15CSE374
INTRODUCTION TO DATA STRUCTURES AND ALGORITHMS

Sarath tv

Last Lecture.

- B-Tree.
- Search
- Linear search
- Binary search.

Sorting

- Reorganizing the data in such as way that it is in the order of smallest to largest.
- Retrieved efficiently.
- Some of the algorithms are relatively easy to develop, but may perform poorly, whereas other algorithms are slightly more complex to implement, but show good performance in sorting the list when we have a long lists.
- Sorting algorithms are categorized by their memory usage, complexity, recursion, and whether they are comparison-based.
  - Algorithms use more CPU cycles
  - Chew on more memory and other computing resources
  - Expressed recursively, iteratively
  - That use comparison as the basis

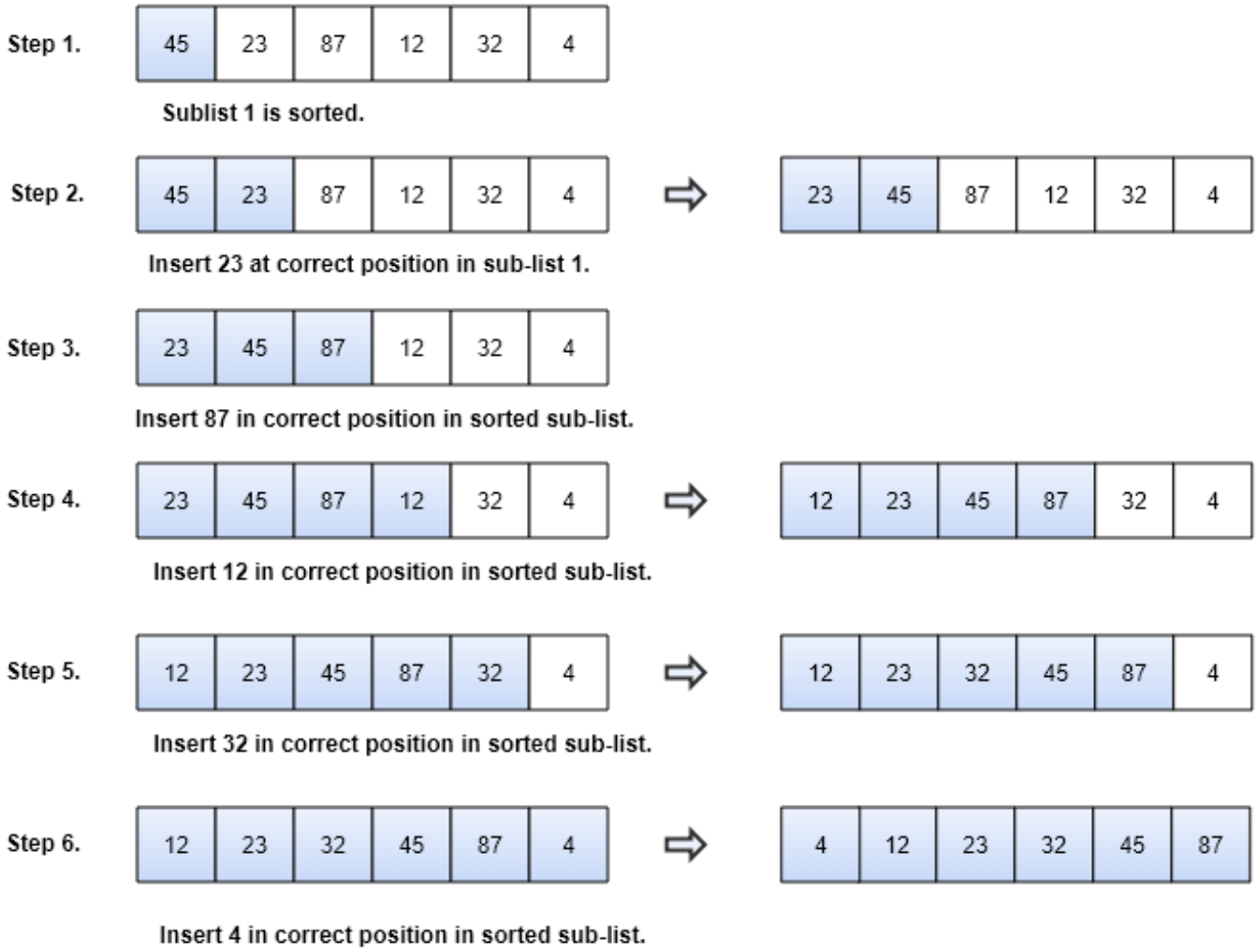
Classification of a Sorting Algorithm

- Based on Number of Swaps or Inversion This is the number of times the algorithm swaps elements to sort the input
- Based on Number of Comparisons This is the number of times the algorithm compares elements to sort the input.
- Based on Recursion or Non-Recursion Some sorting algorithms

Insertion sort

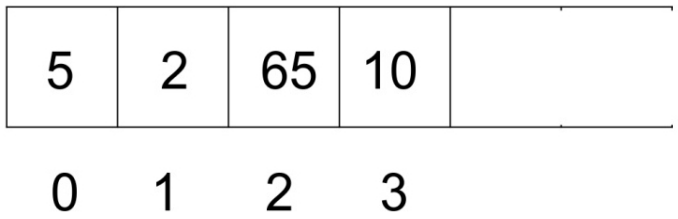
- Swapping adjacent elements to sort a list of items
