COM121β
Data Structures and Algorithms

Lecture 12:

Sorting

Nishanthi Dasanayaka
nishanthid@dcs.ruh.ac.lk

Outline

- What is sorting
- Sorting Techniques
- Sorting Algorithms
    - –Bubble Sort
    - –Selection Sort
    - –Insertion Sort
    - –Shell Sort
    - –Merge Sort
    - –Heap Sort

What is Sorting?

- Sorting is an operation that segregates items into groups according to specified criterion.

- Sorting is the process of **arranging items in order**.
  - Arranging things into ascending or descending order is also sorting.

Important Factors

- **Speed**
  - ✓ The simplest algorithms are $O(n^2)$ while more advanced ones are $O(n \log(n))$.
  - ✓ No algorithm can make less than $O(n \log n)$ comparisons between keys.
- **Storage**
  - ✓ Algorithms that sort in place are the best, needing memory $O(n)$.
  - ✓ Those are using linked list representation need extra n words of memory for references and those that work on a copy of the file needed $O(2^n)$.

Important Factors

- **Simplicity**
  - ✓ The simpler algorithms are often easier to implement and outer perform more sophisticated once for small problem.
- **Stability**
  - ✓ An algorithm is stable iff it preserve the relative order of records with equal keys.
  - ✓ There are ways to convert unstable implementation into stable ones

Sorting Techniques

1. Sorting by selection
2. Sorting by insertion
3. Sorting by exchange

Sorting types

- Bubble Sort
- Insertion Sort
- Selection Sort
- Merge Sort
- Quick Sort
- Heap Sort

