Aditya Gupta
CST SPL-1-25

Date / /

Tutorials-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[i];
         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
      }
```

Date / /

A [j+1] ← key
}

③ 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	$O(n\log(n))$	$O(n\log(n))$	$O(n\log(n))$

④

Inplace sorting	Stable sorting	Online sorting
Bubble	Merge	Insertion
Selection	Bubble	
Insertion	Insertion	
Quick	count	
Heap		

⑤

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

        else
            return binary (arr, m+1, r, x);
```

Date / /

```
        }
        return -1;
}

int binary (int arr[], int l, int r, int n)
{
        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)

(6) Recurrence relation for binary recursive search.

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

(7)

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

Date / /

return 1

}

return -1

}

Time complexity = $O(n \log(n)) + n\,O(\log n)$

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

(8) • Quick sort is the fastest general purpose sort.
• In most practical situations, quick sort is the method of choice. If stability is important & 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])

In arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5}
Total no. of inversions are 31, using merge sort.

(10) Worst case time complexity of quick sort is $O(n^2)$
This case occurs when the picked pivot is always an extreme element. This happens when input array is sorted or reverse sorted.

(11) Recurrence Relation of

Merge sort → $T(n) = 2T(n/2) + n$
Quick sort → $T(n) = 2T(n/2) + n$

Date / /

- Merge sort is more efficient & works faster than quick sort in case of larger array of size or datasets.
- Worst case complexity for quick sort is $O(n^2)$ whereas $O(n \log n)$ for merge sort.

(17) Stable Selection Sort

```
void stable selection (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;
     }
}
```

(18) 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-i; j++)
          { if (a [j]> a [j+1])
```

Date / /

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