- Maintains a sub-list that is always sorted, while the other portion of the list remains unsorted.
- Take elements from the unsorted sub-list and insert them in the correct position in the sorted sub-list, in such a way that this sub-list remains sorted.
- Start with one element, assuming it to be sorted, and then take another element from the unsorted sub-list and place it at the correct position (in relation to the first element) in the sorted sub-list.
- Then, we again take another element from the unsorted sub-list, and place it in the correct position (in relation to the two already sorted elements) in the sorted sub-list.
- We repeatedly follow this process to insert all the elements one by one from the unsorted sub-list into the sorted sub-list.

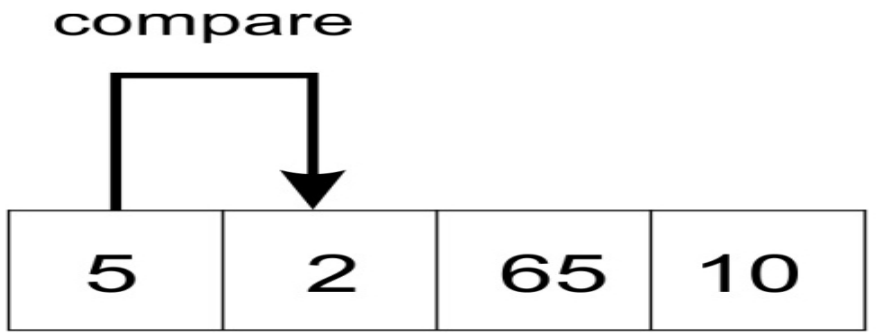
Example

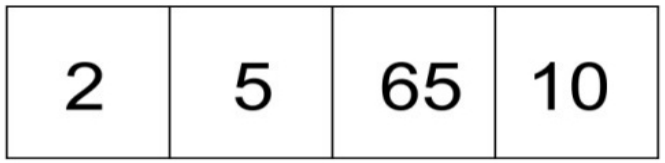

Step 1.
45 23 87 12 32 4
Sublist 1 is sorted.
Step 2.
45 23 87 12 32 4
23 45 87 12 32 4
Insert 23 at correct position in sub-list 1.
Step 3.
23 45 87 12 32 4
Insert 87 in correct position in sorted sub-list.
Step 4.
23 45 87 12 32 4
12 23 45 87 32 4
Insert 12 in correct position in sorted sub-list.
Step 5.
12 23 45 87 32 4
12 23 32 45 87 4
Insert 32 in correct position in sorted sub-list.
Step 6.
12 23 32 45 87 4
4 12 23 32 45 87
Insert 4 in correct position in sorted sub-list.

