DAA

Date: / / /

Tutorial-3

~~Week-3~~

Ans 1

```
search(arr, n, k)
    i = 0, j = n-1
    while i <= j
    { if a[i] < k
        i++
      else if a[i] > k
        j--
      else
        return 'True'
    }
    return False
```

Ans 2 ~~Iterative~~ insertion ~~sort~~

```
Insertion sort()
    for i = 1 to n
        key = a[i]
        j = i-1
        while j >= 0 and a[j] > key
            a[j+1] = a[j]
            j = j-1
        End-while
        a[j+1] = key
    End for
```

Date : / / /

```
void recursive_insertion_sort (int a[], int n)
{   if (n<=1)
        return
    recursive_insertion_sort (a, n-1)
    int val = a[n-1]
    int pos = n-2
    while (pos>=0 && a[pos]>val) {
        a[pos+1] = a[pos]
        pos = pos -1 }
    a[pos+1] = val
}
```

An online algorithm is one that can process its input piece-by-piece in a serial fashion i.e in the order that the input is fed to the algorithm, without having the entire input available from beginning.

Insertion sort - online algo
Selection sort - offline algo
heapsort - offline algo
quicksort - offline algo

DATE .....

PAGE No .....

Ans 3

Algorithm	Time Complexity 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	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$
Count sort	$O(n+k)$	$O(n+k)$	$O(n+k)$

Ans 4

	Inplace	Stable	Online sorting
Bubble sort	Yes	Yes	No
Selection sort	Yes	No	Yes
Insertion sort	Yes	Yes	No
Quick sort	Yes	No	No
Merge sort	No	Yes	No
Heap sort	Yes	No	No

TEACHER'S SIGNATURE .....

DATE .....

PAGE No .....

Ans:- Recursive binary search

```
int binarySearch (int a[], int r, int x)
    if (r >= l)
        int mid = (l+r) / 2

        if (a[mid] == x) then
            return mid
        End if

        if (a[mid] > x) then
            return binarySearch (a, l, mid-1, x
        End if

        return binarySearch (a, mid+1, r, x)
    End if
    return -1
End function
```

Iterative binarySearch

```
int binarySearch (int a[], int l, int r, int
{
    while (l <= r)
    {
        int m = (l+r) / 2
        if (a[m] == x) then
            return m
        End if
```

TEACHER'S SIGNATURE .....

DATE ..... PAGE No .....

```
if (a[m] < x) then
    l = m+1
else
    r = m-1
End while
return -1
End function
```

	Time Complexity	Space complexity
Recursive	O(log(n))	O(log(n))
Iterative	O(log(n))	O(1)

Ans 6 Recurrence relation for Binary search is

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

Ans 7 $A[i] + A[j] = K$

We can find K by first sorting the array using heap sort which will take time complexity $O(n \log n)$ then we can find K using Two-pointer technique which will take time complexity $O(n)$

Total Time complexity $O(n \log n)$

TEACHER'S SIGNATURE .....

```
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
void heapsort (int arr[], int n)
{
    for(int i=n/2-1 ; i>=0 ; i--)
        heapify (arr, n, i);

    for(int i=n-1 ; i>=0 ; i--)
    {
        swap (&arr[0], &arr[i]);
        heapify (arr, i, 0);
    }
}

void heapify ( int arr[], int n, int i)
{
    int largest = i;
    int left = 2*i + 1;
    int right = 2*i + 2;
```

Date : / / /

```
if (left < n && arr[left] > arr[largest])
    largest = left;

if (right < n && arr[right] > arr[largest])
    largest = right;

if (largest != i)
{
    swap(&arr[i], &arr[largest]);
    heapify(arr, n, largest);
}
}

int main()
{
    int arr[MAX]; k;
    int n;
    scanf("%d", &k);
    printf("Enter the size of array:");
    scanf("%d", &n);
    for (int i=0; i<n; i++)
        scanf("%d", &arr[i]);
    heapsort(arr, n);
int l = 0;
int r = n - 1;
```

Date : / / /

```
while (l < r) {
    if (arr[l] + arr[r] == sum.k)
        printf("A[%d] + A[%d] = %d",
                l, r, k);
    else if (arr[l] + arr[k] < sum.k)
        l++;
    else
        r--;
}
return 0;
}
```

Ans 8 Most commonly used sorting algorithm is quick sort. It doesn't make any assumptions about the type of data. It can be done without taking extra memory. It is the fastest general purpose sort.

Ans 9 arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4,

Inversions in given array -

(7, 1), (7, 4), (7, 5), (21, 8), (21, 10), (21, 1), (
(21, 6), (21, 4), (21, 5), (31, 8), (31, 10), (31,
(31, 20), (31, 6), (31, 4), (31, 5), (8, 1), (8, 6), (8, 4)

Date: / / /

(10, 1), (10, 6), (10, 4), (10, 5), (20, 6), (20, 4), (20, 5), (6, 4), (6, 5)

Total inversions = 30

Ans 10 ~~Quick~~ Quick sort

Best case

When the partition process always picks the middle element as pivot.

$$T(n) = 2T(n/2) + \Theta(n)$$

Worst case

Worst case occurs when the partition process always picks greatest or smallest element as pivot and the array is already sorted in ascending or descending order.

$$T(n) = T(n-1) + \Theta(n)$$

Ans 11 Quick sort

Worst case

$$T(n) = T(n-1) + \Theta(n)$$

Best case

$$T(n) = 2T(n/2) + \Theta(n)$$

Date: / / /

Ans13
```
void bubblesort (int arr[], int n)
{
    int i, j;
    for (i=0; i<n-1; i++)
        for (j=0; j<n-i-1; j++)
            if (arr[j] > arr[j+1])
            { int temp;
              temp = arr[j];
              arr[j] = arr[j+1];
              arr[j+1] = temp;
            }
}
```

Ans14 We will be using merge sort algorithm. because we can divide the 4 gb data into 4 packets of 1 gb and sort them separately and combine them latter

Internal sorting - all the data to sort is stored in memory at all times while sorting is in progress

External sorting - all the data is stored outside memory and only loaded into memory in small chunks.