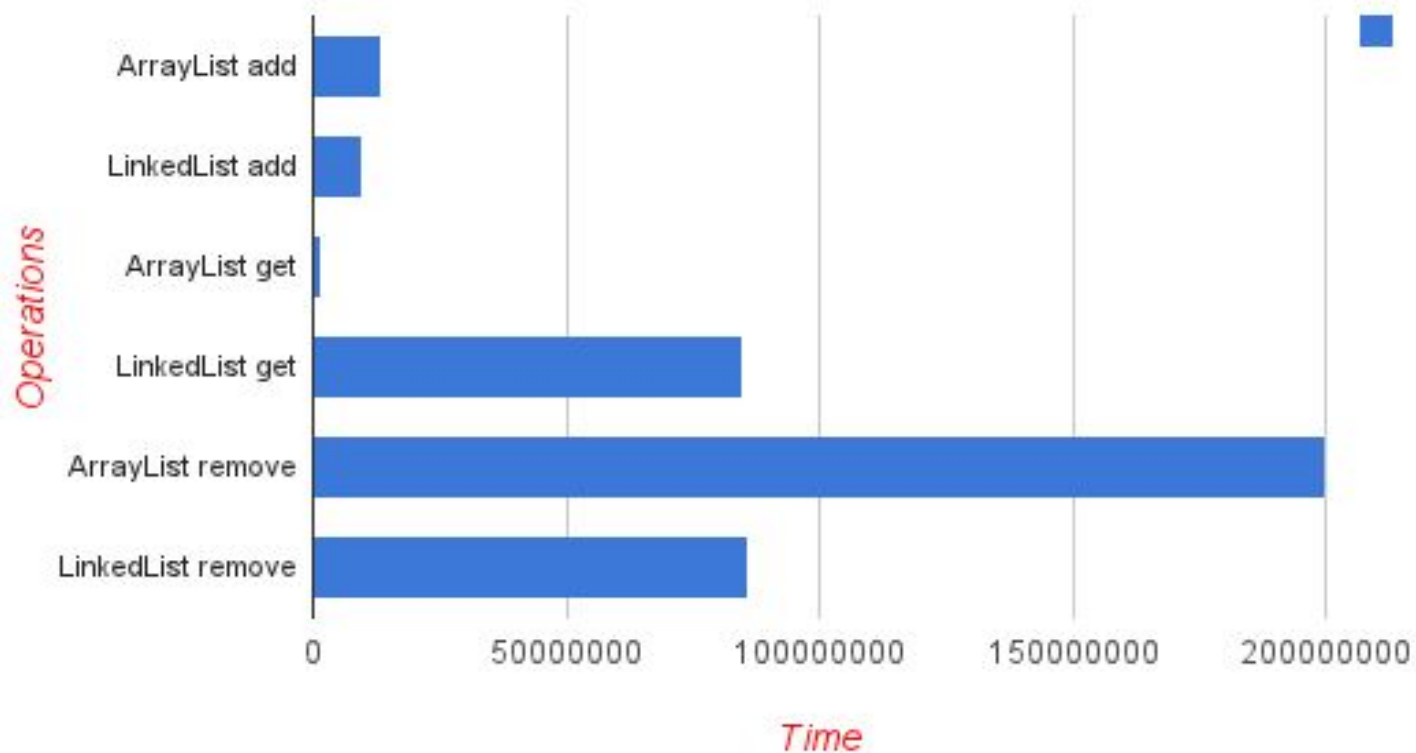
List	Add	Remove	Get	Contains	Next	Data Structure
ArrayList	$O(1)$	$O(n)$	$O(1)$	$O(n)$	$O(1)$	Array
LinkedList	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$	Linked List
CopyOnWriteArrayList	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(1)$	Array

Set	Add	Remove	Contains	Next	Size	Data Structure
HashSet	$O(1)$	$O(1)$	$O(1)$	$O(h/n)$	$O(1)$	Hash Table
LinkedHashSet	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	Hash Table + Linked List
EnumSet	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	Bit Vector
TreeSet	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(1)$	Red-black tree
CopyOnWriteArraySet	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	Array
ConcurrentSkipListSet	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(1)$	$O(n)$	Skip List

Queue	Offer	Peak	Poll	Remove	Size	Data Structure
PriorityQueue	$O(\log n)$	$O(1)$	$O(\log n)$	$O(n)$	$O(1)$	Priority Heap
LinkedList	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	Array
ArrayDeque	$O(1)$	$O(1)$	$O(1)$	$O(n)$	$O(1)$	Linked List
ConcurrentLinkedQueue	$O(1)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	Linked List
ArrayBlockingQueue	$O(1)$	$O(1)$	$O(1)$	$O(n)$	$O(1)$	Array
PriorityBlockingQueue	$O(\log n)$	$O(1)$	$O(\log n)$	$O(n)$	$O(1)$	Priority Heap
SynchronousQueue	$O(1)$	$O(1)$	$O(1)$	$O(n)$	$O(1)$	None!
DelayQueue	$O(\log n)$	$O(1)$	$O(\log n)$	$O(n)$	$O(1)$	Priority Heap
LinkedBlockingQueue	$O(1)$	$O(1)$	$O(1)$	$O(n)$	$O(1)$	Linked List

