Aishwarya Kumbar

CST . SPL 1

Date.

Page No.

6

TUTORIAL 3

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

Date:
Page No.

A[j+1] ← key

}

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

05

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 (i)

Inplace sorting	Stable sorting	Online sorting
Bubble	Mergesort	Insertion
Selection	Bubble	
Insertion	Insertion	
Quick sort	Count	
Heapsort		

Q5

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

Time complexity of binary search
= $O(\log n)$

linear search → $O(n)$

(6) Recurrence relation for binary recursive search

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

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

(7)
```
int find(A[], n, k)
{
    sort(A, n)
    for (i=0 to n-1)
    {   x = binary search(A, 0, n-1, k-A[i])
        if (x)
            return 1
    }
    return x = 1
}
```

Time complexity: $O(n \log(n)) + n \cdot O(\log n)$
$= O(n \log(n))$

(8) Quick sort is the fastest general purpose sort.
In most practical situations, quick sort 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].

(10) The worst case time complexity of quicksort 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 quicksort is when we will select pivot as a mean element.

(11) Recurrence Relation of
Merge sort → $T(n) = 2T(n/2) + n$
Quick sort → $T(n) = 2T(n/2) + n$.

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

Date.
Page No.

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

Q3 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;
  }
}
```

Q3 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[j];
      a[j] = a[j+1];
      a[j+1] = t;
      swaps ++;
    }
  }
  if (swaps == 0)
    break;
  }
}
```