MUKUL RAWAT, CS1-92

Date: /

Tutorial-3

1.) linear search in sorted arr.

```
int idx = -1
for (i=0; i<n; i++)
    {
    if (arr[i] == search)
    idx = i; break;

    if (arr[i] > search)
    break;
    }
return idx;
```

2.) iterative insertion

```
int i, temp, j;
for (i = 1 to n)
    {
        temp = arr[i];
        j = i-1
while (j>0 && arr[j] temp)
    arr[j+1] = arr[j]; j--;

arr[j+1] = temp
}
```

Mukul Rawat

Date: / /

recursive :-

```
if (n <=1)
    return;

insertion_sort (arr, n-1)
last = arr[n-1]
j = n-2

while (j>0 & arr[j] > last)
    arr[j+1] = arr[j]; j--;

arr[j+1] = last
```

insertion sort is known as online sorting because values of whole array need not to known at one time while executing information one by one linearly.

Mau Kawab

Name: Mansi Rawat

Date: /

3) Name.	time	Space
Selection	B → O(n²) W → O(n²)	O(1)
Bubble	B → O(n²) W → O(n²)	O(1)
Insertion	B → O(n) W → O(n²)	O(1)
Merge	B, W → O(n lg n)	O(n)
Quick	B → O(n log n) W → O(n²)	O(n) → Recursion stack
Heap sort	B → O(n log n) W → O(n lg n)	O(1)
Count sort	B, W → O(n+k) K: max element.	O(K)

Heap sort space complexity is O(1)

because heap sort use tail -recursion (which uses constant space). ∴ NO stack call will cause O(log n) space.

Mani Rawat

Date: / /

4) Name	inplace	Stable	online.
Selection	✓		
merque		✓	
Quick	✓		
heap	✓		
Bubble	✓		
count.		✓	
insertion	✓	✓	✓

5) iterative Binary Search

```
l=0, high = n-1
while (low <=high)
{ int mid = (low+high)/2

  if (arr[mid] == element)
    return element;

  else if (arr[mid] > element)
    high = mid-1;

  else
    low = mid+1 }

return -1
```

Name Kawa

Date:

5.) Recursion Binary Search

```
if (l <= r)
{
    int mid = (l+r)/2

    if (arr[mid] == x)
        return mid

    else if (arr[mid] > x)
        binary search (arr, l, mid-1, x)

    else
        binary search (arr, mid+1, r, x)
    } return -1;
```

Time complexity :- Best O(1)
Worst O(log n)

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

7.) Algo :

under-set j = arr[0]

1.) run loop for i=1 to n

2) if (element [i] x - arr[i] != j.end())
    return arr[i], *j
    else
        j.insert(arr[i])

