Tutorial-3

Anchit Sonha
CST-SPL 1
55

1- Pseudocode for Linear Search

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

2-

```
void insertion(int arr[], int n) //recursive
{
  if(n<=1)
    return;
  insertion(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;
}

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

→ Insertion sort is an online sorting because it doesn't know the whole input, more input can be inserted with the insertion sorting is running.

Q3- Complexity

Name	Best	Worse	Average
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$
Heap	$O(n\log n)$	$O(n\log(n))$	$O(n\log(n))$
Quick	$O(n\log(n))$	$O(n^2)$	$O(n\log(n))$
Merge	$O(n\log(n))$	$O(n\log(n))$	$O(n\log(n))$

Q4- Inplace sorting
Bubble
Selection
Insertion
Quicksort
Heapsort

Stable sorting
Mergesort
Bubble
Insertion
Count

Online sorting
Insertion

Q5-
```
int bsearch(int arr[], int l, int r, int n) //recursive
{
    if (r>=l)
    {
        int mid= l+(r-l)/2;
        if(arr[mid]==x)
          return mid;
        else if (arr[mid]>x)
          return bsearch(arr,l,mid-1,x);
        else
          return bsearch(arr,mid+1,r,x);
    }
    return -1;
}
int bsearch(int arr[], int l, int r, int x)
while(l<=r)
{
    int m= l+(r-l)/2;
    if(arr[m]==x)
      return m;
    else if (arr[m]>x)
      r=m-1;
    else
      l=m+1;
}
```

```
    return -1;
}
```

Time Complexity of

Binary Search : O(log n)
Linear Search : O(n)

Q6- Recursive relation for binary recursive search

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

where $T(n)$ is the time required for binary search on an array of size 'n'.

Q7-
```
Int ferd[A[], n, k)
{
    Sort (A, n)
    for (i=0 to n-1)
    {
        n = binarySearch(A, 0, n-1, K-A[i])
        if (n)
        return 1
    }
    return -1
}
```

Time Complexity $= O(n \log(n)) + n \cdot O(\log n)$
$= O(n \log(n))$

Q8- Quick Sort is the fastest general purpose sort. In most practical situations, quicksort is the method of choice. If stability is important and ~~no~~ space is available, merge sort might be best.

Q9- A pair $(a[i], a[j])$ is said to be inversion if $a[i] > a[j]$.
In arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5}.
Total no. of inversion are 31 using merge sort.

Q10- The worst case time complexity of Quick Sort is $O(n^2)$. This case occurs when the picked pivot is always an extreme (smallest or largest) element. This happens when input array is sorted or reverse sorted.
The best case of quick sort is when we will select pivot as a mean element.

Q11- Recursive relation of

Merge sort → $T(n) = 2T(n/2) + n$

Quick sort → $T(n) = 2T(n/2) + n$

↳ Merge sort is more efficient and works faster than quick sort in case of larger array size or datasets.

↳ Worst case complexity for quick sort is $O(n^2)$ whereas $O(n\log(n))$ for merge sort.

Q12- Stable selection sort

```
void stableselection(int arr[], int n)
{
    for(int i=0; i<n-1; i++)
    {
        int min=i;
        for(int j=i+1; j<n; j++)
        {
            if(arr[min]>arr[j])
                min=j;
        }
        int key= arr[min];
        while(min>i)
        {
            arr[min]=arr[min-1];
            min--;
        }
        arr[i]=key;
    }
}
```

Q13- Modified Bubble Sorting

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