DAA assignment

Name - Harsh Gupta

section - DS1

Roll no - 53

Ans-1 :-

```
int linearsearch (vector <int> arr, int key)
{
    int n = arr.size();
    for (i=0 to n-1)
    {
        if (arr[i] == key)
        {
            return i;
        }
    }
    return -1;
}
```

Ans-2 :-

Iterative :-

```
void insertionSort ( vector <int> & arr)
{
    int n = arr.size() , i, j, tmp;
    for (i = 1 to n)
    {
        tmp = arr[i];
        j = i-1;
        while ( j>=0 and arr[j] > tmp)
        {
            arr[j+1] = arr[j];
            j = j-1;
        }
        arr[j] = tmp;
    }
}
```

Recursive ⇒

```
void insertion_sort(vector<int> &arr, int n)
{
    if(n<=1)
        return;
    insertion_sort(arr, n-1);
    int tmp = arr[n-1];
    j = n-2;
    while( j>=0 and arr[j] > tmp)
    {
        arr[j+1] = arr[j];
        j= j-1;
    }
    arr[j+1] = tmp;
}
```

Insertion sort is called an online sorting algorithm because it can sort an array as it receives new element without needing the entire list to be presented beforehand.

Among other sorting algorithms merge sort and bubble can also be adapted to work in online manner.

Q-3

	Time complexity			Space complexity
	best	worst	avg	
bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
quick sort	$O(n \log n)$	$O(n^2)$	$O(n \log n)$	$O(\log n)$
merge sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
Heap sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$

Q-4

	Inplace	stable	online
selection	✓	×	×
insertion	✓	✓	✓
bubble	✓	✓	×
quick	✓	×	×
merge	×	✓	×
heap	✓	×	×

Ans-5 iterative ⇒

```
int binarySearch(vector<int>& arr, int key)
{
    int s = 0;
    int e = arr.size() - 1;
    int mid = s + (e-s)/2;
    while (s <= e)
    {
        if (arr[mid] == key)
            return mid;
        else if (arr[mid] > key)
            e = mid - 1;
        else
            s = mid + 1;
        mid = s + (e-s)/2;
    }
    return -1;
}
```

recursive :-

```
int binarySearch(vector<int> arr, int s, int e, int key)
{
    if (e >= s)
    {
        int mid = s + (e-s)/2;
        if (arr[mid] == key)
            return mid;
        else if (arr[mid] > key)
            return binarySearch(arr, s, mid-1, key);
        else
            return binarySearch(arr, mid+1, e, key);
        return -1;
    }
}
```

```
        return -1;
    }
```

Q-6 write recurrence relation for binary recursive search.

