Page No.
Date : / /

Assignment 2

Q.1) bool linearsearch (~~arr~~ int arr, int n, int key)

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

Q.2) Iterative insertion sort

```
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 = j-1;
    }
    a[j+1] = t;
}
```

recursive insertion sort

```
void sort(arr, 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 next elements as it receives~~

* 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

Page No.
Date : / /

Q.3)

Sorting tech.	complexity best	avg	worst
bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$
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	$O(d*(n+k))$	$O(d*(n+k))$	$O(d*(n+k))$

Q.4)

~~bubble~~	Inplace	stable	online sorting
bubble	✓		
Selection	✓		
Insertion	✓	✓	✓ ✱
Count		✓	
Quick	✓		
Merge		✓	
heap	✓		
Radix		✓	

Q.5) Recursive binary Search

```
T(n)  int Search(arr, target, low, high)
      {
          if(low > high)
              return -1; // element not found
          mid = (low + high) / 2;
          if (arr[mid] == target)
              return mid;
          else if (arr[mid] < target)
T(n/2)        return search(arr, target, mid+1, high);
          else
T(n/2)        return search(arr, target, low, mid-1)
      }
```

⇒ Iterative binary Search

```
int Search(arr, target)
{
    low = 0
    high = length(arr) - 1;
    while(low <= high)
    {
        mid = (low + high) / 2
        if (arr[mid] == target)
            return mid
        else if (arr[mid] < target)
            low = mid + 1
        else
            high = mid - 1
    }
    return -1
}
```

	Recursive	Iterative
Linear Search	$O(n)$	$O(n)$
Binary Search	$O(\log n)$	

	Recursive		Iterative	
	$\Omega(n)$	$\Theta(n)$	$\Omega(n)$	$\Theta(n)$
Linear	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Binary	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(1)$

Q.6 (refer Q.5)

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

Q.7)

