DAA - TUTORIAL-3

PURU SINGH
CS7-SPL-1
Roll no: 60

Ans 1

```
int linearsearch(int a[], int n, int key)
{
    if( abs(a[0]-key) > abs(a[n-1]-key)){
        for(i=n-1; i>0; i--)
            if(a[i]==key)
                return i;
    }
    else
        for( i=0; i<n; i++)
            if( a[i]==key)
                return i;
}
```

Ans 2

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

```
while(j>-1 && a[j]>x)
{
    a[j+1] = a[j];
    j--;
}
a[j+1]=x;
}
}
```

Recursive =>

```
insertionSort(int a[], int n)
{
    if(n<=1)
        return;
    insertionSort(a, n-1);
    int x = a[n-1];
    int j = n-2;
    while(j>=0 && a[j]>x){
        a[j+1]=a[j];
        j--;
    }
    a[j+1] = x;
}
```

⇒ Insertion sort is called online sorting because it only considers

only one input per iteration and produces a partial solution without considering future elements whereas other sorting algorithms process the whole problem data altogether from the beg. and is required to output an answer which solve the problem at hand.

Ans 3

SORTING	BEST	WORST	AVERAGE
Bubble sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$
Selection sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$
Insertion sort	$\Omega(n^2)$	$\Theta(n^2)$	$O(n^2)$
Quick sort	$\Omega(n\log n)$	$\Theta(n^2)$	$O(n\log n)$
Merge sort	$\Omega(n\log n)$	$\Theta(n\log n)$	$O(n\log n)$
Count sort	$\Omega(n+m)$	$\Theta(n+m)$	$O(n+m)$
Heap sort	$\Omega(n\log n)$	$\Theta(n\log n)$	$O(n\log n)$

m = range

Ans 4

SORTING	Inplace	Stable	Online
Bubble sort	✓	✓	×
Selection sort	✓	×	×
Insertion sort	✓	✓	✓
Quick sort	✓	×	×
Merge sort	×	✓	×
Count sort	×	✓	×
Heap sort	✓	×	✓

Ans 5 Recursive =>

```
int binarySearch(int a[], int l, int r, int x)
{
        int mid;
        while(l<=r){
            mid= (l+r)/2;
            if(x> a[mid])
                return binarysearch(a, mid+1, r, x);
            else if (a[mid]> x)
                return binarysearch(a, l, mid-1, x);
            else
                return mid;
        }
}
```

Iterative =>

```
int binarySearch(int a[], int n, int x)
{
        int l=0, r=n-1, mid;
        while (l<=r)
        {
            mid= (l+r)/2;
            if (x< a[mid])
                r= mid-1;
            else if (a[mid]< x)
                l= mid+1;
            else
```

```
        return mid;
    }
}
```

Linear search → Time complexity = O(n)

Space complexity = O(1)

Binary search → Time complexity = O(log n)

Space complexity = O(1)

Ans 6 $T(n) = T(n/2) + 1$

Ans 7

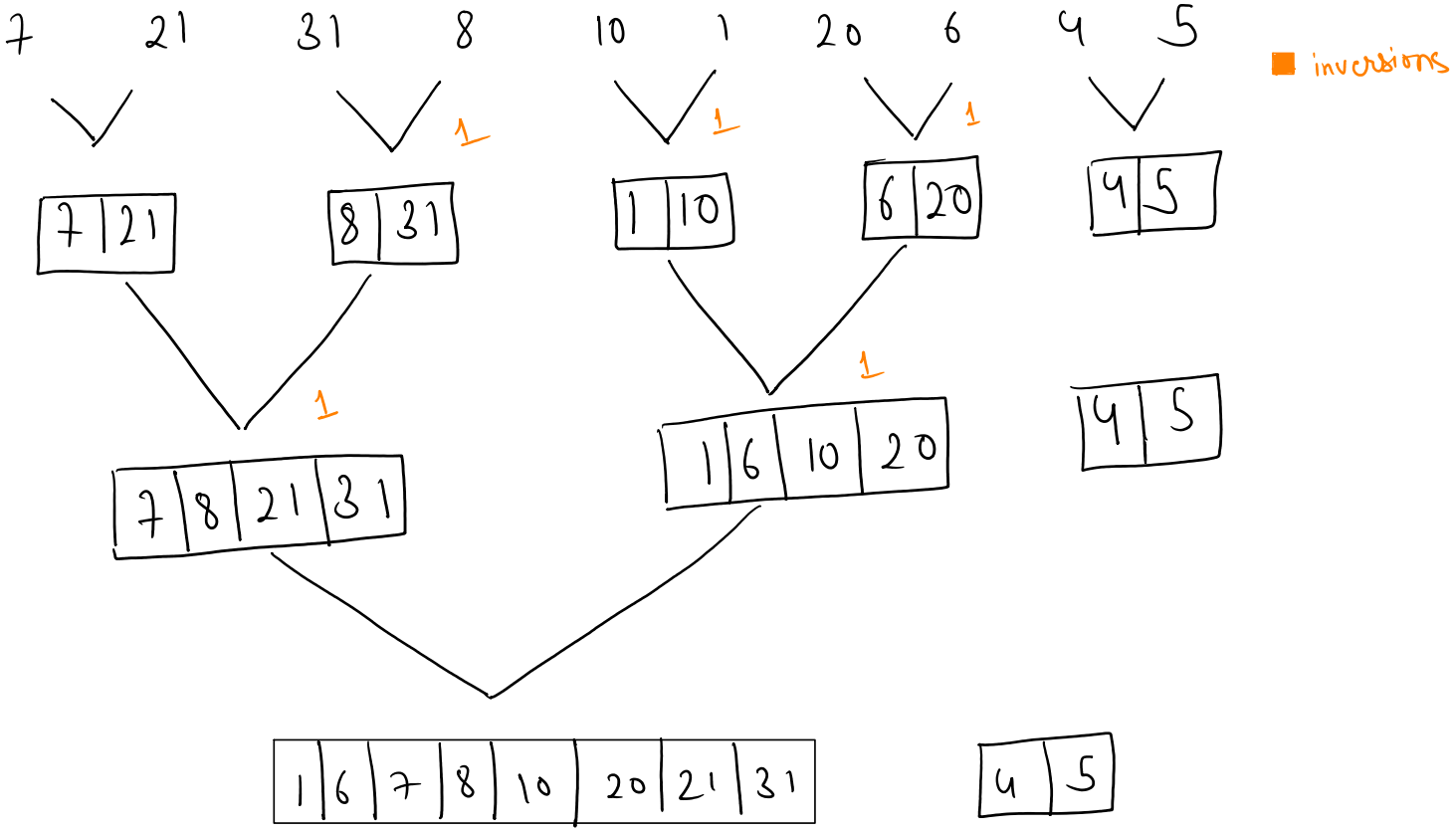
```
findIndex(int a[], int n, int k)
{
    int i=0, j=1;
    while( i<n && j<n)
    {
        if( i!=j && (a[j] - a[i]==k || a[i]
            - a[j] ==k))
                cout<<i<<j;
        else if( a[j] - a[i] < k)
                j++;
        else
                i++;
```

