Name→ Tushar sharma

Roll.no→ 28

Section→ CST core

DAA Tutorial-3

HITAISHI Page ______ Date ______

(1) Psoudocode for linear search

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

=> Iterative.

```
      for   i=1 to n
   {        key ← A [i]
            j ← i-1
      while ( j>=0  and   A [i]> key)
            {  A [j+1]  ← A [j]
                   j ← j-1
            }
```

Date

A[j+1] ← 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))

④

Inplace Sorting	Stable Sorting	Online Sorting
Bubble	Merge Sort	Insertion.
Selection	Bubble	
Insertion	Insertion	
Quick sort	count	
Heap sort		

⑤ 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 (arr [mid] > x)
    return binarySearch (arr, l, mid - 1, x);

return binarySearch ( arr, mid + 1, r, x);
}
return -1;
}
```

Itterative

```
int binary Search ( int arr [], int l, int r, int x)
{
        while ( l <= r)
        {       int m = l+(r-l)/2;
            if ( arr [m] == x)
                  return m;
        if ( arr [m] < x)
            l = m +1;
        else
              r = m-1;
        }
    return -1;
}
```

Time complexity recursive => O(log n)
  Binary search

Linear search => O(n)

6) Reccurrence relation for binary search

$T(n) = T(n/2) + 1$ — ①

$T(n/2) = T(n/4) + 1$ — ②

$T(n/4) = T(n/8) + 1$ — ③

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

$= T(n/8) + 1 + 1 + 1$

$\vdots$

$\Rightarrow T(n/2^k) + 1$ (k times)

let $2^k = n$

$k = \log n$

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

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

8) ⇒ Quicksort 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];

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 $O(n^2)$ ~~when~~ if the picked pivot element is always an extreme (~~right~~ smallest or largest element).
Or the 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

$T(0) = T(1) = 0$ (base)
$T(n) = n + T(n-1)$

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

$T(n) = n + n-1 + T(n-2)$
$T(n) = n + n-1 + n-2 + T(n-3)$
⋮
$T(n) = n\,(k \text{ times}) - (k) + T(n-k)$

let $k = n$

$\therefore T(n) = n \times n + n + T(n-n)$

$= n^2 + n + T(0)$

$\therefore T(n) = O(n^2)$.

Best Case

$T(0) = T(1) = 0$ (Base)

$T(n) = 2T(n/2) + n$ — ①

$T(n/2) = 2T(n/4) + \frac{n}{2}$ — ②

$T(n/4) = 2T(n/8) + \frac{n}{4}$ — ③

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

$T(n) = 2\left(2\left(2T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + \frac{n}{2}\right) + n$

$= 4T\left(\frac{n}{2^3}\right) + n + n + n$

⋮

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

let $2^k = n$

$k = \log n$

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

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

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

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

Quick Sort	Merge Sort
→ splitting of a array of elements is in any ratio, not necessary divided into half.	→ In the merge sort the array is parted into just two halves.
→ Worst complexity O(n)	→ O(n logn)
→ It works well on small array	→ It operates fine on any size of array.
→ It works faster than other sorting algo for small data eg: selection sort.	→ It has an consistent speed on any size of data.
→ Internal sorting method	→ External 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 bette version of bubble sort, known as modified bubble sort, includes a flag that is set if an exchang is made after an entire pass over the array. If no exchange is made, then it should be clear th the array is alread order because no two elemen need to be switchead. In that case the sort s -ld 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;
     }
}
```