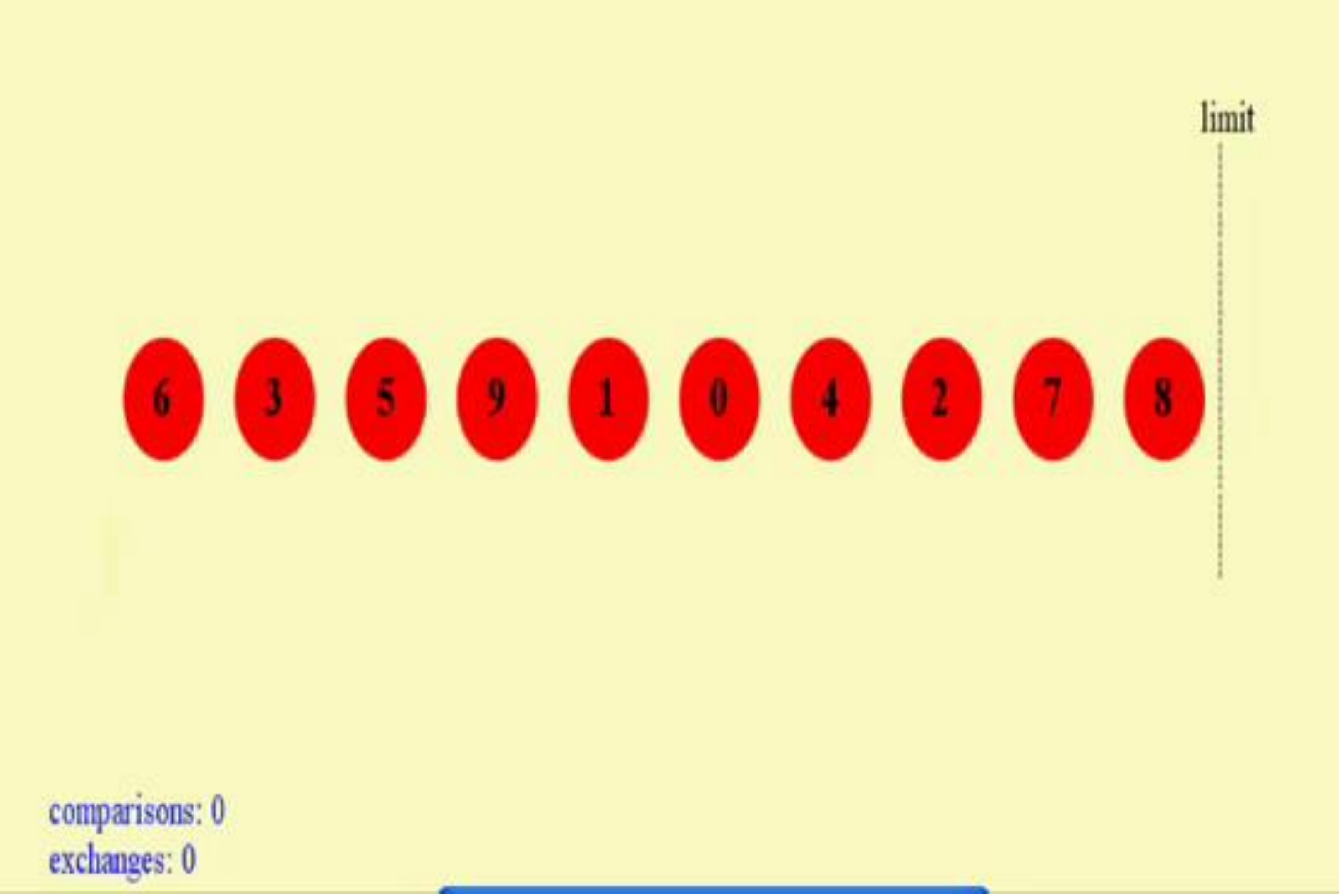
Bubble sort

- Simple sorting Algorithm
- Oldest and simplest sort in use.
- Unfortunately it is also the slowest.
- Complexity – $O(n^2)$

Bubble sort

- **Compare adjacent elements**. If the first is greater than the second, swap them.
- Do this for each pair of adjacent elements, starting with the first two and ending with the last two. At this point the last element should be the greatest.
- Repeat the steps for all elements except the last one.
- Keep repeating for one fewer element each time, until you have no more pairs to compare.

Bubble Sort



Time complexity of bubble sort

- In Bubble Sort, n-1 comparisons will be done in 1st pass, n-2 in 2nd pass, n-3 in 3rd pass and so on. So the total number of comparisons will be,

  $(n-1)+(n-2)+(n-3)+.....+3+2+1$

  Sum = $n(n-1)/2$

  i.e O(n2)

  Hence the complexity of Bubble Sort is $\mathbf{O(n^2)}$

Selection Sort

- Find the minimum value in the list
- Swap it with the value in the first position
- Repeat the steps above for the remainder of the list (starting at the second position and advancing each time)

0	1	2	3	4	5
77	42	35	12	99	5
5	42	35	12	99	77
5	12	35	42	99	77
5	12	35	42	99	77
5	12	35	42	99	77
5	12	35	42	99	77

Running time analysis

- Selection sort is O(n2) regardless of the initial order of the elements in an array.

Insertion Sort

- An insertion sort of an array partitions the array into two parts.
- One part is sorted and initially contains just the first element in the array.
- The second part contains the remaining elements.
- The sort inserts one by one the elements in the unsorted part of the array into their proper location within the sorted part of the array.

Example

Consider an array arr having 5 elements

5 4 3 1 2

Arrange the elements in ascending order

Merge Sort

- A divide-and-conquer algorithm.
- Divide the unsorted array into 2 halves until the sub-arrays only contain one element.
- Merge the sub-problem solutions together:
  - ✓ Compare the sub-array's first elements
  - ✓ Remove the smallest element and put it into the result array
  - ✓ Continue the process until all elements have been put into the result array

Example

Consider an array arr having 6 elements

5 4 3 1 2 6

Arrange the elements in ascending order using merge sort algorithm

Running time analysis

- The height h of the merge-sort is O(log n)
- At each recursive call it divides half the sequence
- The overall amount or work done at the nodes of depth i is