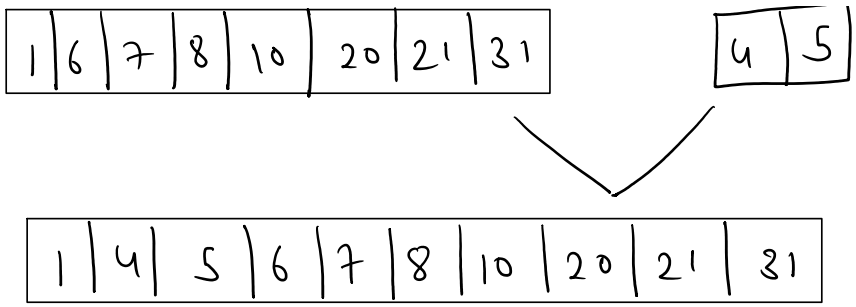
}

}

Ans 8 Quick sort is one of the most efficient sorting algorithms which makes it one of the most used as well. It is faster as compared to other sorting algorithms. Also, its time complexity is O(nlogn). But in case of a larger array Merge sort is preffered.

Ans 9 Inversion in an array basically define how far or close an array is from being sorted. If an array is already sorted, then inversion count = 0 ; If an array is reverse sorted then inversion count = maximum.





Ans 10 Best case :

If partitioning element is in middle

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

Worst case :

If pivot is at extreme position and array is already sorted in increasing or decreasing order

Time complexity : $O(n^2)$

Ans 11 Quick sort ⇒

Best : $T(n) = 2T(n/2) + n$

Worst : $T(n) = T(n-1) + n$

Merge Sort ⇒

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

In merge sort, the array is divided into 2 halves n times

$\therefore$ T.C = $O(n \log n)$

In quick sort, the array is divided into any ratio depending on the position of pivot element.

$\therefore$ T.C varies from $O(n^2)$ to $O(n \log n)$

Ans 12 Selection sort works by inserting the minimum element at its correct position by swapping it with the element in the position of this minimum element. This is what makes it unstable.

Code =>

```
void stableSelectionSort(int a[], int n)
{
    for(int i=0; i<n-1; i++)
    {
        int min=i;
        for(int j=i+1; j<n; j++)
            if(a[min] > a[j])
                min=j;

        int key = a[min];
        while(min>i)
        {
            a[min] = a[min-i];
            min--;
        }
        a[i] = key;
    }
}
```

Ans 13

```
void fixedBubbleSort(int a[], int n)
{
    for(int i=0; i<n; i++)
    {
        int swaps=0;
```

```
            for( int j=0 ; j<n ; j++){
                if(a[i] < a[j]){
                    int temp = a[i];
                    a[i]=a[j];
                    a[j]=temp;
                    swaps++;
                }
            }
            if(swaps==0)
                break;
        }
    }
```

Ans 13 In such case, external sorting algorithms such as k-way merge sort is used that can handle large amount of data which can't fit in main memory at once. Parts of array reside in RAM during the execution whereas in internal sorting algorithms the array to be sorted is entirely placed into the RAM (eg. Bubble, etc)