```
int findpairwithsumk(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)
}
```

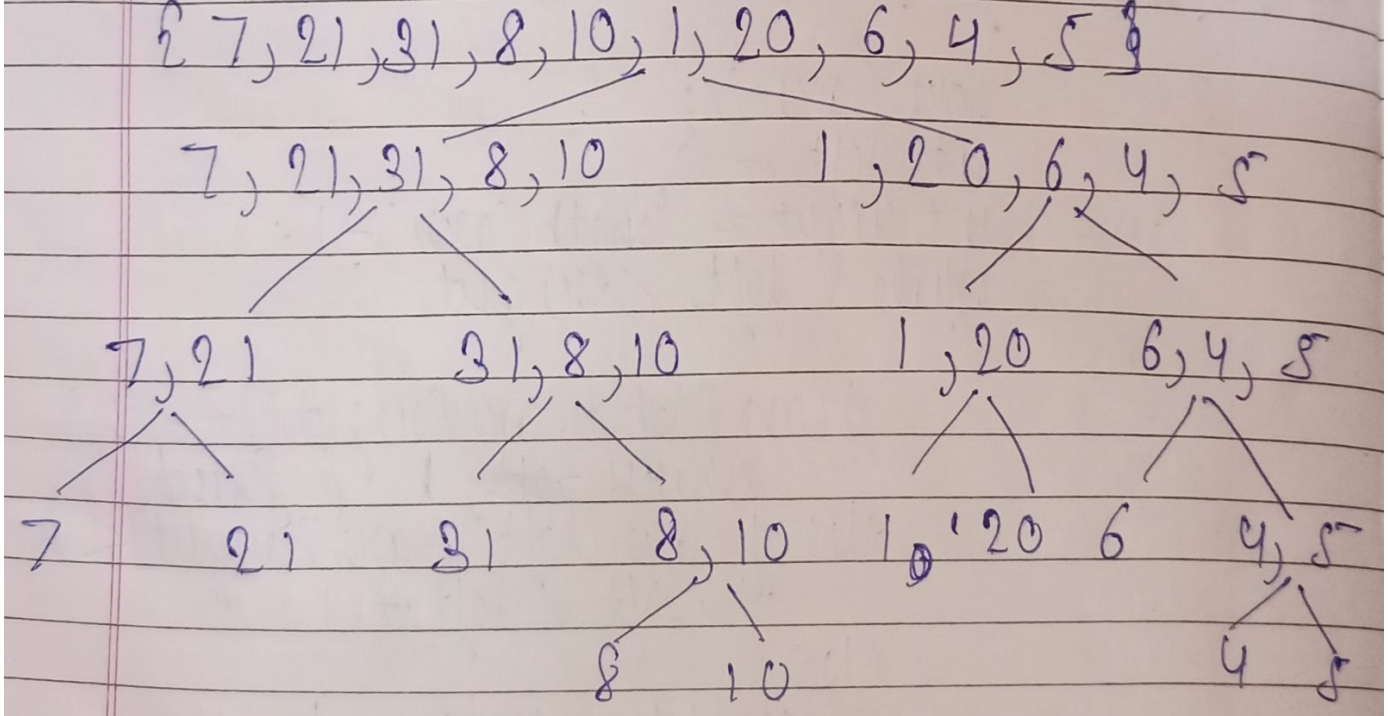
Q8) Quick Sort — Quick and fastest sorting algo especially for large dataset

Merge sort — TC O(n log n) in all cases
Divide conquer nature

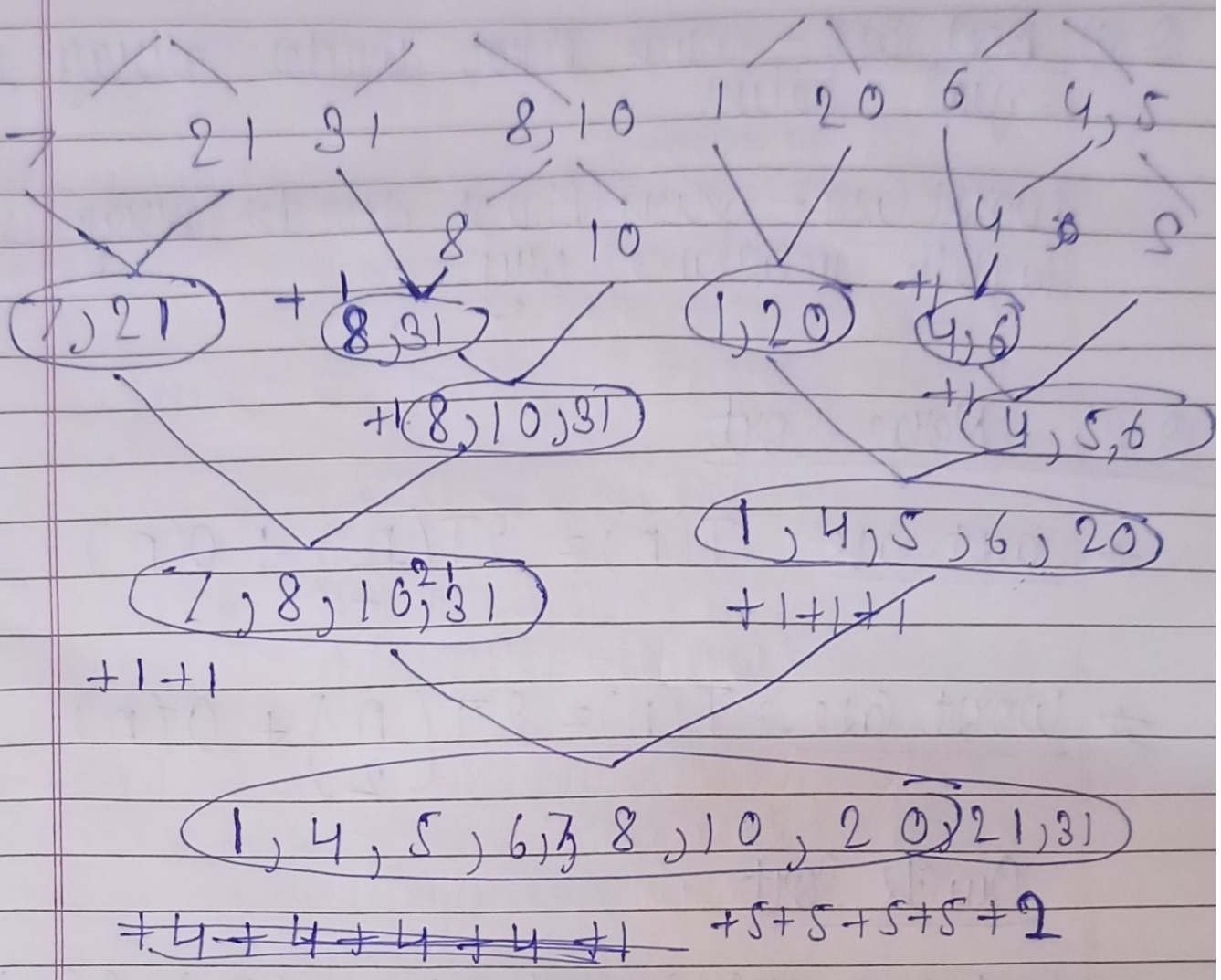
Heap sort — TC O(n log n)
doesn't require extra space

Insertion sort — Inplace, stable, online sorting

Q9) inversion : when smaller element is after larger element or vice versa



+1+1+1+1+1+1 +1+1+1



No. of iw = 1+1+1+1+1+1+1+1+1+4+4+4+4+

= 26

No. of iw = 1+1+1+1+1+1+1+1+1+5+5+5+5+2

= 31

Q 10) Best Case - When Pivot divides array in equal values

Worst Case - When Pivot divides array in highly imbalanced way

Q 11) Merge Sort

* best case : $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

* Worst Case : $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

Quick Sort

* best case : $T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + O(n)$

* wortase : $T(n) = T(n-1) + O(n)$

Similarities

* both have divide & conquer.

* $TC = O(n \log n)$

Difference

* In Quick sort TC varies

Q.12)

```
void selectionsort (arr, n)
{
    for(i=0; i<n; i++)
    {
        int minIndex = i;
        for(j=i+1; j<n; j++)
        {
            if (arr[j] < arr[minIndex])
                minIndex = j;
        }
        minValue = arr[minIndex];
        while(minIndex > i)
        {
            arr[minIndex] = arr[minIndex - 1]
            minIndex = minIndex - 1
        }
        arr[i] = minvalue;
    }
}
```

Q.13)

```
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;
    }
}
```

Q 14) Merge sort : optimized for external sorting

* Divide and conquer approach.
* Requires small portion of data to fit in memory.

PTO

External Sorting	Internal Sorting
☆ On virtual memory	☆ ON RAM
☆ ex- Quick, Merge Sort.	☆ ex- bubble, selection, insertion sort.