**March 22, 2021**

1. **Stack and Queue**

* **Implementations: static array, dynamic array, or linked list**
* **Linked list implementation:** 
  + **One pointer is needed for stacks to point to the top of the stack**
  + **Two pointers are needed for queues to point to the front and rear of the queue**

1. **A class with a linked list data member**
   1. **Deep copy is required for copy constructor to create an independent linked list for different objects even though the values of the linked lists are the same**
   2. **You have to override the destructor to deallocate every node in the linked list**
   3. **If a class does not contain a linked list data member, you don’t need to use deep copy. In this case, shallow copy is sufficient.**
2. **Sorting algorithms**

**Sorting: we take an unsorted list as an input and output a sorted list.**

**Example:**

**Input list: 3, 2, 5, 4**

**Output list: 2, 3, 4, 5 (ascending order)**

**5, 4, 3, 2 (descending order)**

**Two groups of methods:**

1. **Basic approaches**

**Easy to implement but slow to execute ( O(n^2))**

**Bubble sort, insertion sort, and selection sort**

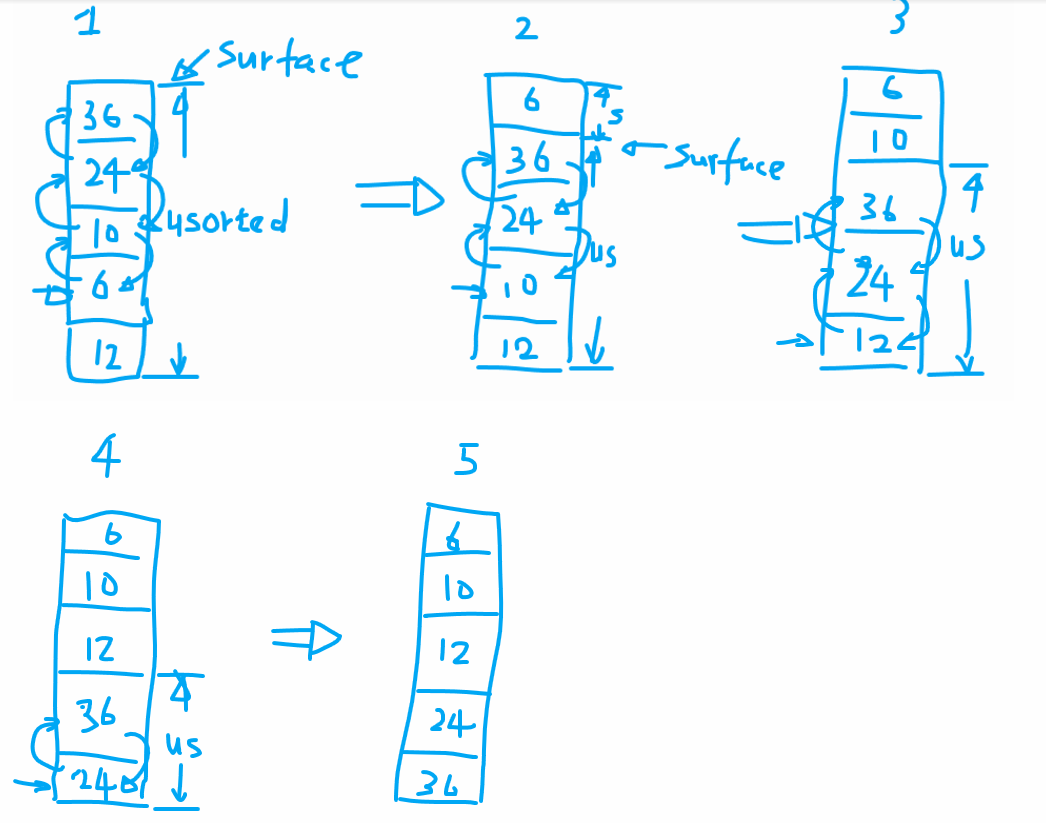
1. **Advanced approaches**

**Difficult to implement but fast to run ( O(n log n))**

**Quick sort and merge sort**

1. **Bubble sort**

**Basic idea: We observe the movement of bubbles in water and mimic this process for sorting a list. We consider the bubble as the smallest element in an unsorted list.**

