

TUTORIAL-3

Q.1) Write linear search pseudocode to search an element in a sorted array with minimum comparisons.

A.1) No. of comparisons can be reduced by copying the element to be searched (suppose x) to last iteration so that one last comparison when x is not present in $arr[1]$ - saved

pseudocode :-

```
search( arr, n, x )
```

```
if arr[n-1] == x // 1 comparison  
    return "true"
```

```
backup = arr[n-1];
```

```
arr[n-1] = x;
```

```
for i=0; i < n // No termination condition
```

```
if arr[i] == x; // execute at most n times
```

```
i.e. at most n comparisons
```

```
arr[n-1] = backup
```

```
return (i < n-1) // 1 comparison
```

```
return found;
```

```
else
```

```
return not found;
```

Q.2) Write Pseudocode for Iterative & Recursive Insertion sort why Insert sort is also called Online sort.

Iterative Pseudocode:

// Sort an arr[] of size n
procedure sort (arr, n)
 loop from i = 1 to n - 1
 a → pick element arr[i] & insert
 it into sorted sequence arr [0 . . . i - 1]

Recursive Pseudocode

Base case - if ($n \leq 1$)

 return; // If arr 1 or smaller, return.

sort first($n-1$) elements
i.e (arr, $n-1$).

last = arr [$n-1$]; // Insert last elem at correct
j = $n-2$; position

Apply Insert algo.

print utility (f) of arr of size n.

- Insertion sort is called online sort because it does not need to know about what values it will sort and information is required while algo runs.
- whereas selection is offlinesort.

Q. 3) Complexity of all sorting Algorithms :-

Selection sort - $O(n^2)$

Bubble sort - $O(n)$

Inception $\Theta(n)$

Heap $\Theta(n \log n)$

Quick $\Theta(n \log n)$

Merge $\Theta(n \log n)$

Count $\Theta(n+k)$

Radix $\Theta(nR)$

Q.4) Divide all sorting algorithm into Inplace / Stable / Online.

A.4) Inplace :- Bubble sort, Selection sort, Insertion & Heap sort

Stable :- Merge sort, Count sort, Insertion sort & Bubble sort

Unstable :- Quicksort, Heap & Selection sort

Q.5.) Write recursive / iterative pseudo code for binary search.

What's Time & space complexity of, Bisection & Binary search.

A.5) low = 0, high = n - 1;

while (low <= high) // loop till search exhausts.

{

 int mid = (low + high) / 2; // find mid value in search

 space & compare it with

 { If (target == num[mid])

 target

 return mid; }

 // if target found

 else

 if (target < num[mid])

 high = mid - 1;

}

```

else {
    low = mid + 1;           // If target > middle element discarded
}                                all element in left slice
return -1;                         // If no element exists in array.
}

```

Recursive Pseudocode

Base condition:

```

↳ if (low > high)
    return -1;
}

```

$mid = (low + high) / 2;$ // find mid value

if ($target = num[mid]$)

{ . return mid; // If base condition met.
 }

else if ($target < num[mid]$)

{ . return binarysearch(num, low, mid - 1, target);
 }

else

" (num, mid + 1, high, target)

Linear Search

Time(C)

$O(n)$

Binary Search

$O(\log_2 n)$

Space(C)

$O(1)$

$O(1)$ $O(\log_2 n)$
Iterative Recursively

- Q.6) Recurrence relation for binary recursive search.

As mean element is selected as pivot array divides in branches of equal size so that height of tree \rightarrow minimum

In worst case time complexity it will be $O(N^2)$ and will happen when array - sorted and smallest + largest element is selected as pivot.

A.6) Recurrence relation of binary search

$$u = T(n) = T(n/2) + 1.$$

A.7) Quicksort is best sorting algo in practical use as it follows locality of reference & also its best case complexity is $O(n \log n)$.

Q.8.) which sorting is best for practical uses, explain.

A.8.) Quicksort is fastest general purpose sort. It's more effective for datasets that fit in memory.
It's an in-place sort (doesn't require extra storage) so it's appropriate for arrays

Q.9.) what do you mean by number of inversions in an array?

count

A.9.) Inversion for an array indicates how far or close is the array from being sorted. If array already sorted inversion count is zero(0). but if array is sorted in reverse order inversion count = maximum

Given array:-

$$[] = \{7, 21, 31, 8, 10, 1, 20, 6, 4, 5\}$$

No. of inversions:-
 $\{(7, 1), (7, 6), (7, 4), (7, 5), (21, 20)$
 $(31, 8), (31, 10), (31, 1), (31, 20)$
 $(31, 6), (31, 4), (31, 5), (8, 1)$
 $(8, 6), (8, 5), (8, 4), (10, 1), (10, 6), (10, 4)$
 $(10, 5)\}$

Q.10.) In which cases quicksort will give best & worst case time complexity.

A.10.) Best case time complexity of quicksort is $O(N \log(N))$ & that will be when pivot is selected as mean element

Q.1.) Write recurrence relation of merge & quick sort on best & worst case. what are similarities & differences between the two?

A.1.) Recurrence relation of merge sort in best case:-

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

on solving $T(n) = O(n \log n)$

" " quick sort

$$T(n) = T(n-1) + O(n)$$

worst case: $O(n^2)$

Merge Sort

Quick Sort

- array partitioned into 2 halves
- Array is partitioned into any ratio.
- Worst + average case has same complexity $O(n \log n)$.
- requires lots of comparisons.
- can work well with any type datasets irrespective of size.
- Not in place, requires additional memory space to store auxiliary arrays.
- Quicksort is in place as it doesn't require any additional storage.
- Stable
- Unstable
- linked lists.
- Arrays.

Q. 12.) Selection sort can be made stable if instead of swapping, minimum element is placed in its position without swapping.

eg:-

```
void SSort(int a[], int n)
for (int i=0; i<n-1; i++) // Iteration
```

```
    int min=i; // find min element from arr[i] → arr[n-1]
```

```
    for (int j=i+1; j<n; j++)
        if (a[min]>a[j])
```

```
            min=j;
```

```
    int key=a[min]; // Make min element at current i.
```

```
    while (min>i)
```

```
}
```

```
    a[min]=a[min-1];
```

```
    min--;
```

```
}
```

```
    a[i]=key;
```

```
}
```

```
}
```

Q. 13.) Bubble sort scans whole array even when array is sorted. Can you modify bubble sort so that it doesn't scan whole array once sorted.

A. 13.) Bubble sort can be optimized if inner loop doesn't cause any swap.

```
void bubbleSort(int arr[], int n)
```

```

    {
        int i, j;
        bool swapped;
        for (i=0; i<n-1; i++)
        {
            swapped = false;
            for (j=0; j<n-i-1; j++)
            {
                if (arr[j] > arr[j+1])
                {
                    Swap(arr[j], arr[j+1]);
                    swapped = true;
                }
            }
            if (swapped == false)
                break;
        }
    }

```

Q.14) Your computer has RAM = 2GB & Array = 4GB - sorting which algorithm will be used & why? Explain external + internal sorting.

In such case merge sort will be preferred as it is an external sorting algorithm i.e. data is divided into chunks & sorted using merge sort.

A.14.)

Internal Sorting

- All data is stored in memory at all times, while sorting - progress.
- shell sort applicable i.e. access whatever array element you wish to e whenever.

External Sorting

- Data stored outside memory & loaded to memory in small chunks.
- Shell sort not applicable
 \because arr not entirely in memory
 \therefore random access not possible