Sorting, Searching and Merging

Akash Hegde
Seventh Sense Talent Solutions
Vivekananda Institute of Technology
25 March 2021

Introduction to Searching

- Searching – check for an element or retrieve an element from any data structure where it is stored.
- Classification of search algorithms based on type of search operation:
  - Sequential Search – the list or array is traversed sequentially and every element is checked.
    Example: Linear Search
  - Interval Search – the list or array is split into two search spaces and repeatedly checked.
    Example: Binary Search
- Interval search - more efficient than sequential search for sorted data structures.

Linear Search

- Problem: Given an array *arr[]* of *n* elements, write a function to search a given element *x* in *arr[]*.
- Approach:
  - Start from the leftmost element of *arr[]* and one by one compare *x* with each element of *arr[]*
  - If *x* matches with an element, return the index.
  - If *x* doesn't match with any of elements, return -1.
- Example: search for element 10 in the array 2, 4, 6, 8, 10, 12, 14.
- Time Complexity: *O(n)*

Binary Search

- Problem: Given an array *arr[]* of *n* elements, write a function to search a given element *x* in *arr[]*.
- Approach:
  - Compare *x* with the middle element.
  - If *x* matches with middle element, we return the mid index.
  - Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.
  - Else (x is smaller) recur for the left half.
- Example: search for element 10 in the array 2, 4, 6, 8, 10, 12, 14.
- Time Complexity: $O(\log_2 n)$

Complexity Analysis of Binary Search

- Time Complexity: $O(\log_2 n)$
- Example: search for element 10 in the array 2, 4, 6, 8, 10, 12, 14.
- Iteration 1: middle element = 8, therefore search towards right sub-array
- Iteration 2: middle element = 12, therefore search towards left sub-array
- Iteration 3: middle element = 10, search successful
- Let say the iteration in Binary Search terminates after **k** iterations. In the above example, it terminates after 3 iterations, so here, **k = 3**

Complexity Analysis of Binary Search

- At each iteration, the array is divided by half. So let's say the length of array at any iteration is **n**
- Iteration 1: Length of array = n
- Iteration 2: Length of array = $n/2$
- Iteration 3: Length of array = $(n/2)/2 = (n/2^2)$
- Therefore, after Iteration k: Length of array = $(n/2^k)$
- Also, after k iterations, length of array becomes 1.
- Thus, $(n/2^k) = 1 \quad \Rightarrow n = 2^k$
- Applying $\log_2$ on both sides, $\log_2(n) = \log_2(2^k)$
  $\Rightarrow \log_2(n) = k \log_2(2) \quad \Rightarrow \mathbf{k = \log_2(n)}$

Linear Search vs Binary Search

- Input data needs to be sorted in Binary Search and not in Linear Search.
- Linear search does the sequential access whereas Binary search access data randomly.
- Time complexity of linear search is $O(n)$, Binary search has time complexity $O(\log_2 n)$.
- Linear search performs equality comparisons and Binary search performs ordering comparisons.

Introduction to Sorting

- Sorting – used to rearrange a given array or list elements according to a comparison operator on the elements.
- The comparison operator is used to decide the new order of element in the respective data structure.
- Example:
  Input -> **v i v e k a n a n d a**
  ...sorting according to their ASCII values...
  Output -> **a a a d e i k n n v v**

Sorting Algorithms

- Bubble Sort
- Selection Sort
- Insertion Sort
- Quick Sort
- Merge Sort
- Heap Sort
- Radix Sort
- Bucket Sort
- Shell Sort
- Tree Sort

Sorting Terminology

- In-place sorting / Internal sorting:
  Uses constant extra space for producing the output (modifies the given array only).
  Sorts the list only by modifying the order of the elements within the list.
  Example: Insertion sort, Selection sort
- Out-of-place sorting / External sorting:
  When all data that needs to be sorted cannot be placed in-memory at a time, the sorting is called external sorting.
  External Sorting is used for massive amount of data.
  Example: Merge sort

