TUTORIAL-03

M T W T F S S | Page No.: | Date: | YOUVA

Q(1). Write linear search pseudocode to search an element in a sorted array with minimum comparisons.

Ans:-
```
int LinearSearch (int *arr, int n, int key)
{       for (i>=0 to n-1)
            if arr[i] = key.
                return i
        return -1
}
```

Q(2). Write pseudocode for iterative and recursive insertion sort. Insertion sort is called online storing why? What about other sorting algo. that has been discussed in lectures?

Ans:- iterative insertion sort.

```
void insertionsort (int arr[], int n)
    int i, temp, j;
    for i ← 1 to n
            temp ← arr[i]
            j ← i-1
        while (j>=0 AND arr[j]>temp)
                arr[j+1] ← arr[j]
                j ← j-1
        arr[j+1] ← temp
```

recursive insertion sort

```
Void insertionsort (int arr[], int n)
        if (n<=1)
            return
        insertionsort (arr, n-1)
        last = arr[n-1]
        j = n-2
        while (j>=0 AND arr[j]>last)
            arr[j+1] = arr[j]
```

j--
arr[j+1] = last

→ Insertion sort is called online sorting because it does not need to know anything about what values it will sort and the information is requested while the algorithm is running.

Q(3):- Complexity of all the sorting algorithms that has been discussed in lectures.

Ans:- (i) Selection Sort
→ time complexity = best = $O(n^2)$, worst = $O(n^2)$
→ space complexity = $O(1)$

(ii) Insertion Sort
→ time complexity = best = $O(n)$, worst = $O(n^2)$
→ space complexity = $O(1)$

(iii) Merge Sort
→ time complexity = best = $O(n\log n)$, worst = $O(n\log n)$
→ space complexity = $O(n)$

(iv) Quick Sort
→ time complexity = best = $O(n\log n)$, worst = $O(n^2)$
→ space complexity = $O(n)$

(v) Heap Sort
→ time complexity = best = $O(n\log n)$, worst = $O(n\log n)$
→ space complexity = $O(1)$

(vi) Bubble Sort
→ time complexity = best = $O(n^2)$, worst = $O(n^2)$
→ space complexity = $O(1)$

Q(4) Divide all sorting algorithms into inplace/ stable / online sorting.

Sorting	inplace	stable	Online
selection	✓		
insertion	✓	✓	✓
merge		✓	
quick	✓		
heap	✓		
bubble	✓	✓	

Q5:- Write recursive / iterative pseudocode for binary search. What is the Time & Space complexity of Linear and Binary Search.

Iterative binary Search

```
int binarysearch (int arr[], int l, int r, int x)
        while (l <= r)
                int m ← (l+r)/2
        if (arr[m] = x)
                return m;
        if (arr[m] < x)
                l ← m+1;
        else
                r ← m-1;
        return -1;
```

Time complexity -

Best case = $O(1)$

Average case = $O(\log_2 n)$

Worst case = $O(\log n)$

Recursive Binary Search

```
int binarySearch (int arr[], int l, int r, int x)
        if (r >= 1)
            int mid ← (l+r)/2
        if ( arr[mid] = x)
            return mid
        else if (arr[mid] > x)
            return binarysearch ( arr,
                l , mid - 1 , x)
        else
            return binary search ( arr,
                mid + 1 , r , x)
        return -1
```

Time Complexity

Best case = O(1)

Worst case = O(log n)

Average case = O(log n)

Q(6) Write recurrence relation for binary recursive search.

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

Q(7) Find two indexes such that $A[i] + A[j] = K$ in minimum time complexity

Ans:- $A[i] + A[j] = K$

Q(8) Which sorting is best for practical uses? Explain

Ans:- Quick sort is the fastest general-purpose sort. In most practical situations, quicksort is the method of choice. If stability is important

space is available, merge sort might be best.

Q 9:- What do you mean by no. of inversions in an array? Count no. of inversions in Array arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5} using merge sort?