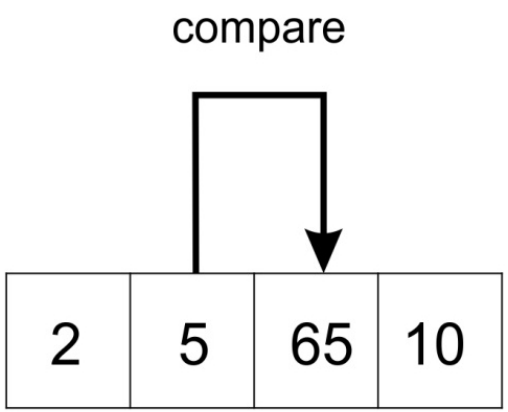
Selection sort

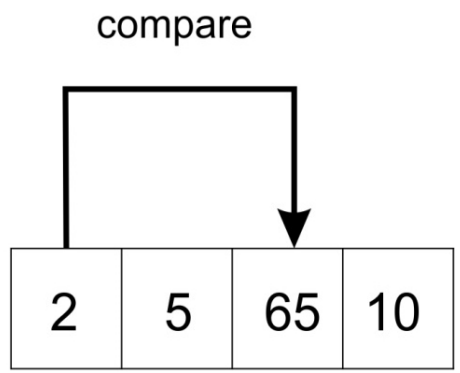
- Begins by finding the smallest element in the list, and interchanges it with the data stored at the first position in the list.
- The sub-list sorted up to the first element. Next, the second smallest element, which is the smallest element in the remaining list, is identified and interchanged with the second position in the list.
- This makes the initial two elements sorted. The process is repeated, and the smallest element remaining in the list should be swapped with the element in the third index on the list.


5
2
65
10
0
1
2
3


compare
5
2
65
10


2
5
65
10


compare
2
5
65
10


compare
2
5
65
10

Heap

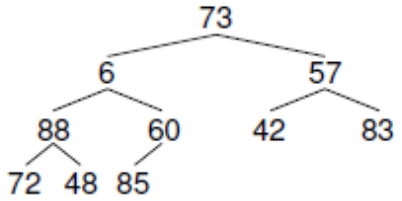
- Priority queue.
- A heap is a data structure that satisfies a **heap property.**
- Min heap **and** Max heap.

Heap Sort

- It converts the unsorted segment of the list to a Heap data structure, so that we can efficiently determine the largest element.
- We begin by transforming the list into a Max Heap - a Binary Tree where the biggest element is the root node. We then place that item to the end of the list. We then rebuild our Max Heap which now has one less value, placing the new largest value before the last item of the list.

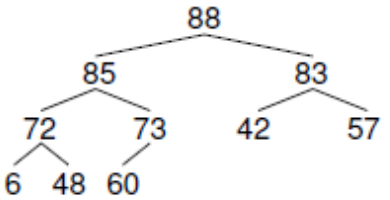
Original Numbers

73	6	57	88	60	42	83	72	48	85



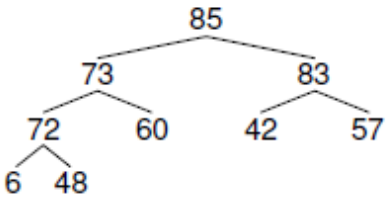
Build Heap

88	85	83	72	73	42	57	6	48	60



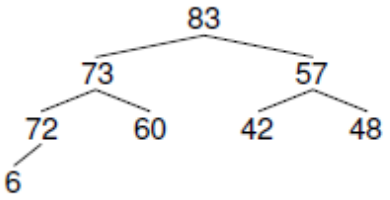
Remove 88

85	73	83	72	60	42	57	6	48	88



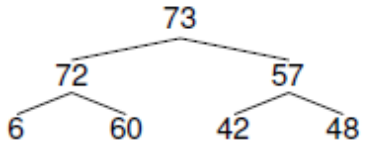
Remove 85

83	73	57	72	60	42	48	6	85	88



Remove 83

73	72	57	6	60	42	48	83	85	88



THANK YOU!!!!!