Bubble Sort

- Works by repeatedly swapping the adjacent elements if they are in wrong order.
- Example: Array - 10 2 4 8 6
- Pass 1:
  10 2 4 8 6 -> 2 10 4 8 6
  2 **10 4** 8 6 -> 2 4 10 8 6
  2 4 **10 8** 6 -> 2 4 8 10 6
  2 4 8 **10 6** -> 2 4 8 6 10
- Pass 2:
  2 4 8 6 10 -> 2 4 8 6 10
  2 **4 8** 6 10 -> 2 4 8 6 10
  2 4 **8 6** 10 -> 2 4 6 8 10

Bubble Sort

- Pass 3:
  2 4 6 8 10 -> 2 4 6 8 10
  2 **4 6** 8 10 -> 2 4 6 8 10
- Pass 4:
  2 4 6 8 10 -> 2 4 6 8 10
- Time complexity: $O(n^2)$

Selection Sort

- Sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning.
- The algorithm maintains two subarrays in a given array.
  i) The subarray which is already sorted.
  ii) Remaining subarray which is unsorted.
- In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.

Selection Sort

- Example: Array – 10 2 4 8 6
- Pass 1: min element = 2
  10 **2** 4 8 6 -> 2 10 4 8 6
- Pass 2: min element = 4
  2 10 **4** 8 6 -> 2 4 10 8 6
- Pass 3: min element = 6
  2 4 10 8 **6** -> 2 4 6 8 10
- Pass 4: min element = 8
  2 4 6 **8** 10 -> 2 4 6 8 10

Insertion Sort

- The array is virtually split into a sorted and an unsorted part.
- Values from the unsorted part are picked and placed at the correct position in the sorted part.
- Approach:
  - Iterate from arr[1] to arr[n] over the array.
  - Compare the current element (key) to its predecessor.
  - If the key element is smaller than its predecessor, compare it to the elements before.
    Move the greater elements one position up to make space for the swapped element.

Insertion Sort

- Example: Array – 10 2 4 8 6
- Pass 1:
  10 2 4 8 6 -> 2 10 4 8 6
- Pass 2:
  2 **10 4** 8 6 -> 2 4 10 8 6
- Pass 3:
  2 4 **10 8** 6 -> 2 4 8 10 6
- Pass 4:
  2 4 8 **10 6** -> 2 4 **8 6** 10 -> 2 4 6 8 10

Quick Sort

- Divide and Conquer algorithm.
- Picks an element as pivot and partitions the given array around the picked pivot.
- Target of partitions – given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x.
- All this should be done in linear time.

Quick Sort

- Example: Array – 100 20 40 80 60
- low = 0, high = 4, pivot = arr[high] = 60
- Initialize index of smaller element, i = -1
- Traverse from j = low to j = high-1
- j = 0, i = -1: arr[j] > pivot, therefore do nothing
- j = 1, i = -1: arr[j] <= pivot, therefore do i++ (i=0) and swap arr[i] and arr[j]
  100 20 40 80 60 -> 20 100 40 80 60
- j = 2, i = 0: arr[j] <= pivot, therefore do i++ (i=1) and swap arr[i] and arr[j]
  20 **100 40** 80 60 -> 20 40 100 80 60

Quick Sort

- j = 3, i = 1: arr[j] > pivot, therefore do nothing
- Come out of loop because j = high-1
- Now swap arr[i+1] with pivot (new i = 2)
  20 40 100 80 60 -> 20 40 60 80 100
- Pivot at its correct place.
  All elements < pivot are towards its left.
  All elements > pivot are towards its right.

Merging and Merge Sort

- Divide and Conquer algorithm.
- Divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves.
- **merge() function** is used for merging two halves. The merge(arr, l, m, r) is a key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one.

Merging and Merge Sort

- Example: Array – 10 2 4 8 6
- Middle element = 4
- Left half subarray: 10 2 4
- Right half subarray: 8 6
- Left half sorting: 2 4 10
- Right half sorting: 6 8
- Merged array: 2 4 6 8 10

Time Complexities

- Number of times a particular instruction set is executed rather than the total time is taken.

Algorithm	Best Case	Average Case	Worst Case
Bubble Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Selection Sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$
Insertion Sort	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Quick Sort	$\Omega(nlogn)$	$\Theta(nlogn)$	$O(n^2)$
Merge Sort	$\Omega(nlogn)$	$\Theta(nlogn)$	$O(nlogn)$

Thank you!