  /space complexity O(n)
- It partitions and merges 2i sequences of size n/2i
- There are 2i+1 recursive calls
- Thus, the worst case total running time of merge-sort is

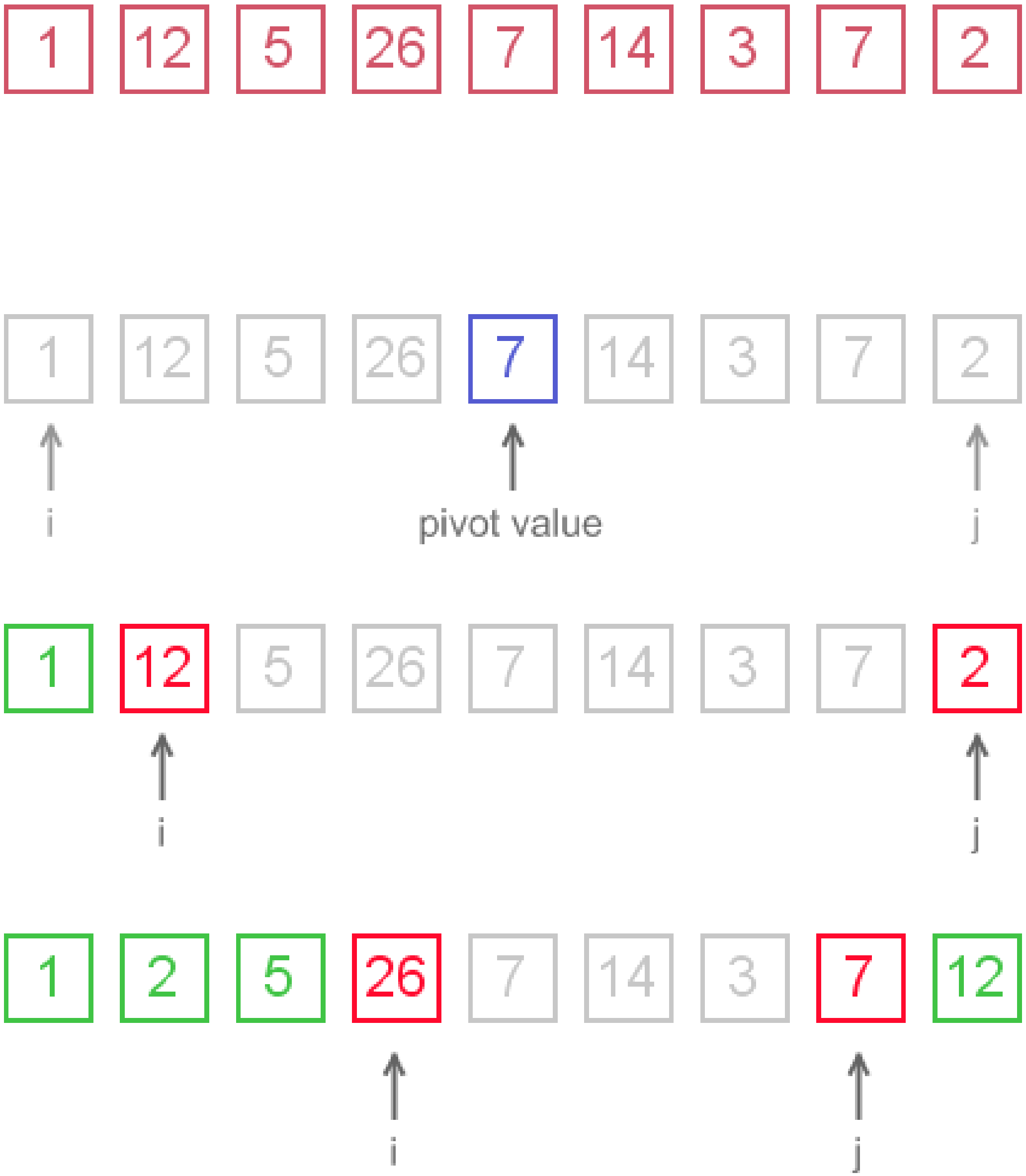
O(n log n)

