Name Hrishabh Upadhyay
Rollno : 32

Date.
Page No.

Assignment : 02

Ques-1) Linear search Pseudocode :

```
bool linearsearch ( int arr [], int key)
{
    for ( i=0 ; i< n ; i++)
    {
        if (arr [i] == key)
            return true ;
    }
        return false;
}
```

Ques-2) Pseudo code for iterative insertion sort :

ITERATIVE

```
for( int i=1; i<n; i++)
{
    int t = a[i];
    int j = i-1;
    While (j>=0 && a[j] >t)
    {
        a[j+1] = a[j]
        j--;
    }
    a[j+1] = t;
}
```

RECURSIVE :

```
void sort (int arr [], int n)
{
     if ( n< = 1)
          return;
     sort (arr, n-1);
     int last = arr [n-1];
     int j= n-2;
     while (j >= 0 && arr [j] > last)
     {
          arr [j+1] = arr [j];
           j= j+1;
     }
     arr [j+1] = last.
}
```

→ It can sort elements while receiving new ones thats why it is called online sorting.
→ Other sorting techniques like merge, Quick, selection can't do this.

Date.
Page No.

Ques 3>

Sorting technique	best	avg	worst
	COMPLEXITY		
Bubble	$O(n)$	$O(n^2)$	$O(n^2)$
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$
Count	$O(n+k)$	$O(n+k)$	$O(n+k)$
Quick	$O(n\log n)$	$O(n\log n)$	$O(n^2)$
Merge	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$
heap	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$
Radix	$O(d*(n+k))$	$O(d*(n+k))$	$O(d*(n+k))$

Date. ______
Page No. ______

Ques 4)

Sorting techniques	Inplace	Stable	Online
Bubble	Yes	Yes	No
Selection	Yes	No	No
Insertion	Yes	Yes	Yes
Count	No	Yes	No
Quick	Yes	No	No
Merge	No	Yes	No
heap	Yes	No	No
Radix	No	Yes	No

Date.
Page No.

Ques5) Recursive Binary Search:

```
int BinSearch ( int arr[], int target, int l, int h)
{
    if (l > h) {
        return -1;   // element not found
    } mid = l+(h-l)/2;
    if (arr[mid] == target) {
        return mid;
    }
    if ( target < arr [mid]) {
        return binsearch (arr, l, mid-1, target);
    } else
        return binsearch (arr, mid+1, high, target);
}
```

Iterative binary Search:

```
int binSearch (int arr[], int target) {
{
    low = 0;
    high = length (arr) -1;
    While (low <= high)
    {
    mid = low + (high - low)/2;
    if (arr [mid] = = target);
        return mid;
    elseif (arr [mid] < target)
        low = m+1;
    else
```

Date. ________
Page No. ________

```
        high = mid-1;
    }
    return -1;
}
```

Ques-6 Dividing the array until target element is found or the array search is resulted in out of bound

1. Calling the fnc with parameter arr as sorted array target as element to search and low, high as starting and ending point of ~~seach~~ search.
2. base case if low > high that is target not found
3. Otherwise
   - mid will be ~~call~~ calculated.
   - and check if arr's mid == target
     target found at mid which is returned.
   - and if arr's mid < target
     high is set to mid-1 ~~is return~~
   - and if arr's mid > target.
     low = mid + 1

pseudo code

```
fnc binsearch (arr, low, high, target) {
    if (low > high)
        return -1
    mid = (low + high)/2
    if (arr[mid] == target)
        return mid;
```

Date.
Page No.

```
elseif (arr [mid] > target);
        ~~high = mid-1~~
      return binsearch (arr, low, m-1, target);
else
   return bin searh (arr, mid+1, high target);

}
```

Ques-7)

```
int findpairsumK (arr, k) {
   sort (arr);
   int left = 0;
   int right = length (arr) -1;
   while (left < right) {
     if (arr [left] + arr [right] == k)
         return 1        // found
     else if (arr [left + arr [right < k)
           left = left +1
     else
         right = right-1
   }
   return -1        //not found.
}
```

Date. ______

Page No. ______

Ques 8) • QuickSort – fastest sorting algo especially for large dataset.

• Merge sort / Heap sort for moderate dataset

• Insertion or Selection Sort for small dataset

• Insertion sort for nearly sorted data.

Ques 9)

Inversion: When smaller element is after larger element if the sorting is ~~ougt~~ ought to be in ascending order or vice versa for descending order.

{7, 21, 31, 8, 10, 1, 20, 6, 4, 5}

$n = 10 \gg mid = \frac{10}{2} = 5$

{7, 21, 31, 8, 10, 1, 20, 6, 4, 5}

{7, 21, 31, 8, 10} {1, 20, 6, 4, 5}

{7, 21} {31, 8, 10} {1, 20} {6, 4, 5}

{7} {21} {31} {8, 10} {1} {20} {6} {4 5}

{8} {10} {4} {5}

Date.
Page No.

{7}{21} {31} {8} {10} {1}{20}{6} {4} {5}

+1 +1 +1

{7, 21} {8, 31} {1, 10} {6, 20} {4, 5}

+ +1

{7, 8, 21, 31} {1, 6, 10, 20}

+1 +1

{7, 21} {8, 31} {10} {1, 20} {4, 6} {5}

+1 +1 +1

{7, 8, 21, 31} {10} {1, 4, 6, 20} {5}

+1 +1 +1 +1

{7, 8, 10, 21, 31} {1, 4, 5, 6, 20}

+5 +5 +5 +5 +2

{1, 4, 5, 6, 7, 8, 10, 20, 21, 31}

No. of inversion : $\underline{5+5}+\underline{5+5}+2+1+1+1+1+1$
$+1+1+1+1$
$= 31$

Ques 10) Best case: When pivot divids array in equal values.
Worst case: When pivot divides array in highly imbalanced way.

Date.
Page No.

Merge sort:

best case: $T(n) = 2T(n/2) + O(n)$

Worst cast $T(n) = 2T(n/2) + O(n)$

Quick Sort

best case $T(n) = T(n/2) + T(n/2) + O(n)$

Worst case $T(n) = T(n-1) + O(n)$

Similarities:

both have divide and conquer

$TC = O(n \log n)$

Difference In Quick Sort TC varies.

Ques-12)
```
void Selection Sort (arr, n) {
    for (i = 0; i<n; i++)
    {
        int min = i;
        for (j = i+1 ; j<n; j++) {
            if (arr [j] < arr [min])
                min = j;
        }
        minvalue = arr [min];
```

Date.
Page No.

```
while (min > i)
{
    arr[min] = arr [min-1]
    min = min-1;
}
  arr [i] = minvalue;
 }
}
```

Ques-(3)

```
void sort (arr, n) {
bool swapped;
for ( int i=0; i < n-1; i++) {
  swapped = false;
  for( int j= 0; j < n-1; j++) {
   if (arr[j] > arr [j+1])
   {
       swap (arr[j], arr [j+1]);
       swapped = true;
   }
  }
  if (!swapped)
     break;
 }
}
```

Date.
Page No.

Ques 14) • Merge sort optimized for external sorting
• Divide and conquer approach.
• Requires small portion of data to fit memory.

External Sorting	Internal Sorting
On virtual memory	On RAM.
Eg: Quick, merge Sort	Eg: bubble, selection, inserting. sort.