Lists:

- A list is a collection of ordered elements that are used to store list of items. Unlike array lists, these can expand and shrink dynamically

LinkedList:

- LinkedList is a sequence of nodes that have properties and a reference to the next node in the sequence. It is a linear data structure that is used to store

data. The data structure permits the addition and deletion of components

from any node next to another node. They are not stored contiguously in

memory, which makes them different arrays.

- Losing the head node in the singly linked list means losing access to your linkedlist. When I say “never lose the address of head node”, it means you have to be careful while storing it in global variables and prevent replacing it or watch out for variable scopes in local variables.

Set:

- A Set is a linear data structure that has a collection of values that are not repeated. A set can store unique values without any particular order. In the real world, sets can be used to collect all tags for blog posts and conversation participants in a chat. The data can be of Boolean, integer,float, characters, and other types. Static sets allow only query operations,

which means operations related to querying the elements. Dynamic and mutable sets allow for the insertion and deletion of elements. Algebraic operations such as union, intersection, difference, and subset can be defined on the sets. The following example shows the Set integer with a map integer key and bool as a value:

Queues:

- A queue consists of elements to be processed in a particular order or based on priority

- A queue is a linear data structure and a sequential collection

Sorting algorithms are classified by the following criteria:

+ Computational complexity

+ Memory usage

+ Stability

+ Type of sorting: serial/parallel

+ Adaptability

+ Method of sorting

bubble(noi bot): The bubble sort algorithm is a sorting algorithm that compares a pair of

neighboring elements and swaps them if they are in the wrong order. The

algorithm has a complexity of O(n ), where n is the number of elements to

be sorted. The smallest or greatest value bubbles up to the top of the

collection, or the smallest or greatest sinks to the bottom (depending on

whether you're sorting into ascending or descending order).

Sắp xếp nổi bọt là một giải thuật sắp xếp đơn giản. Giải thuật sắp xếp này được tiến hành dựa trên việc so sánh cặp phần tử liền kề nhau và tráo đổi thứ tự nếu chúng không theo thứ tự.

Giải thuật này không thích hợp sử dụng với các tập dữ liệu lớn khi mà độ phức tạp trường hợp xấu nhất và trường hợp trung bình là Ο(n2) với n là số phần tử.

Giải thuật sắp xếp nổi bọt là giải thuật chậm nhất trong số các giải thuật sắp xếp cơ bản. Giải thuật này còn chậm hơn giải thuật đổi chỗ trực tiếp mặc dù số lần so sánh bằng nhau, nhưng do đổi chỗ hai phần tử kề nhau nên số lần đổi chỗ nhiều hơn

**Selection** sort is an algorithm that divides the input collection into two

fragments. This sublist of elements is sorted by swapping the smallest or

largest element from the left of the list to the right. The algorithm is of the

order O(n ). This algorithm is inefficient for large collections, and it

performs worse than the insertion sort algorithm

-Giải thuật sắp xếp chọn (Selection Sort) là một giải thuật đơn giản. Giải thuật sắp xếp này là một giải thuật dựa trên việc so sánh **in-place**, trong đó danh sách được chia thành hai phần, phần được sắp xếp (sorted list) ở bên trái và phần chưa được sắp xếp (unsorted list) ở bên phải. Ban đầu, phần được sắp xếp là trống và phần chưa được sắp xếp là toàn bộ danh sách ban đầu.

Phần tử nhỏ nhất được lựa chọn từ mảng chưa được sắp xếp và được tráo đổi với phần bên trái nhất và phần tử đó trở thành phần tử của mảng được sắp xếp. Tiến trình này tiếp tục cho tới khi toàn bộ từng phần tử trong mảng chưa được sắp xếp đều được di chuyển sang mảng đã được sắp xếp.

Giải thuật này không phù hợp với tập dữ liệu lớn khi mà độ phức tạp trường hợp xấu nhất và trường hợp trung bình là O(n2) với n là số phần tử.

**Insertion sort** is an algorithm that creates a final sorted array one element at a time. The algorithm's performance is of the order O(n ). This algorithm is less efficient on large collections than other algorithms, such as quick, heap, and merge sort. In real life, a good example of insertion sort is the way cards are manually sorted by the players in a game of bridge

* Trường hợp tốt: O(n)
* Trung bình: O(n^2)
* Trường hợp xấu: O(n^2)

Không gian bộ nhớ sử dụng: O(1)

**The shell sort** algorithm sorts a pair of elements that are not in sequence in a collection. The distance between the elements to be compared is decreased sequentially. This algorithm performs more operations and has a greater cache miss ratio than the quick sort algorithm

Giải thuật này tránh các trường hợp phải tráo đổi vị trí của hai phần tử xa nhau trong giải thuật sắp xếp chọn (nếu như phần tử nhỏ hơn ở vị trí bên phải khá xa so với phần tử lớn hơn bên trái).

Giải thuật này sử dụng giải thuật sắp xếp chọn trên các phần tử có khoảng cách xa nhau, sau đó sắp xếp các phần tử có khoảng cách hẹp hơn. Khoảng cách này còn được gọi là **khoảng (interval)** – là số vị trí từ phần tử này tới phần tử khá

Giải thuật này khá hiệu quả với các tập dữ liệu có kích cỡ trung bình khi mà độ phức tạp trường hợp xấu nhất và trường hợp trung bình là O(n), với n là số phần tử.

**Merge sort** algorithm is a comparison-based method that was invented by John Von Neumann. Each element in the adjacent list is compared for sorting. The performance of the algorithm is in the order of O(n log n). This algorithm is the best algorithm for sorting a linked list.

Giống như Quick sort, Merge sort là một thuật toán chia để trị. Thuật toán này chia mảng cần sắp xếp thành 2 nửa. Tiếp tục lặp lại việc này ở các nửa mảng đã chia. Sau cùng gộp các nửa đó thành mảng đã sắp xếp. Hàm merge() được sử dụng để gộp hai nửa mảng. Hàm merge(arr, l, m, r) là tiến trình quan trọng nhất sẽ gộp hai nửa mảng thành 1 mảng sắp xếp, các nửa mảng là arr[l…m] và arr[m+1…r] sau khi gộp sẽ thành một mảng duy nhất đã sắp xếp.

**Quick sort** is an algorithm for sorting the elements of a collection in an organized way. Parallelized quick sort is two to three times faster than merge sort and heap sort. The algorithm's performance is of the order O(n log n). This algorithm is a space-optimized version of the binary tree sort algorithm. In the following code snippet, the quick sort algorithm is implemented.

**Searching**

**The linear** search method finds a given value within a collection by sequentially checking every element in the collection. The time complexity of the linear search algorithm is O(n). The binary search algorithm and hash tables perform better than this search algorithm

**The binary** search algorithm compares the input value to the middle element of the sorted collection. If the values are not equal, the half in which the element is not found is eliminated. The search continues on the remaining half of the collection. The time complexity of this algorithm is in the order of O(log n)

**The interpolation** search algorithm searches for the element in a sorted collection. The algorithm finds the input element at an estimated position by diminishing the search space before or after the estimated position. The time complexity of the search algorithm is of the order O(log log n)