Algorithms Design and Analysis

Presented by Md
Sahin Alom
Id : 18CSE051

Presented to Md
Faruk Hossen
lecturer dept of CSE

Outlines

- Longest common Subsequence
- Quick sort
- Heap sort
- Merge sort
- Counting and Radix sort

Longest Common Subsequence

What is subsequence

- A subsequence of a string is **a new string that is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters**. (i.e., "ace" is a subsequence of "abcde" while "aec" is not).

Longest Common Subsequence

- A **subsequence** is a sequence that can be derived from another sequence by deleting some or no elements without changing the order of the remaining elements.
- A **longest subsequence** of string s1 and s2 is the length of the longest subsequence which is common in both the strings.

LCS Theorem

- The *LCS* can be found by dynamic programming formulation. One can easily show:
  - **Theorem:** With a score of 1 for each match and a zero for each mismatch or space , the matched characters in an alignment of maximum value for a *LCS*.
- Since it is using the general dynamic programming algorithm its complexity is *O(nm)* .
- A longest substring problem, on the other hand has a *O(n+m)* solution. Subsequences are much more complex than substrings.

Example

- Given two strings as input:

  s1="abdca"

  s2="bdca"

- Output: 3
- This shows the longest possible subsequence of s1 which is "bda." This can be derived from sequence s1 by deleting the element 'a' without changing the order of the remaining elements, hence, the length of string is 3. One brute force solution is to try all possible subsequences of 's1' and 's2' to find the longest one.

DYNAMIC METHOD

- If we draw the complete recursion tree, then we can see that there are many subproblems which are solved again and again. So this problem has Overlapping Substructure property and recomputation of same subproblems can be avoided by either using Memoization or Tabulation.

