Date __/__/__ Page No.: _____

Tutorial - 3

Q1 - Write linear search Pseudocode to search an element in a sorted array with minimum comparisons.

No. of comparisons can be reduced by copying the element to be searched (suppose x) to last location. So, that one last comparison when X is not present in arr[ ] - 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++ // No termination condition
    if arr[i] == x ; // execute at most n
                     times ; i.e, most n
                     comparisons

arr[n-1] = backup
return (i<n-1) 1 / 1 comparison
    return found ;
    else
    return not found ;
```

Q2 - Write Pseudocode for Iterative & Recursive Insertion sort why Insert is also called Online sort.

Date ___/___/_____ Page No: _____

Ans 2 - Iterative Pseudocode :

```
// sort an arr[] of size n
insertionsort (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 :

```
Basecase — If (n <= 1)
        return ; // If arr 1 or smaller, return

sort first (n-1) elements
    i.e, [arr, n-1]
last = arr [n-1]        // Insert last element at correct position

    j = n-2
Apply Isort algo
    Print utility of arr of size n
```

- Insertion sort is called online sort because it does not need to know about what values it will sort & information is requested while algo runs
- whereas selection is offline sort

Q3 - Complexity of all sorting Algorithms :-

selection sort	—	$\Omega(n^2)$
bubble sort	—	$\Omega(n)$
Insertion	—	$\Omega(n)$
Heap	—	$\Omega(n \log(n))$
Quick	—	$\Omega(n \log(n))$
Merge	—	$\Omega(n \log(n))$

Chitra

Date ___/___/_____ Page No.: _____

Count — O(n+k)
Radix — O(1K)

Q4 - Divide all sorting algorithm into inplace / stable / others

Ans4 - Inplace : Bubble sort, Selection sort, Insertion & Heap sort

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

Unstable : Quick sort, Heap & selection sort

Q5 - Write recursive / iterative pseudocode for Binary search. What's time & space complexity of linear & Binary search.

Ans5 - 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 target
if (target == num[mid])
{
return mid; }
use
if (target < num[mid])
} high = mid - 1 ;
}

```
else {
    low = mid + 1 ;          // If target > middle element discard all element in left space
}
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 Binary search (num, low, mid-1, target)
  }
else
    " (num, mid+1, high, target)
```

	Linear Search	Binary Search	
Time (C)	$O(n)$	$O(\log_e n)$	
Space (C)	$O(1)$	$O(1)$ Iterative	$O(\log_e n)$ Recursively

Date ___/___/_____ Page No.: _____

Q6 - Recurrence relation for binary recursive search.

Ans6 - As mean element is selected as pivot array divides in branches of equal size. So that height of tree → minimum

In worst case, time complexity it will be $O(N^2)$ & will happen when array - sorted & smallest to largest elements is selected as pivot.

Recurrence relation of binary search

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

Ans7 - Quick sort is best sorting algo in practical case as it follows locality of reference & also it's best case complexity is $O(\log(n))$

Q8 - which sorting is best for practical uses. Explain.

Ans8 - 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.

Q9 - What do you mean by number of inversions in an array?

Ans9 - Inversion count 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

Date ___/___/_____ Page No.: _____

Given array :

arr[] = {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)}

Q10 - In which cases Quick sort will give best & worst case time complexity ?

Ans 10 - Best case the complexity of quicksort is O(N log(N)) & that will be when pivot is selected as mean element.

Q11 - Write recurrence relation of merge & quick sort in best & worst case. What are similarities & differences between the two ?

Recurrence relation of merge sort in best case :-

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

On solving, $T(n) = O(n \log(n))$ Ans

" " quick sort :-

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

worst case : $O(n^2)$ Ans

Merge Sort	Quick Sort
→ Array parted into 2 halves	→ Array is parted into any ratio
→ worst + avg. case has same complexities $O(n \log(n))$	→ Worst case $T(c) = O(n^2)$ requires lots of comparisons

Date ___/___/_____ Page No.: _____

→ Can work well with any type dataset irrespective of size	→ Does not work well with large dataset
→ Not in place, requires additional memory space to store auxiliary array	→ Quicksort is in place as it doesn't require any additional storage
→ Stable	→ Unstable
→ Linked lists	→ Arrays

Ans 12 - Selection sort can be made stable if instead of swapping, minimum element is placed in it's position without swapping.

Eg :-

```
Void sort (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] ; // move min element at current
                       i.
    while (min > i)
    }
      a [i] = key
    {
}
```

Chitra

Date ___/___/___ Page No: ___

Q13 - 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.

Ans13 - Bubble sort can be optimized if inner loop doesn't cause any swap.

```
Void bubble sort (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 ;
 }
}
```

Q14 - Your computer has RAM = 2GB & Array = 4GB - sorting which algorithm will be used & why ? Explain external + internal sorting.

Date ____/____/_____ Page No.: _____

Ans 14 - In such case, merge sort will be efficient as it is an external sorting algorithm, i.e, data is divided into chunks & sorted using merge sort

	Internal Sorting	External Sorting
→	All data is stored in memory at all times, while sorting - progress	→ Data stored outside memory & loaded to memory in small chunks
→	Shell sort applicable i.e, access whatever array elements you wish to & whenever	→ Shell sort not applicable ∵ "arr" not entirely in memory, therefore, random access not possible