Name - Gaurav Bhandari Sec - F Roll no - 58

U. Roll no - 2016747 ID - 20021182

Tutorial - 3

①
```
for ( i=0 to n)
{  if (arr[i] == value)
       // element found
}
```

② **Iterative**

```
void insertion_sort (int a[], int n)
{  for (int i=ϕ; i<n; i++)
    {
        j = i-1;
        x = A[i];
        while ( j>-1 && A[i]>x)
        {
          A[j+1] = A[j]
          j--;
        }
        A[j+1] = x;
    }
}
```

Recursive

```
void insertionsort (int a[], int n)
{  if (n<=1)
     return;
   insertionsort (a, n-1);
   int last = a[n-1];
   int j = n-2;
while (i>=0 && a[~~j~~ i] > last)
{
  a[i+1] = a[i];
  j--;
}
a[j+1] = last;
}
```

Insertion sort is called online sort because it doesn't need to know anything about what value it will sort & the information is requested while the algorithm is running.

<u>Other sorting Algorithm</u>

- Bubble sort
- Quick sort
- Merge sort
- Selection sort
- Heap sort.

③

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

(4) <u>Implace sorting</u>
- Bubble
- Selection
- Insertion
- Quick sort
- Heap sort

<u>Stable sorting</u>
- Merge sort
- Bubble
- Insertion
- count

<u>Online sorting</u>
- Insertion

(5) <u>Iterative</u>

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

<u>Recursive</u>

```
int binarysearch (int a[]; int l , int r, int key)
{ while ( l<=r)
  { ~~d~~ int m = (l+r)/2;
      if (key == a[m])
        return m;
      else if (key < a[m])
        return binarysearch ( a, l, m-1, key);
        else
            return binarysearch(a, m+1, r, key);
      }
      return -1;
  }
```

<u>Time Complexity</u>

- Linear search - $O(n)$
- Binary search - $O(\log n)$

⑥ $T(n) = T(n/2) + 1;$

$T(n/2) = T(n/4) + 1$

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

$$\begin{aligned} T(n) &= T(n/2) + 1 \\ &= T(n/4) + 1 + 1 \\ &= T(n/8) + 1 + 1 + 1 \\ &\vdots \\ &T(n/2^k) + k \end{aligned}$$

Let $2^k = n$
$k = \log n$

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

7.
```
for (int i=0; i<n; i++)
{
  for (int j=0; j<n; j++)
  {
     if (a[i] + a[j] == k)
         printf("%d%d", i, j);
  }
}
```

8. Quick sort is the fastest general purpose sort. In most practical situation quicksort is the method of choice. If stability is important and space is available, mergesort might be best.

9. A pair (a[i], a[j]) is said to inversion is
   - a[i] > a[j]
   - i < j

Total no. of inversion in given array are 31 using merge sort.

10. Worst case - The worst case occur when the picked pivot is $O(n^2)$ always an ~~extra~~ extrence (smallest or largest) element. This happen when input array is sorted or reverse sorted.

<u>Best case</u> : The best case occur when we will select pivot $O(n \log n)$ element as mean elements.

⑪ **Merge sort**

Best case ~~sort~~ - $T(n) = 2T(n/2) + O(n) \rightarrow O(n \log n)$

Worst case - $T(n) = 2T(n/4) + O(n) \longrightarrow O(n \log n)$

Quick sort

Best - $T(n) = 2T(n/2) + O(n) \longrightarrow O(n \log n)$

Worst - $T(n) = T(n-1) + O(n) \longrightarrow O(n^2)$

In Quick sort the array of elements is divided into part repeatedly until it is possible to divide it function. It is not necessary to divide half.

In Merge sort the element are split into two sub array $(n/2)$ again & again ~~ut~~ until only one element is left.

⑫
```
for (int i=0; i<n-1; i++)
{ int ~~no~~ min = i;
  for ( int j = i+1; j<n; j++)
  {  if (a[min] > a[i])
         min = j;
  }
  int key = a[min];
  while (min>i)
  {
    a[min] = a[min-j];
    min--;
  }
  a[i] = key;
}
```

(13) A better version of bubble sort known as m-bubble sort include a flag that is ~~s~~ let if a exchange is made after on entire pass over the array. If no exchange is made, then it should be classified the array is already sort because no two element need to be sorted. ~~B~~ In that case sort is minimum.

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