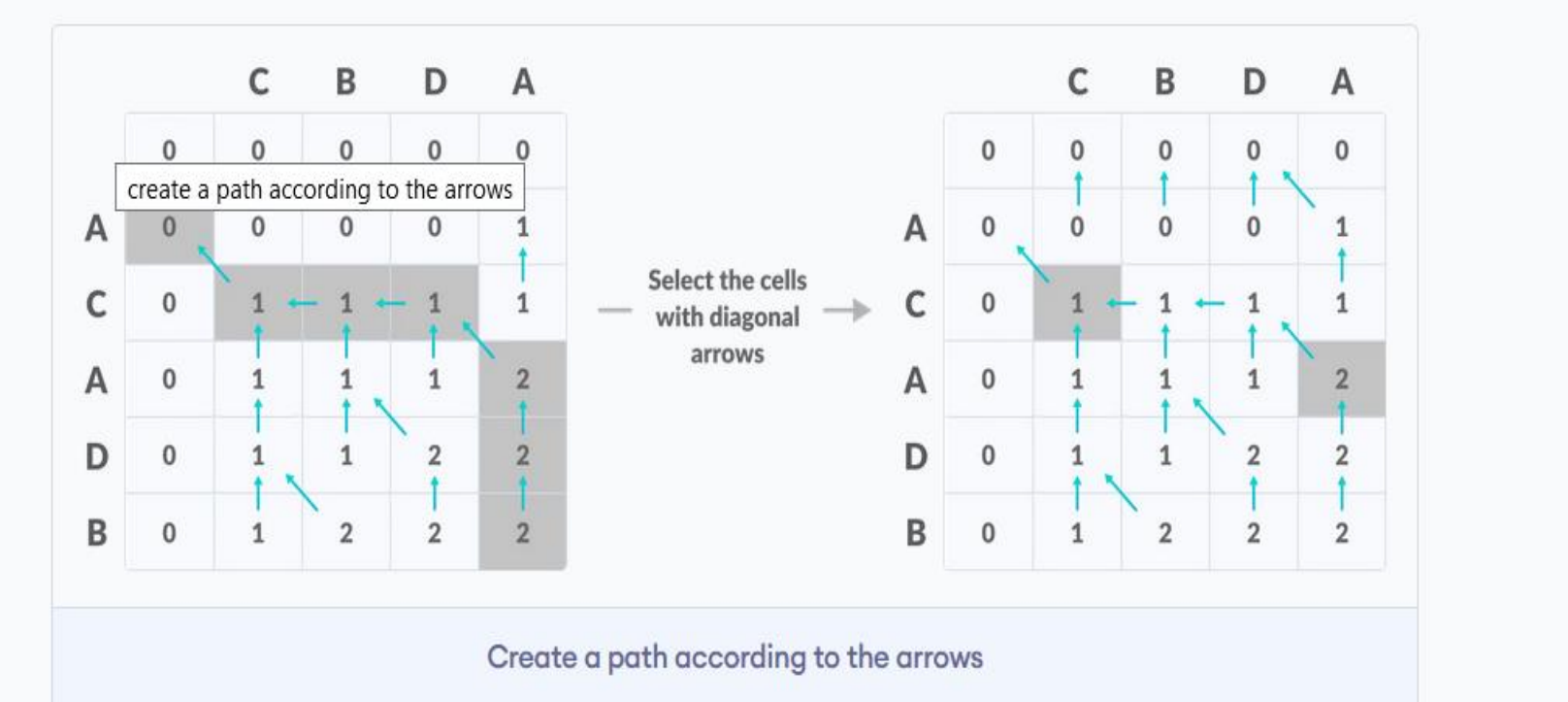
DYNAMIC METHOD

The following steps are followed for finding the longest common subsequence.

1.Create a table of dimension n+1*m+1 where n and m are the lengths of *X* and *Y* respectively.
2.The first row and the first column are filled with zeros.

Initialise a table

DYNAMIC METHOD



Quick sort

Introduction

- This sorting algorithm uses the idea of divide and conquer.
- It finds the element called pivot which divides the array into two halves in such a way that elements in the left half are smaller than pivot and elements in the right half are greater than pivot.

Quick sort

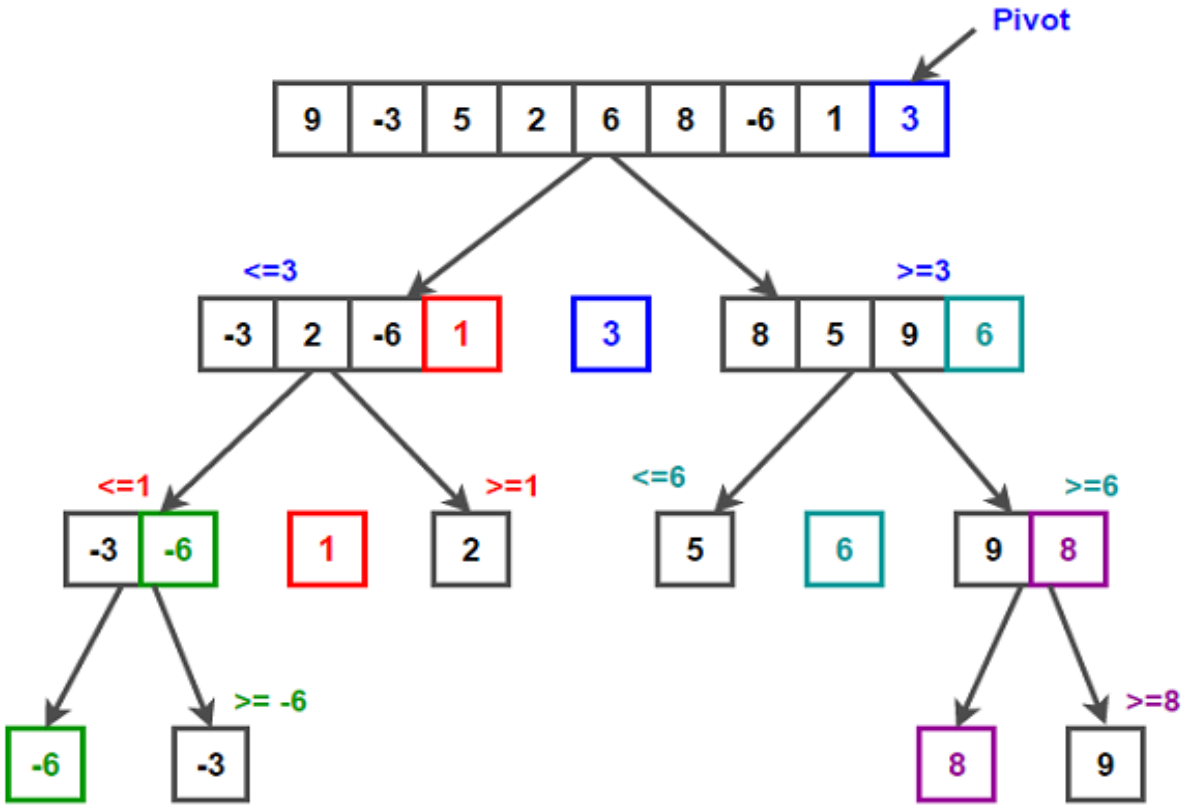
- Three steps

1. Find pivot that divides the array into two halves.
2. Quick sort the left half.
3. Quick sort the right half.

Algorithm

```
PARTITION(A, p, r)
1   x = A[r]
2   i = p – 1
3   for j = p to r - 1
4       if A[j] <= x
5           i = i + 1
6           exchange A[i] with A[j]
7   exchange A[I + 1] with A[r]
8   return i + 1;
```

Example of quick sort



Quick sort time complexity

- The best-case time complexity of Quicksort is: $O(n \log n)$
- The best-case time complexity of Quicksort is also: $O(n \log n)$
- The worst-case time complexity of Quicksort is: $O(n^2)$

Advantages

- It is in-place since it uses only a small auxiliary stack.
- It requires only n (log n) time to sort n items.
- It has an extremely short inner loop.
- This algorithm has been subjected to a thorough mathematical analysis, a very precise statement can be made about performance issues.

