Searching & Sorting Algorithms

Searching

1. The Sequential Search

   Analysis of Sequential Search

2. The Binary Search

   Analysis of Binary Search

Sequential Search/ Linear Search

Sequential checking of elt from list / collection for desired value

If found stop

Else check till end then stop

Complexity -

Worst case - $O(n)$

Best Case - $\Omega(1)$

Avg Case - $\theta(n)$

Binary Search

Binary Search is a searching algorithm for finding an element's position in a sorted array.

element is always searched in the middle of a portion of an array.

implemented in two ways

- Iterative Method
- Recursive Method

The recursive method follows the divide and conquer approach.

Complexity -

Worst case - $O(\log n)$

Best Case - Ω (1)

Avg Case - θ (log n)

Iteration Method

```
do until the low and high meet each other.
    mid = (low + high)/2
    if (x == arr[mid])
        return mid
    else if (x > arr[mid]) // x is on the right side
        low = mid + 1
    else                   // x is on the left side
        high = mid - 1
```

Recursive Method

```
binarySearch(arr, x, low, high)
    if low > high
        return False
    else
        mid = (low + high) / 2
        if x == arr[mid]
            return mid
```

```
else if x > arr[mid]        // x is on the right side
    return binarySearch(arr, x, mid + 1, high)
else                        // x is on the left side
    return binarySearch(arr, x, low, mid - 1)
```

Introduction to sorting

- Selection sort
- Insertion sort
- Bubble sort
- Heapsort
- Mergesort
- Quicksort

Bubble Sort

compares two adjacent elements and swaps them until they are in the intended order.

Just like the movement of air bubbles in the water.

each element of the array move to the end in each iteration. Therefore, it is called a bubble sort.

Complexity -

Worst case - O(n^2)

Best Case - Ω (n^2)

Avg Case - θ (n^2)

array size =5 (n)

Iterate array for every elt of array

For every elt check all consecutive elt till size -i-2

(n=5 j will run from 0 - 5-0-2=3 (0 -3 ⇒4 times) ⇒ j<size-i-1)

for(i= 0 to size-1(4))

for(j = 0 to size-i-1(3/<4))

if(a[j] > a[j+1])

swap

Insertion sort

Places an unsorted element at its suitable place in each iteration.

Ex. - sorting cards in our hand in a card game.

assume that the first card is already sorted then, we select next card.

If the unsorted card is greater than the card in hand, it is placed on the right otherwise, to the left and so on for other cards

Checking done in reverse order of sorted card from last to first

Complexity -

Worst case - O(n^2)

Best Case - Ω (n^2) Avg Case - θ (n^2)

```
 mark first element as sorted
 for each unsorted element X
   'extract' the element X
   for j → lastSortedIndex down to 0
     if current element j > X
       move sorted element to  the right by 1 with j decreament
   break loop and insert X here at j+1 index
end insertionSort
```

Selection sort

selects the smallest element from an unsorted list in each iteration and places that element at the beginning of the unsorted list.Set the first element as minimum.

Selection Sort Steps

Select first element as minimum

Compare minimum with the all remaining elements.

If any smaller than minimum found, assign that element as minimum.

After each iteration, minimum is placed in the front of the unsorted list.

Continue above process for remaining elements(list) by skipping sorted elements till last position.

Complexity -

Worst case - $O(n^2)$

Best Case - $\Omega(n^2)$

Avg Case - $\theta(n^2)$

Mergesort

Follows Divide and Conquer approach

Problem is divided into multiple sub-problems.

Each sub-problem is solved individually.

Finally, sub-problems are combined to form the final solution.

Complexity -

Worst case - O(nlogn)

Best Case - Ω (nlogn)

Avg Case - θ (nlogn)

Quicksort

Follows divide and conquer approach

- An array is divided into subarrays by selecting a pivot element (element selected from the array).

  While dividing the array, the pivot element should be positioned in such a way that elements less than pivot are kept on the left side and elements greater than pivot are on the right side of the pivot.

- The left and right subarrays are also divided using the same approach.
- This process continues until each subarray contains a single element.
- At this point, elements are already sorted. Finally, elements are combined to form a sorted array
- Complexity -

Worst case - $O(n^2)$

Best Case - Ω (nlogn)

Avg Case - θ (nlogn)

Heapsort

Efficient sorting algorithm

Prerequisite of two types of data structures - arrays and trees.

The initial set of numbers that we want to sort is stored in an array

e.g. `[11, 8, 56, 42, 13, 30]` and after sorting, we get a

sorted array `[8,11,13,30,42,56]`.

Heap sort works by visualizing the elements of the array as a

special kind of complete binary tree called a heap.

Example

8, 3, 10, 6, 4, 14, 7, 1, 13

Complexity -

Worst case - O(nlogn)

Best Case - Ω (nlogn)

Avg Case - θ (nlogn)