Quick Sort

- Like Merge Sort, QuickSort is a Divide and Conquer algorithm.
- Quicksort is undoubtedly the most popular sorting algorithm, and for good reason:
- In the majority of situations, it’s the fastest, operating in O(N*logN) time

Quick Sort

- It picks an element as pivot and partitions the given array around the picked pivot.
- There are many different versions of quickSort that pick pivot in different ways.
    - Always pick first element as pivot.
    - Always pick last element as pivot
    - Pick a random element as pivot.
    - Pick median as pivot.

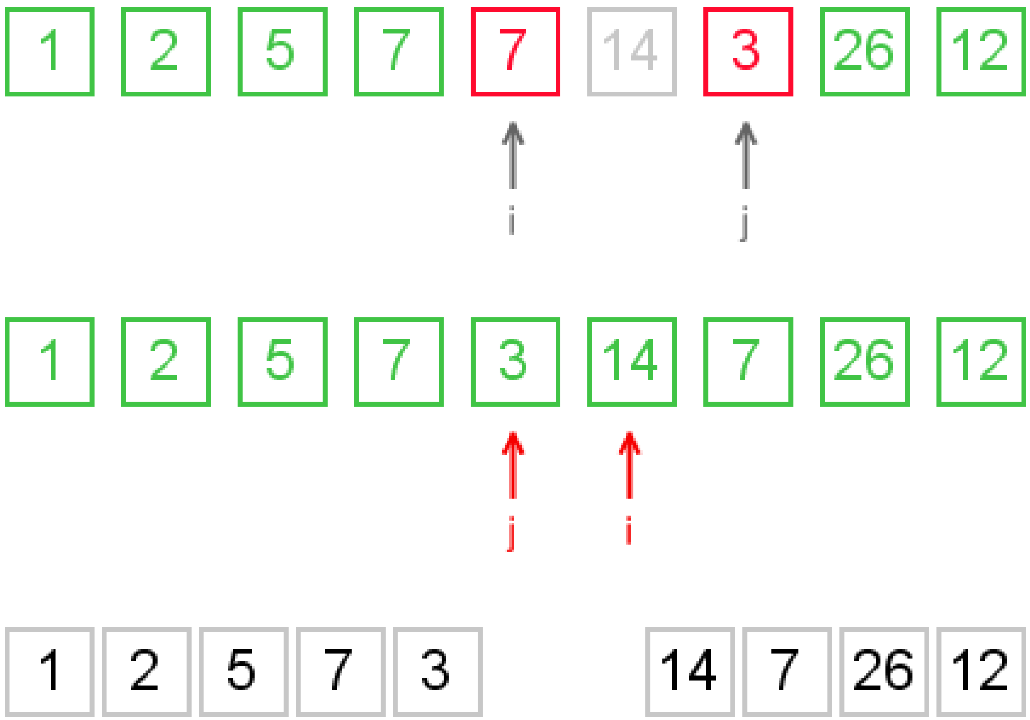


unsorted

pivot value = 7

12 ≥ 7 ≥ 2, swap 12 and 2

26 ≥ 7 ≥ 7, swap 26 and 7



$7 \geq 7 \geq 3$, swap 7 and 3

$i > j$, stop partition

run quick sort recursively

1	2	3	5	7	7	12	14	26	sorted

Heap Sort

- In simple,
- The heap sort works as it name suggests
- It begins by building a heap out of the data set, and then
  - Removing the largest item and placing it at the end of the sorted array.
- After that removing the largest item : it reconstructs the heap and removes the largest remaining item and places it in the next open position from the end of the sorted array.
- This is repeated until there are no items left in the heap and the sorted array is full.

Heap Sort

- Heap sort is not stable but,
- The heap sort does not require any extra storage beside the element themselves.
- It is guaranteed to take no longer than O(n log n) time, no matter what the input is.
- In general Heap sort is slower than quick sort.