

29

## ASSIGNMENT - Q3

## ① Pseudocode for Linear search

```
for ( i=0 to n )
{   if ( arr[i] == value )
    // element found
}
```

## ② void recursiveInsertion( int arr[], int n )

```
{   if ( n <= 1 )
    return;
```

```
recursiveInsertion( arr, n-1 );
```

```
int nth = arr[n-1];
```

```
int j = n-2;
```

```
while ( j >= 0 & arr[j] > nth )
{
```

```
    arr[j+1] = arr[j];
    j--;
}
```

```
arr[j+1] = nth;
```

```
y
```

⇒ Iteration :

```
for ( i=1 to n )
```

```
{   key ← arr[i]
```

```
    j ← i-1
```

```
while ( j >= 0 and arr[i] > key )
```

```
{   arr[j+1] ← arr[j]
```

```
    j ← j-1
}
```

$\{ \text{Arr}[1..n] \leftarrow \text{key} \}$

(3)

Complexity of all sorting Algorithm

(4)

	Best	Worst	Average
a) Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
b) Bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$
c) Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$
d) Heap sort	$O(n \log(n))$	$O(n \log(n))$	$O(n \log(n))$
e) Quick sort	$O(n \log(n))$	$O(n^2)$	$O(n \log(n))$
f) Merge sort	$O(n \log(n))$	$O(n \log(n))$	$O(n \log(n))$

(5)

Inplace Sorting

Bubble

Selection

Insertion

Quick sort

Heap sort

Stable sorting

Merge sort

Bubble

Insertion

Count

Online sorting

Insertion.

(6)

Recursive Binary Search

int binarySearch ( int arr[], int l, int r, int x )

{ if ( r >= l )

{ int mid = l + (r-l)/2;

if ( arr[mid] == x )

return mid;

if ( $\text{arr}[\text{mid}] > \text{x}$ )  
return  $\text{binarySearch}(\text{arr}, \text{l}, \text{mid}-1, \text{u});$

return  $\text{binarySearch}(\text{arr}, \text{mid}+1, \text{r}, \text{x});$

5

return -1;

3

Implementation

int  $\text{binarySearch}(\text{int arr}[], \text{int l, int r, int x})$

l

while ( $\text{l} \leq \text{r}$ )

{ int  $m = \text{l} + (\text{r} - 1) / 2;$

if ( $\text{arr}[m] \geq \text{x}$ )

return  $m;$

if ( $\text{arr}[m] < \text{x}$ )

$\text{l} = m + 1;$

else

$\text{r} = m - 1;$

3

return -1;

y

Time complexity recursive  $\Rightarrow O(\log n)$

Binary search

Linear search  $\Rightarrow O(n)$

6 Recurrence relation for binary search

$$T(n) = T(n/2) + 1 \quad \text{--- (1)}$$

$$T(n/2) = T(n/4) + 1 \quad \text{--- (2)}$$

$$T(n/4) = T(n/8) + 1 \quad \text{--- (3)}$$

$$\Rightarrow T(n) = T(n/4) + 1 + 1 \\ = T(n/8) + 1 + 1 + 1$$

$$\vdots \\ \Rightarrow T(n/2^k) + 2(k \text{ times})$$

let  $\textcircled{13} = 2^k = n$   
 $k = \log n$

$$\therefore T(n) = T\left(\frac{n}{n}\right) + \log n$$

$$T(n) = T(1) + \log n = O(\log n)$$

8) Quick sort is the fastest general purpose sort.

In most practical situations, quicksort is the method of choice. If stability is important and space is available, merge sort might be best.

9) A pair ( $a[i], a[j]$ ) is said to be inversion if  $a[i] > a[j]$ .

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

Total number of inversion are 31. using merge sort.

### (10) Worst Case in Quick Sort

The worst case time complexity of a Quick sort is  $O(n^2)$  if the picked pivot element is always an extreme (smallest or largest element). Or for given array is sorted and we pick either first or last element.

### Best Case in Quick Sort

The best case is  $O(n \log(n))$  when we will select pivot element as a mean element.

### (11) Quick Sort

#### Worst Case

$$T(0) = T(1) = 0 \quad (\text{base})$$

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

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

$$T(n-1) = (n-1) + T(n-2)$$

$$T(n-2) = (n-2) + T(n-3)$$

$$T(n) = n + n-1 + T(n-2)$$

$$T(n) = n + n-1 + n-2 + T(n-3)$$

$$T(n) = n (\text{k times}) - (k) + T(n-k)$$

~~Worst Case~~  $k = n$

$$\begin{aligned}\therefore T(n) &= n \times n + n + T(n-n) \\ &= n^2 + n + T(0) \\ \therefore T(n) &= O(n^2).\end{aligned}$$

Best Case

$$\begin{aligned}T(0) &= T(1) = O \quad (\text{base}) \\ T(n) &= 2T(n/2) + n \quad - \textcircled{1}\end{aligned}$$

$$T(n_2) = 2T(n_4) + \frac{n}{2} \quad - \textcircled{2}$$

$$T(n_4) = 2T(n_8) + \frac{n}{4} \quad - \textcircled{3}$$

$$\therefore T(n) = 2 \left( 2T\left(\frac{n}{4}\right) + \frac{n}{2} \right) + n$$

$$\begin{aligned}T(n) &= 2 \left( 2 \left( 2T\left(\frac{n}{8}\right) + \frac{n}{4} \right) + \frac{n}{2} \right) + n \\ &= 4T\left(\frac{n}{2^3}\right) + n + n + n.\end{aligned}$$

$$T(n) = 2^K T\left(\frac{n}{2^K}\right) + n \text{ (K times).}$$

$$\begin{aligned}\text{let } 2^K &= n \\ K &= \log n\end{aligned}$$

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

$$T(n) = \log n T(1) + n \log n$$

$$T(n) = \log n + n \log n$$

$$T(n) = O(n \log n).$$

Quick Sort

→ Splitting of an array of elements in in any ratio, not necessarily divided into half

- Worst complexity  $O(n^2)$
- It works well on small arrays
- It works faster than other sorting algo for small data eg: Selection sort.
- Internal sorting method

Merge Sort

→ In the merge sort the array is partitioned into just two halves.

→  $O(n \log n)$

- It operates fine on any size of array.
- It has an consistent speed on any size of data.

→ Internal sorting method

(12)

Stable Selection Sort

```
for (int i=0 ; i<n-1 ; i++)
```

```
{ int min = i;
```

```
    for (int j = i+1 ; j < n ; j++)
```

```
        if ( a[min] > a[j] )
```

```
            min = j;
```

```
    }
```

```
    int key = a[min];
```

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

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

```
        min--;
    }
```

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

```
}
```

(13) A better version of bubble sort, known as modified bubble sort, includes a flag that is set if an exchange is made after an entire pass over the array. If no exchange is made, then it should be clear the array is already sorted because no two elements need to be switched. In that case the sort is -ed end.

```
void bubble ( int a[], int n )
{
    for ( int i = 0; i < n; i++ )
    {
        int swaps = 0;
        for ( int j = 0; j < n - i - 1; j++ )
        {
            if ( a[j] > a[j + 1] )
            {
                int t = a[j];
                a[j] = a[j + 1];
                a[j + 1] = t;
                swaps++;
            }
        }
        if ( swaps == 0 )
            break;
    }
}
```