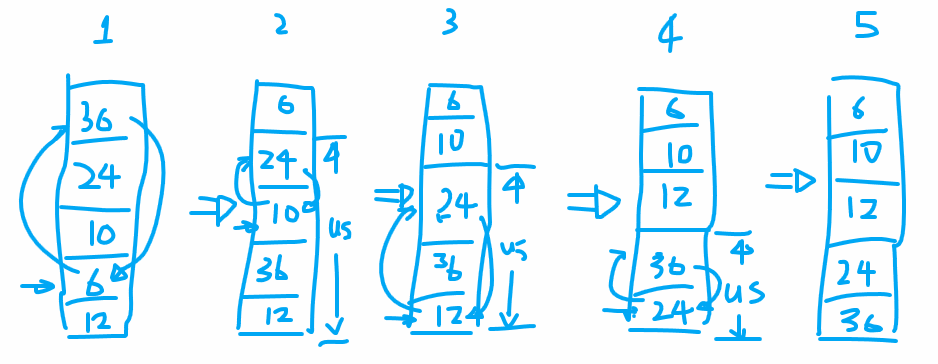
****

**Algorithm for Bubble Sorting:**

1. **Do a linear search in the unsorted part to find the element with the smallest value**
2. **Perform swaps between this smallest element with its neighboring elements to the top of the unsorted part**
3. **Repeat steps 1 & 2 until the unsorted part has one element left.**
4. **Selection Sort**

**Algorithm for Selection Sorting:**

1. **Do a linear search in the unsorted part to find the element with the smallest value**
2. **Perform swaps between this smallest element with the element at the top of the unsorted part**
3. **Repeat steps 1 & 2 until the unsorted part has one element left.**

****

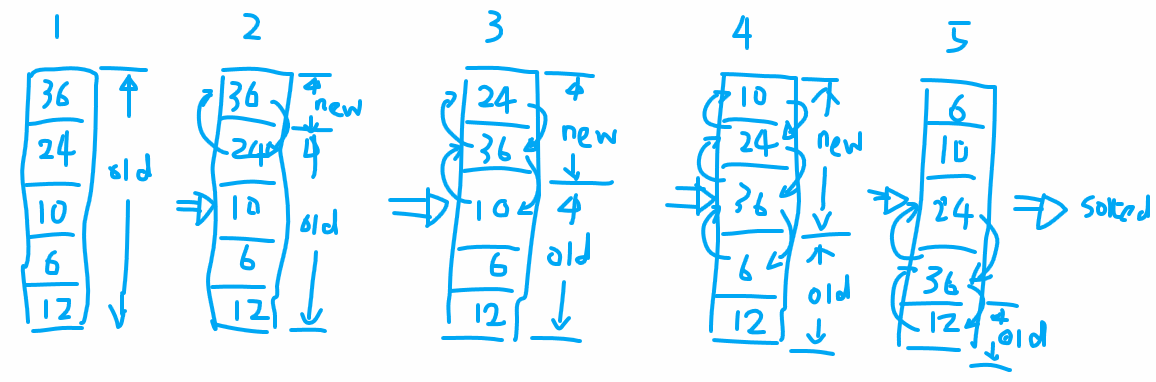
**(3) Insertion Sort**

**Basic idea: Consider the input list as a stack of card. Each time we take one card from the old stack and put it onto a new stack. Initially the new stack is empty and eventually the new stack will become full. When we put a card onto the new stack, we follow the ordering of sequence, leading to a sorted list.**

**Memory strategy:**

**Way 1: we use a second array to hold the new stack.**

**Way 2: we use the same array to hold both the new and old stacks.**

****

**Algorithm for Insertion Sort:**

1. **Pick the first element from the old stack**
2. **Insert this element into the new stack by bubble sort**
3. **Repeat steps (a) and (b) till the old stack becomes empty**

**Note that initially the new stack is empty. At this moment, the first element of the old stack don’t need to be moved and stay where it was.**