Disadvantages

- The primary disadvantage with quicksort is that **its absolute worst case is O(n ^2)**. If we choose the pivot points randomly, we can guarantee that this behavior is unlikely, but we cannot provide an absolute guarantee.

Heap sort

Introduction

- Binary tree in which each node is either a leaf or has degree exactly 2.
- Binary tree in which all leaves are on the same level and all internal nodes have degree 2

What is heap sort?

- A sorting algorithm that works by first organizing the data to be sorted into a special type of binary tree called a heap
- The binary heap data structures is an array that can be viewed as a complete binary tree. Each node of the binary tree corresponds to an element of the array. The array is completely filled on all levels except possibly lowest.

Types of Heap

- Max Heap
  - Store data in ascending order
  - Has property of

    $A[Parent(i)] \geq A[i]$
- Min Heap
  - Store data in descending order
  - Has property of

    $A[Parent(i)] \leq A[i]$

Algorithm

- Since the tree satisfies Max-Heap property, then the largest item is stored at the root node.
- **Swap:** Remove the root element and put at the end of the array (nth position) Put the last item of the tree (heap) at the vacant place.
- **Remove:** Reduce the size of the heap by 1.
- **Heapify:** Heapify the root element again so that we have the highest element at root.
- The process is repeated until all the items of the list are sorted.

Time Analysis

- **Worst complexity:** n*log(n)
- **Average complexity:** n*log(n)
- **Best complexity:** n*log(n)

Merge sort

introduction

- Merge Sort is a complex and fast sorting algorithm that repeatedly divides an unsorted section into two equal subsections, sorts them separately and merges them correctly.

Definition

- Merge sort is a **DIVIDE AND CONQUER** algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. The merge() function is used for merging two halves.

Steps involved:

- Divide the problem into sub-problems that are similar to the original but smaller in size.
- Conquer the sub-problems by solving them recursively. If they are small enough, just solve them in a straightforward manner.
- Combine the solutions to create a solution to the original problem.

Algorithm:

- mergeSort(arr[], l, r)
- If l < r
- 1. Find the middle point to divide the array into two halves: middle m = (l+r)/2
- 2. Call mergeSort for first half: Call mergeSort(arr, l, m)
- 3. Call mergeSort for second half: Call mergeSort(arr, m+1, r)
- 4. Merge the two halves sorted in step 2 and 3: Call merge(arr, l, m, r)

Time Analysis

- **Worst complexity:** n*log(n)
- **Average complexity:** n*log(n)
- **Best complexity:** n*log(n)

Why Merge Sort??

- Compared to insertion sort merge sort is faster.
- On small inputs, insertion sort may be faster. But for large enough inputs, merge sort will always be faster, because its running time grows more slowly than insertion sorts.

Applications

- Merge sort type algorithms allows large data sets to be sorted easily.
- Merge sort accesses data sequentially and the need of random access is low.
- Inversion count problem.
- Used in External Sorting.