Ans:- Inversion count for any array indicates: how far (or close) the array is from being sorted. If the array is already sorted, then the inversion count is 0, but if array is sorted in the reverse order, the inversion count is maximum.
arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5}

```
#include <bits/stdc++.h>
using namespace std;
int _mergesort(int arr[], int temp[], int
                    left, int right);
int merge(int arr[], int temp[], int left,
                int mid, int right);
int mergesort( int arr[], int array_size)
{
        int temp[array_size];
        return _mergesort(arr, temp, 0, array_
                                size - 1);
}
int _mergesort(int arr[], int temp[], int left,
                int right)
{       int mid, inv_count = 0;
        if (right > left)
        {
                mid = (right + left)/2;
```

```
    inv_count += _mergesort(arr, temp, left, mid);
    inv_count += _mergesort(arr, temp, mid+1, right);
    inv_count += merge(arr, temp, left, mid+1, right);
  }
  return inv_count;
}
int merge(int arr[], int temp[], int left,
          int mid, int right)
{   int i, j, k;
    int inv_count = 0;
    i = left;
    j = mid;
    k = left;
    while((i <= mid-1) && (j <= right))
    {   if(arr[i] <= arr[j])
            temp[k++] = arr[i++];
        else
        {
            temp[k++] = arr[j++];
            inv_count = inv_count + (mid-i);
        }
    }
    while(i <= mid-1)
    {   temp[k++] = arr[i++]; }
    while(j <= right)
        temp[k++] = arr[j++];
    For(i = left; i <= right; i++)
    {   arr[i] = temp[i];
        return inv_count;
}
int main()
{
    int arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5};
```

```
int n = sizeof(arr) / sizeof(arr[0]);
int ans = mergesort(arr, n);
cout << " No. of inversion are " << ans;
return 0;
}
```

Q (10):- In which cases Quick sort will give the best and worst case time complexity?

Ans:- The worst case time complexity of quicksort is $O(n^2)$. The worst case occurs when the picked pivot is always an extreme (smallest or largest) element. This happen when a input array is sorted or reverse sorted and either first or last element is picked as pivot.

→ The best case of quick sort is when we will select pivot as a mean element.

Q (11):- Write Recurrence Relation of Merge and Quick sort in best and worst case? What are the similarities and differences between complexities of two algo and why?

Ans:- Recurrence Relation

(a) Merge sort ⇒ $T(n) = 2T(n/2) + n$

(b) Quick sort ⇒ $T(n) = 2T(n/2) + n$

→ Merge sort is more efficient and works faster than quick sort in case of larger array size or datasets.

→ Worst case complexity for quick sort is $O(n^2)$ whereas $O(n \log n)$ for merge sort.

M T W T F S S

Page No.: YOUVA

Date:

Q(12) Selection Sort is not stable by default but can you write a version of stable selection sort.

Ans:- stable selection sort

```
#include <bits/stdc++.h>
using namespace std;
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-1];
                    min --;
            }
            a[i] = key;
    }
}
int main ()
{       int a[] = { 4, 5, 3, 2, 4, 1};
        int n = sizeof (a) / sizeof (a[0]);
        stableselectionsort (a, n);
        for ( int i=0; i<n ; i++)
                cout << a[i] << "  ";
        cout << endl;
        return 0;
}
```

M T W T F S S

Page No.: YOUVA

Date:

Q13. 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 concept of External and Internal sorting.

Ans:- The easiest way to do this is to use external sorting. We divide our source file into temporary files of size equal to the size of RAM & first sort these files.

- External Sorting - if the input data is such that it cannot adjusted in the memory intirely at once it needs to be sorted in a hard disk, floppy disk or any other storage device. This is called external sorting.

- Internal Sorting - if the input data is such that it can adjusted in the main memory at once, it is called internal sorting.

Q14. Bubble sort scans whole array even when array is sorted. Can you modify the bubble sort so that it doesn't scan the whole array once it is stored.

Ans:- A better version of bubble sort, known as m-bubble sort, includes a flag that is set if a exchange is made after an entire pass over. If no exchange is made after then it should be called the array is already order because no 2 elements need to be switched.

```
void bubble (int arr[], int n)
{
    for (int i=0 ; i<n ; i++)
    {       swaps = 0;
        for( int j=0 ; j<n-i-j ; j++)
        {
            if (arr[j] > arr[j+1])
            {
                int t = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = t;
                swap++;
            }
        }
        if (swap==0)
            break;
    }
}
```