Mohd Zaweed

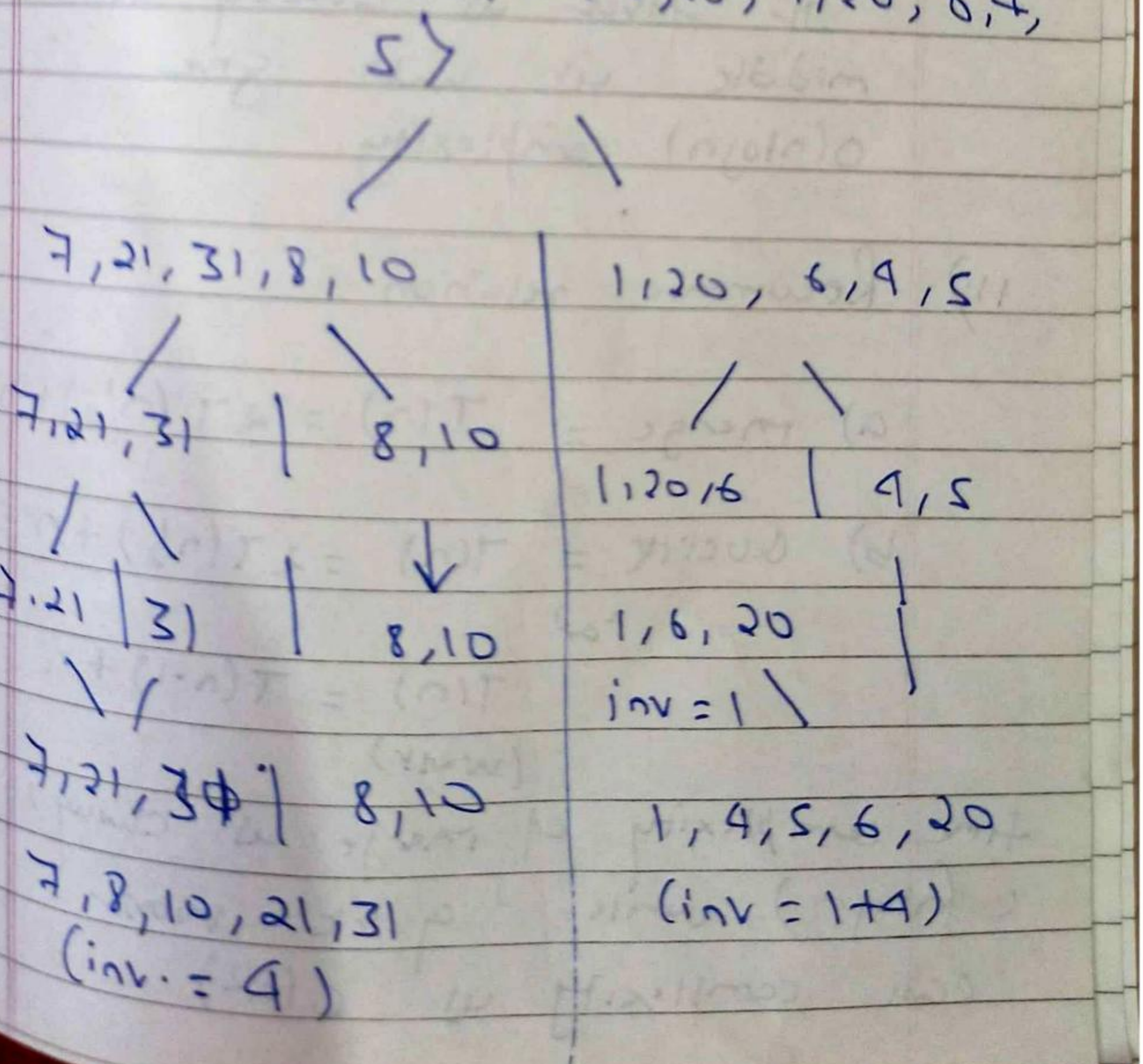
Date: / /

8) Quicksort is best for general purpose. Or stability is important & space is available then merge sort is best.

9) Inversion denote how far a array is from being sorted. If inversion is '0' array is sorted. Inversion is MAX if array is in descending order.

condition:- if $i < j$, $arr[i] > arr[j]$

arr[]: {7, 21, 31, 8, 10, 1, 20, 6, 4, 5}



Merge Sort

Date:

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

21

1, 4, 5, 6, 7, 8, 10, 20, 31

inv = 4 + 5 + 3 x 4 + 5 x 2

= 4 + 5 + 12 + 10

= 31 inversions.

iv) when pivot is always on extreme quick sort will give $O(n^2)$ time complexity. If pivot is always in middle it will give $O(n \log n)$ complexity.

ii) Recurrence relation

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

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

or

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

(worst)

time complexity of merge is always $O(n \log n)$ while quick worst case complexity is $O(n^2)$

Mani Rawat

Date: / /

12) just change condition of getting minimum element to get selection sort

or chge

```
if (arr[i] <= temp)
    min = i
```

to

```
if (arr[i] < temp)
    min = i;
```

13) bool is_sorted

```
for ( i = 0    i < n-1; i++ )
{
    is_sorted = ~~false~~ true

    for (int j = i+1;  j < n-1; i++ )
    {
        if ( arr[j] > arr[j-1])
            swap (arr[j], arr[j-1])
            is_sorted = false }

    if (is_sorted)
        break;
}
```