Counting and Radix sort

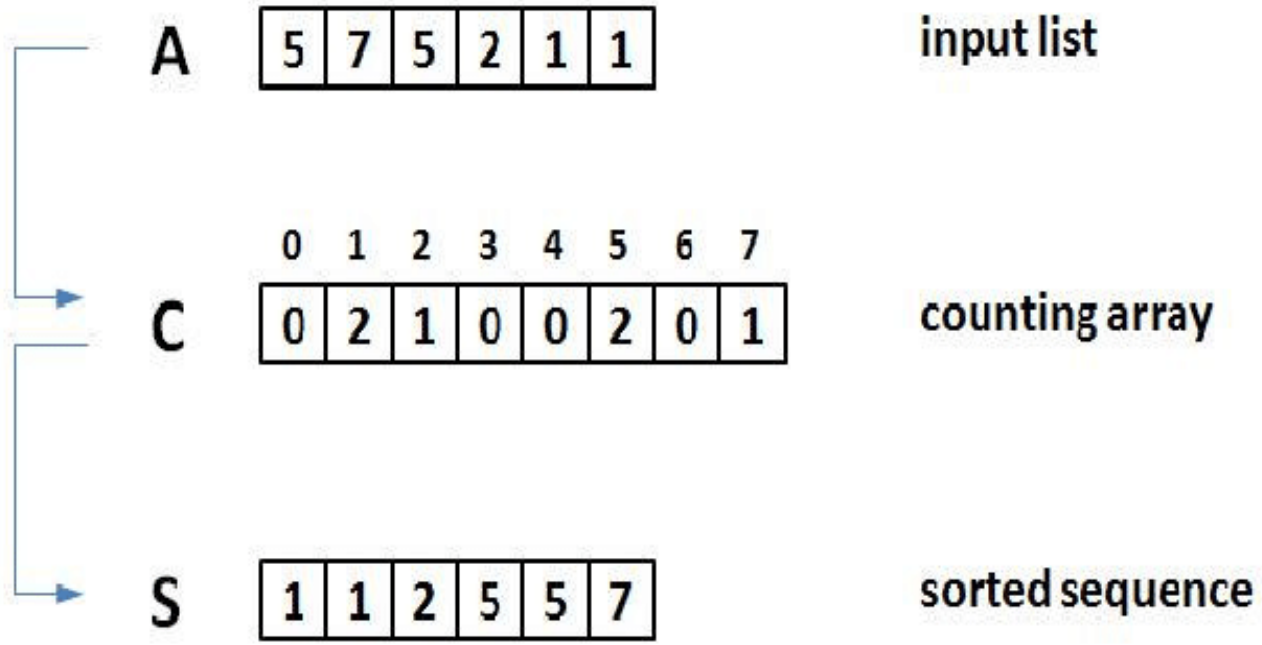
Introduction

- Counting sort works by iterating through the input, counting the number of times each item occurs, and using those counts to compute an item's index in the final, sorted array.

What is counting sort?

- Counting sort is a sorting technique based on keys between a specific range. It works by **counting the number of objects having distinct key values** (kind of hashing). Then doing some arithmetic to calculate the position of each object in the output sequence.

Example of counting sort



Counting sort properties

Worst case time	O(n)$O(n)$
Best case time	O(n)$O(n)$
Average case time	O(n)$O(n)$
Space	O(n)$O(n)$

Points to be noted:

- Counting sort is efficient if the range of input data is not significantly greater than the number of objects to be sorted. Consider the situation where the input sequence is between range 1 to 10K and the data is 10, 5, 10K, 5K.
- It is not a comparison based sorting. It running time complexity is O(n) with space proportional to the range of data.
- It is often used as a sub-routine to another sorting algorithm like radix sort.
- Counting sort uses a partial hashing to count the occurrence of the data object in O(1).
- Counting sort can be extended to work for negative inputs also.

Advantages and Disadvantages

- **Linear time.**

  Counting sort runs in O(n)$O(n)$ time, making it asymptotically faster than comparison-based sorting algorithms like quicksort or merge sort.

- **Restricted inputs**. Counting sort only works when the range of potential items in the input is known ahead of time.
- **Space cost**. If the range of potential values is big, then counting sort requires a lot of space (perhaps more than O(n)$O(n)$).

What is Radix sort?

- Radix sort is **one of the sorting algorithms used to sort a list of integer numbers in order**. In radix sort algorithm, a list of integer numbers will be sorted based on the digits of individual numbers. Sorting is performed from least significant digit to the most significant digit.

Algorithm

- **Step 1 -** Define 10 queues each representing a bucket for each digit from 0 to 9.
- **Step 2 -** Consider the least significant digit of each number in the list which is to be sorted.
- **Step 3 -** Insert each number into their respective queue based on the least significant digit.
- **Step 4 -** Group all the numbers from queue 0 to queue 9 in the order they have inserted into their respective queues.
- **Step 5 -** Repeat from step 3 based on the next least significant digit.
- **Step 6 -** Repeat from step 2 until all the numbers are grouped based on the most significant digit.

Example of radix sort

Consider this input array

170	45	75	90	802	24	2	66
170	90	802	2	24	45	75	66
802	2	24	45	66	170	75	90
2	24	45	66	75	90	170	802

Complexity of the Radix Sort Algorithm

- To sort an unsorted list with '**n**' number of elements, Radix sort algorithm needs the following complexities.
- **Worst Case : O(n)**
  Best Case : O(n)
  Average Case : O(n)

Advantages

- Fast when the keys are short i.e. when the range of the array elements is less.
- Used in suffix array constuction algorithms like Manber's algorithm and DC3 algorithm.
- **Radix Sort** is stable **sort** as relative order of elements with equal values is maintained.

Disadvantages

- Since Radix Sort depends on digits or letters, Radix Sort is much less flexible than other sorts. ...
- The constant for Radix sort is greater compared to other sorting algorithms.
- It takes more space compared to Quicksort which is inplace sorting.