Ans:- $T(n) = T(n/2) + 1$

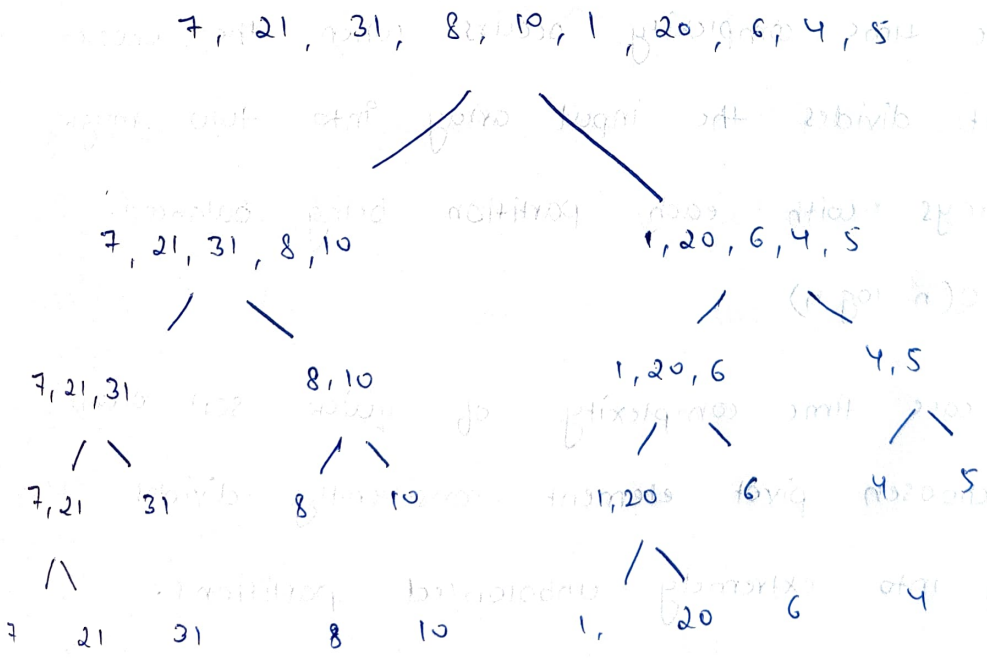
Ans - 7 :-

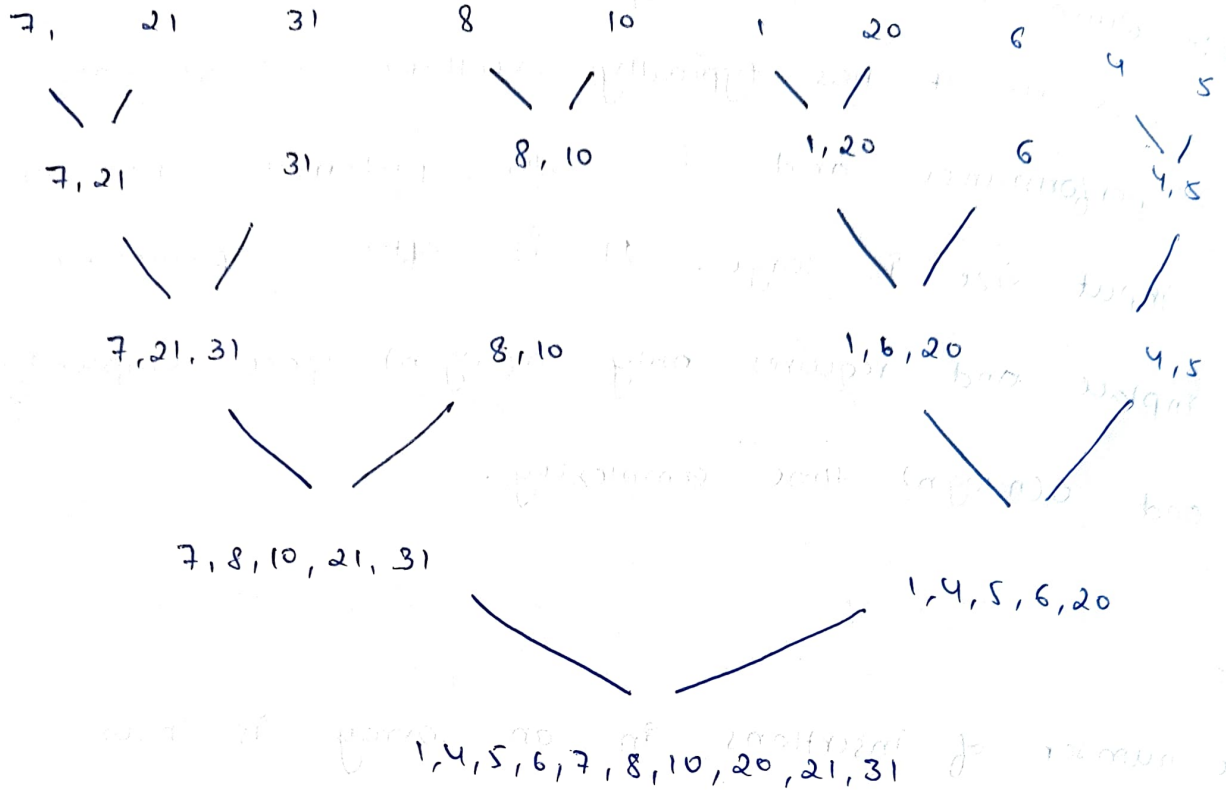
```
vector<int> findindex(vector<int> arr, int k)
{
    unordered_map<int, int> m;
    vector<int> indexes;
    for(int i=0; i< nums.size(); i++)
    {
        int diff = k - nums[i];
        if(m.find(diff) != m.end())
        {
            indexes.push_back(i);
            indexes.push_back(m[diff]);
        }
        m[arr[i]] = i;
    }
    return indexes;
}
```

Ans-8:- Quick sort is generally considered best for practical uses as it has typically excellent average case performance and is often preferred when the input size is large. It is often implemented inplace and requires only $O(\log n)$ space complexity and $O(n \log n)$ time complexity.

Ans-9:-

The number of insertions in an array is how unsorted the array is. An inversion occurs when two elements in an array are out of order relative to each other.





total inversions - 31

Ans - 10 :-

The best case time complexity occurs when the choosen pivot element divides the input array into two roughly sized subarrays with each partition being balanced.

T.C. ⇒ $O(n \log n)$

The worst case time complexity of quick sort occurs when the choosen pivot element consistently divides the input array into extremely unbalanced partitions. (when input array is already sorted in ascending or descending order).

T.C. ⇒ $O(n^2)$

Ans-11 :- merge sort :-

- Best case recurrence relation ⇒ $T(n) = 2T(n/2) + O(n)$
- worst case recurrence relation ⇒ $T(n) = 2T(n/2) + O(n)$

Quick sort :-

- Best case recurrence relation ⇒ $T(n) = 2T(n/2) + O(n)$
- worst case recurrence relation ⇒ $T(n) = T(n-1) + O(n)$

similarities :-

Both the algorithms have best case time complexity $O(n \log n)$ when the input is well behaved.

Both the algorithms use divide and conquer approach.

differences :- merge sort → stable

quick sort → not stable

merge sort typically requires $O(n)$ additional space for merging while quick sort is implemented in place requiring only $O(\log n)$ additional space.

Ans - 12

```
void selectionSort(vector<int> &arr)
{
    int n = arr.size();
    for(int i=0; i<n-1; i++)
    {
        int min = i;
        for(int j=i+1; j<n; j++)
        {
            if(arr[j] < arr[min])
                min = j;
        }
        int minval = arr[min];
        while(min > i)
        {
            arr[min] = arr[min-1];
            min--;
        }
        arr[i] = minval;
    }
}
```

Ans - 13

```
void bubbleSort(vector<int> arr)
{
    int n = arr.size();
    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 == false)
        {
            break;
        }
    }
}
```

Q.14 your computer has a RAM of 2 GB and you are given an array of 4 GB for sorting. which algorithm you are going to use for this purpose and why? Also explain the concept of external and internal sorting.

→ As per the given conditions, we cannot perform the sorting entirely in memory using traditional sorting algorithms. Instead we could need an external sorting algorithm. One such algorithm used is external merge sort.

External merge sort is an algorithm designed to handle datasets that cannot fit entirely in the memory. It works by dividing the dataset into smaller chunks that can fit in memory, sorting these smaller chunks internally and then merging these sorted chunks together to produce the final sorted output.

- Internal sorting algorithms are designed to sort datasets that can fit entirely in memory.
- External sorting algorithms are designed to sort datasets that are too large to fit entirely in memory.