Map	Get	ContainsKey	Next	Data Structure
HashMap	$O(1)$	$O(1)$	$O(h / n)$	Hash Table
LinkedHashMap	$O(1)$	$O(1)$	$O(1)$	Hash Table + Linked List
IdentityHashMap	$O(1)$	$O(1)$	$O(h / n)$	Array
WeakHashMap	$O(1)$	$O(1)$	$O(h / n)$	Hash Table
EnumMap	$O(1)$	$O(1)$	$O(1)$	Array
TreeMap	$O(\log n)$	$O(\log n)$	$O(\log n)$	Red-black tree
ConcurrentHashMap	$O(1)$	$O(1)$	$O(h / n)$	Hash Tables
ConcurrentSkipListMap	$O(\log n)$	$O(\log n)$	$O(1)$	Skip List

Performance Test



List Interface

List (danh sách) là cấu trúc dữ liệu tuyến tính trong đó các phần tử được sắp xếp theo một thứ tự xác định. List Interface định nghĩa các phương thức để tương tác với list cũng như các phần tử bên trong list. Tương tự như Set Interface, List Interface cũng được kế thừa và có đầy đủ các phương thức của Collection Interface.

- **ArrayList**: là 1 class dạng list được implement dựa trên mảng có kích thước thay đổi được.
- **LinkedList**: là một class dạng list hoạt động trên cơ sở của cấu trúc dữ liệu danh sách liên kết đôi (double-linked list)
- **Vector**: là 1 class thực thi giao diện List Interface, có cách thức lưu trữ như mảng tuy nhiên có kích thước thay đổi được, khá là tương tự với ArrayList, tuy nhiên điểm khác biệt là Vector là synchronized, hay là đồng bộ, có thể hoạt động đa luồng mà không cần gọi synchronize một cách tường minh
- **Stack**: cũng là 1 class dạng list, Stack có cách hoạt động dựa trên cơ sở của cấu trúc dữ liệu ngăn xếp (stack) với kiểu vào ra LIFO (last-in-first-out hay vào sau ra trước) nổi tiếng.

Queue Interface

Queue (hàng đợi) là kiểu dữ liệu nổi tiếng với kiểu vào ra FIFO (first-in-first-out hay vào trước ra trước), tuy nhiên với Queue Interface thì queue không chỉ còn dừng lại ở mức đơn giản như vậy mà nó cũng cấp cho bạn các phương thức để xây dựng các queue phức tạp hơn nhiều như priority queue (queue có ưu tiên), deque (queue 2 chiều), ... Và cũng giống như 2 interface trước, Queue Interface cũng kế thừa và mang đầy đủ phương thức từ Collection Interface.

Một số class về Queue thường sử dụng:

- **LinkedList**: chính là LinkedList mình đã nói ở phần List
- **PriorityQueue**: là 1 dạng queue mà trong đó các phần tử trong queue sẽ được sắp xếp.
- **ArrayDeque**: là 1 dạng deque (queue 2 chiều) được implement dựa trên mảng

TreeSet

Một số class thực thi Set Interface thường gặp:

- TreeSet
- HashSet: là 1 class implement Set Interface, mà các phần tử được lưu trữ dưới dạng bảng băm (hash table).
- EnumSet: là 1 class dạng set như 2 class ở trên, tuy nhiên khác với 2 class trên là các phần tử trong set là các enum chứ không phải object.

Map Interface

Map (đồ thị/ánh xạ) là kiểu dữ liệu cho phép ta quản lý dữ liệu theo dạng cặp key-value, trong đó key là duy nhất và tương ứng với 1 key là một giá trị value. Map Interface cung cấp cho ta các phương thức để tương tác với kiểu dữ liệu như vậy. Không giống như các interface ở trên, Map Interface không kế thừa từ Collection Interface mà đây là 1 interface độc lập với các phương thức của riêng mình.

Dưới đây là một số class về Map cần chú ý:

- **TreeMap**: là class thực thi giao diện Map Interface với dạng cây đỏ đen (Red-Black tree) trong đó các key đã được sắp xếp. Class này cho phép thời gian thêm, sửa, xóa và tìm kiếm 1 phần tử trong Map là tương đương nhau và đều là $O(\log(n))$
- **HashMap**: là class thực thi giao diện Map Interface với các key được lưu trữ dưới dạng bảng băm, cho phép tìm kiếm nhanh $O(1)$.
- **EnumMap**: cũng là 1 Map class nữa, tuy nhiên các key trong Map lại là các enum chứ không phải object như các dạng Map class ở trên.
- **WeakHashMap**: tương tự như HashMap tuy nhiên có 1 điểm khác biệt đáng chú ý là các key trong Map chỉ là các Weak reference (hay Weak key), có nghĩa là khi phần tử sẽ bị xóa khi key được giải phóng hay không còn một biến nào tham chiếu đến key nữa.