Assignment-2

Q-1 write linear search code in sorted array with minimum comparisons.

Ans-

```
function LS(arr[], n, k)
(where n=number of terms and k is the element
 to be searched)
{
    flag=0
    for(i to n-1)
    {
        if(k==arr[i])
        {
            flag=1;
        }
    }
    return i; // returns 1 if element found
}
```

~~Q- write a~~

Q-2 Write Pseudocode for iterative and recursive insertion sort. why is the insertion sort called an online sorting. why?.

Ans-

a. Recursive approach:-

```
function insertion( arr[], n)
  {
     if (n <= 1)
      {
        return
      }
   insertion( Arr, n-1)
     last = arr[n-1]
     j = n-2
     while( j >=0 && arr[j] > last)
        {
          arr[j+1] = arr[j]
          j--
        }
     arr[j+1] = last;
}
```

b- iterative approach:-

```
function insertion (arr[], n)
{
    for (i from 1 to n-1)
    {
        key = arr[i]
        j = j-1
        while (j>=0 and arr[j]>key)
        {
            arr[j+1] = arr[j];
            j = j-1
        }
        arr[j+1] = key;
    }
}
```

Insertion sort is called online sort, since it's Algorithm, the element with ~~the~~ same value can be both the elements with the same value, in the sorted array too.

other on the sorting:

1. insertion sort — ✓
2. Merge sort — ✗
3. Heap sort — ✓
4. quick sort — ✓
5. Radix sort — ✓
6. Count sort — ✓
7. Bubble sort — ✓
8. selection sort — ✓

Q-3 Draw Complexity table for all the sorting algorithms discussed in the class.

Ans- Sort	Time Complexity	Space Complexity
(i) Bubble	$O(n^2)$	
(ii) Heap	$O(n \log n)$	$O(1)$
(iii) selection	$O(n^2)$	$O(1)$
(iv) insertion	$O(n^2)$	$O(1)$
(v) Quick	$O(n^2)$	$O(1)$
(vi) merge	$O(n \log n)$	$O(n)$
(vii) Counting	$O(n+k)$	$O(n)$
(viii) Radix	$O(n*k)$	$O(n+k)$

☆ All for worst cases.

4 Discuss for all the sorting Algorithms. write there they are. In place, stable or online.

Sort	In Place	Stable	On-line
Bubble	1	1	0
Heap	1	0	0
Insertion	1	1	1
Quick	1	0	0
Count	0	1	0
Merge	0	1	0
Radix	0	1	0
Selection	1	0	0

5- Write Pseudocode for Binary Search - compare it's space and time complexity.

Ans. 5

Binary Search

Iterative Approach:

```
function BS(arr[], n, key) /* (condition -> array must be sorted */
{
    r=n;
    l=0;
    while(l<=r)
    {
        mid = (l+(r-l)/2
        if (arr[mid] == key)
        {
            return true;
        }
        else if (arr[mid] > key)
        {
            l = mid-1
        }
        else
        {
            r = mid-1;
        }
```

- U loop ends i.e element not found
return false;

Recursive Approach:-

```
function BS(arr[], l, r, key)
{ if (l<=r)
  {
  mid=(l+(r-l)/2)
  if(arr[mid] == key)
      return true;
  else if (arr[mid] <-key)
      return BS(arr, mid+1,
        r, key)
  else
      return BS(arr, 0, mid-1, key)
}
return false;
```

Linear search (iteration)	Binary Search (iteration)
T.C = O(n) S.C = O(1)	T.C = O(log n) S.C = O(1)
Linear Search (recursive)	**Binary Search (Recursive)**
T.C = O(n) S.C = O(1) No stack used	T.C = O(log n) S.C = O(log n) Stack to Recursive calls

Q-6 write down the recurrence Relation for Binary Search.

Ans-

Pseudocode :-

```
T(n) ⇒ BS(a, i, j, x)  —  O(1)
       mid = (i+j)/2   —  O(1)
       if (a[mid] == x)
          return[mid]
       if (a[mid] > x)
       ** return BS(a, i, mid-1, x)  ← T(n/2)
```

else
** return BS(a, mid+1, b, x) ← T(n/2)

Hence, $\boxed{T(n) = T(n/2) + 1}$

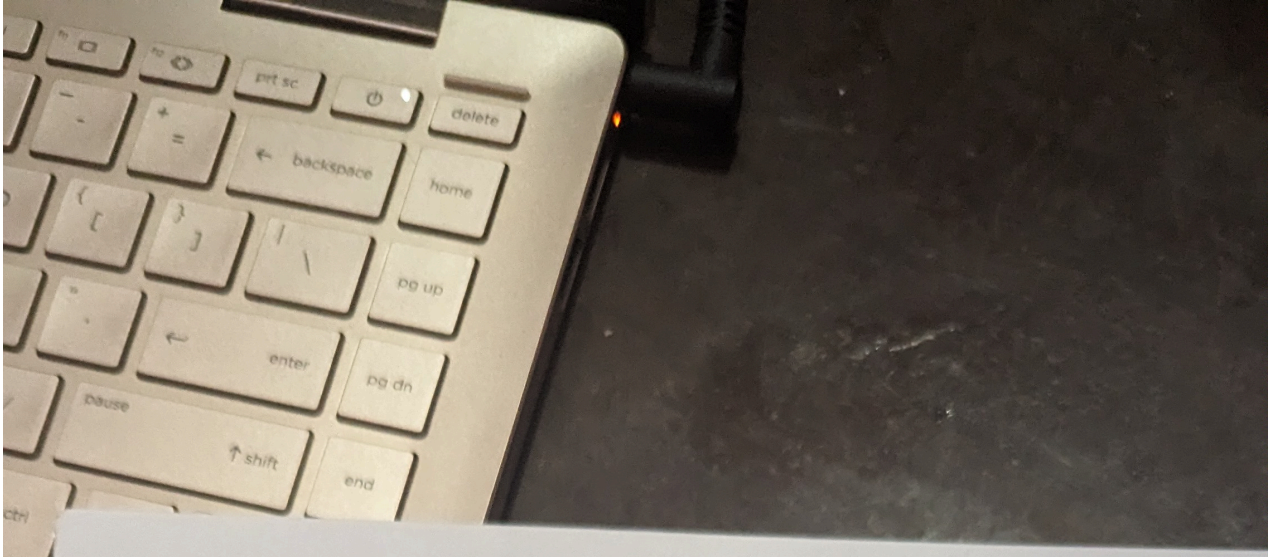
Q-7 which is the best sorting for Practical uses? explain

Ans- Quicksort is often the preferred sorting algorithm for Practical applications. It's one of the most efficient sorting Algorithms and it's generally considered the fastest. Some real world-applications of sorting include: searching, element uniqueness, telephone dictionaries, Ranks based on scores.

Q-13 Bubble sort scans whole array even when array is sorted. (or it be modified so that it doesn't scan the whole array once it is sorted.

Ans- Yes. Bubble sort can be optimized for better usage for partially or fully sorted arrays:-

```
function BubbleSortArr(arr[], n)
    ~~n=length~~
    for (i=0; i<n; i++)
    {
      swapped = False;
      for (j=0; j<n-i-1; j++)
      { if (arr[j] > arr[j+1])
          {
            swap;
          }
      }
    if (not_swapped)
        break;
```



In this modification, the swapped flag has been used to define whether any swaps were made during the inner loop.