(DAA)

DESIGN AND ANALYSIS OF ALGORITHMS

ASSIGNMENT (Tutorial-3)

Submitted by :-- ARJUN AHALAWAT

Section :-- DS1 Class Roll no :--23

Q1) Write a linear search pseudocode to search an element in a sorted array with minimum comparisons.

⇒
```
int linearSearch(int A[], int n, int key)
{
    for(int i=0; i<n; i++)
    {
        if(A[i] == key)
            return 1;
    }
    return 0;
}
```

Q2) Write pseudo code for iterative and recursive insertion sort. Insertion sort is called online sorting. Why? What about other sorting algorithms that has been discussed in lectures?

⇒
```
for (i=1; i<n; i++)
{
    int t = A[i];
    j = i-1;
    while( j>=0 && A[j]>t)
    {
        A[j+1] = A[j];
```

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

Insertion sort is called an online sorting algo because it processes its input in a serial manner, that is, the order in that/which input is fed to the algorithm, without having the entire input available from the beginning.

Other ~~ord~~ sorting algorithms discussed, for example selection sort, does not fulfills the criteria that is discussed above for an online sort and hence can be termed that selection sort is not an online sorting algorithms.

Q3) Complexity of all the sorting algorithms discussed in lectures.

⟹

Sorting Algo's	Time Complexity			Space Complexity
	Best	Avg	Worst	
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Count Sort	$O(n+k)$	$O(n+k)$	$\rightarrow O(n+k)$	$O(n+k)$
Quick Sort	$O(n\log n)$	$O(n\log n)$	$O(n^2)$	$O(\log n)$
Merge Sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(n)$
Heap Sort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$	$O(1)$

Q4) Divide all the sorting Algorithms into Inplace/Stable/Online sortings.

=>>

Inplace	Stable	Externals
Bubble Sort	Merge Sort	K-way Merge Sorts
Selection Sort	Bubble Sort	
Insertion Sort	Insertion Sort	
Heap Sort	Count Sort	

Q5) Write recursive/iterative pseudo code for Binary Search. What is the Time and Space Complexity of Linear and Binary Searches.

=>> Iterative:-

```
bool binarySearch(int A[], int l, int r, int Key)
{
    while (l<=r)
    {
        mid = l+(r-l)/2;
        if (A[mid]==key)
            return true;
        elseif (A[mid]<key)
            l = mid+1;
        else
            r = mid-1;
    }
    return false;
}
```

(P.T.O.).

Recursive :-

```
if (l > r)
{
    return 0;
}
int mid = l + (r - l)/2;
if (A[mid] == key)
    return 1;
else if (A[mid] < key)
    binarySearch (A, mid+1, r, key);
else
    binarySearch (A, l, mid-1, key);
```

Search	Space Complexity	Time Complexity
Linear	O(1)	Best = O(1) Avg = O(n) Worst = O(n)
Binary	O(1) - Iterative O(logn) - Recursive	Best = O(1) Avg = O(logn) Worst = O(logn)

Q6) Write the recurrence relation for binary recursive search.

⇒

```
bool BS (int A[], int l, int r, int key) ⟶ T(n)
{
    if (l < r)
```

```
    {
        int mid = l+(r-l)/2;
        if (A[mid]==key)                    ---> O(1)
                return true;
        else if (A[mid]<key)
                BS (A, mid+1, r, key);      } T(n/2)
        BS (A, l, mid-1, key);
    }
}
```

Recurrence Relation $\Rightarrow\Rightarrow$ $T(n) = T(n/2) + O(1)$

Q7) Find two indexes such that $A[i] + A[j] = K$ in minimum time complexity.

$\Rightarrow\Rightarrow$

```
pair<int, int> Indexes (vector<int>&A, int k)
{
    map<int, int> Indexmap;
    for (int i=0; i<A.size(); ++i)
    {
        int value= k- A[i];
        if (Indexmap.find(value) != Indexmap.end())
        {
            return {Indexmap[value], i};
        }
        else
        {
            Indexmap[A[i]] = i;
        }
    }
    return {-1, -1};
}.
```

Q8) Which sorting is best for practical uses. Explain.

⇒ Quick Sort is the best sorting algorithm as per the analysis as it uses one of the finest and effective approach of divide and conquer, and is one of the fastest sorting algorithms so far.

It picks an pivot element and partitions the list around it. After this, on pivot element basis recursive quick approach is applied for other two sublists and so on.

Q9) What do you mean by number of inversions in an array? Count the number of inversions in Array arr[] = { 7, 21, 31, 8, 10, 1, 20, 6, 4, 5 } using Merge Sort Algorithms.

⇒ Inversions in an array indicates that how far or close the array is from being sorted. For example, if an array is already sorted then inversions will be 0 and if the array is in reverse order, the inversions will be maximum.

arr = { 7, 21, 31, 8, 10, 1, 20, 6, 4, 5 }
In this, there will be around 22 no. of inversions.
for, examples:-.

arr[] = { 1, 20, 6, 4, 5 }
Inversions = (20, 6), (20, 4), (20, 5), (6, 4). (6, 5.).

Q10) In which cases Quick Sort will give the best and worst case time complexities?

⇒ The best case for Quick Sort is when the array is completely arranged in an random manner. Whereas, the worst case is when the array is already sorted that is when the array is in ascending or descending orders.

Q11) Write Recurrence Relation for Merge and Quick sorts in worst and best case? What are the similarities and differences between complexities of the two algorithms and why?

Ans: *Quick Sort :-.

① Best / General Case Recurrence Relation:-.

$$T(n) = 2T(n/2) + n$$
$$= n \log n$$
$$= O(n \log n)$$

② Worst Case Recurrence Relation:-.

$$T(n) = T(n-1) + 1 + n$$
$$= O(n^2)$$

*Merge Sort :-.

$$T(n) = 2T(n/2) + n$$
$$= n \log n$$
$$= O(n \log n).$$

* Similarities are:-

Quick and Merge Sorts both have sample time complexity for best and general/average cases, i.e. $O(n \log n)$

* Difference is:-

The worst case complexity of Quick is $O(n^2)$ and that of Merge Sort is $O(n \log n)$.

Q12). Selection Sort is not Stable by default but can you write a version for Stable Selection Sort.

⇒ Yes, Selection Sort can be made stable if it can be implemented without swapping of minimum element and some other version is used for placing the minimum elements.

```
void stableSS(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];
        for(int k = min; k > i; k--)
        {
            A[k] = A[k-1];
        }
        A[i] = key;
    }
}
```

Q13) Bubble sort scans the whole array even when array is sorted. Can you modify the bubble sort so that it doesn't scans the whole array once it is sorted.

⇒ Yes, it can be optimised by stopping the algorithm if the inner loop didn't cause any swaps.

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

Q14) Your computer has a RAM (Physical memory) of 2GB and you are given an array of 4GB for sorting. Which algorithms you are going to use for this purpose and Why? Also explain the concepts of External and Internal Sortings.

⇒ For this purpose K-Way Merge Sort will be best suited approach for implementation as it is an External Sorting Algorithm.

External Sorting Algorithm:--

When an external memory space is used for storing the chunks (small parts) divided from the large size files for the purpose of sorting, such algorithms are called as External Sorting Algorithms.
examples:-- K-Way Merge Sort.

Internal Sorting Algorithms:--

Sorting Algorithms that exclusively uses only the main memory leading to high speed random access to all memory are called as the Internal Sorting Algorithms.
examples:-- Quick Sort.

Submitted by:-- ARJUN AHALAWAT
Section :-- DS1 Class Roll no:-- 23