SHRADDHA BAHUGUNA
CST SPL-1
30

Date
Page No

# DESIGN AND ANALYSIS OF ALGORITHMS

## TUTORIAL - 3

1 Write linear search pseudo code to search an element in a sorted array with minimum comparisons.

```
boolean linear_Search(a[], e, n)
  {
    i=0
    while (a[i]<e && i<n)
      {
        if a[i]=e
          return true
        else
        i=i+1
      }
    return false
  }
```

Shraddha

Date
Page No

2 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

ITERATIVE

```
void insertion_sort(a[], n)
 {
    for i= 1  to  n-1
          temp = a[i]
          j = i-1
          while(j>=0  && a[j] > temp)
                a[j+1] = a[j]
                j--
          a[j+1] = temp.

 }
```

RECURSIVE

```
void insertion_sort(arr[], n)
   {
      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
   }
```

Shraddha

Date
Page No.

Insertion sort is called an online sorting algorithm because insertion sort considers one input element per iteration and produces a partial solution without considering future elements

3 Complexity of all sorting algorithms that has been discussed in lectures.

| SORTING TECHNIQUE | BEST | AVERAGE | WORST |
|---|---|---|---|
| BUBBLE SORT | $O(n)$ | $O(n^2)$ | $O(n^2)$ |
| SELECTION SORT | $O(n^2)$ | $O(n^2)$ | $O(n^2)$ |
| INSERTION SORT | $O(n)$ | $O(n^2)$ | $O(n^2)$ |
| HEAP SORT | $O(n\log n)$ | $O(n\log n)$ | $O(n\log n)$ |
| QUICK SORT | $O(n\log n)$ | $O(n\log n)$ | $O(n^2)$ |
| MERGE SORT | $O(n\log n)$ | $O(n\log n)$ | $O(n\log n)$ |
| COUNT SORT | $O(n+k)$ | $O(n+k)$ | $O(n+k)$ |

4 Divide all the sorting algorithms into inplace / stable / online sorting.

| SORTING ALGORITHM | INPLACE | STABLE | ONLINE |
|---|---|---|---|
| BUBBLE SORT | ✓ | ✓ | X |
| SELECTION SORT | ✓ | X | X |
| INSERTION SORT | ✓ | ✓ | ✓ |
| QUICK SORT | ✓ | X | X |
| MERGE SORT | X | ✓ | X |
| HEAP SORT | ✓ | X | X |

Shraddha

Date:
Page No.

5 Write recursive/iterative pseudo code for binary search. What is the Time and Space Complexity of Linear and Binary Search (Recursive and Iterative)

ITERATIVE

```
int binarySearch(a[], x)
  {
     low = 0 , high = a.length -1
     while (low <= high)
          mid = (low + high)/2
          if x == a[mid]
               return mid
          else if (x < a[mid])
               high = mid -1
          else
               low = mid +1

     return -1
  }
```

RECURSIVE

```
int binarySearch(a[], low, high, x)
  {
     if low > high
          return -1
     mid = (low + high)/2
     if x = a[mid]
          return mid
     else if (x < a[mid])
          return binarySearch(A, low, min-1, x)
     else
          return binarySearch (A, mid+1, high, x)
  }
```

Shraddha

Date.
Page No.

## LINEAR SEARCH

Time Complexity → $O(n)$
Space Complexity → $O(1)$

## BINARY SEARCH

Time Complexity → $O(\log N)$
Space Complexity → $O(1)$

Shraddha

Date
Page No.

6 Write recurrence relation for binary recursive search.

Recurrence Relation → $T(n) = T(n/2) + 1$

DERIVATION

1st step $T(n) = T(n/2) + 1$ —①

put $n = n/2$ in ①

$T(n/2) = T(n/4) + 1$ —②

put eqⁿ ② in ①

$T(n) = T(n/4) + 1 + 1$ —③

put $n = n/4$ in eqⁿ ①

$T(n/4) = T(n/8) + 1$ —④

put ④ in ③

$T(n) = T(n/8) + 1 + 1 + 1$ —⑤

$T(n) = T\left(\frac{n}{2^k}\right) + (1+1+k)$

put $\frac{n}{2^k} = 1 \Rightarrow n = 2^k$ $\boxed{\log_2 n = k}$

$T(n) = T(1) + \log_2 n$

$= 1 + \log_2 n$

$\boxed{T(n) = O(\log_2 n)}$

Shraddha

Date
Page No

7 Find two indexes such that A[i] + A[j] = k in minimum time complexity.

```
vector<int> find(arr[], k, n)
 {
     vector <int> sol;
     for i=0 to n-1
         for j=0 to n
             if arr[i] + arr[j] = k
                 sol.pushback(i)
                 sol.pushback(j
                 return sol
```

8 Which sorting is best for practical uses? Explain

Quicksort is the fastest general purpose sort. In most practical situations quicksort is a method of choice. If stability is important and space is available, merge sort might be best. In some performance-critical applications, the focus maybe on just sorting numbers, so it is reasonable to avoid the costs of using references and sort primitive types instead

Shraddha

Date.
Page No.

9 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

Inversion count for an 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 the array is already sorted, then the inversion count is 0, but if the array is sorted in reverse order, the inversion count is the maximum.

Pair of inversions in array arr[] = (7,1) (7,6) (7,4) (7,5) (21,8) (21,10) (21,1) (21,20) (21,6) (21,4) (21,5) (31,8) (31,10) (31,1) (31,20) (31,6) (31,4) (31,5) (8,1) (8,6) (8,4) (8,5) (10,1) (10,6) (10,4) (10,5) (20,6) (20,4) (20,5) (6,4) (6,5)

count = 31

Shraddha

Date.
Page No.

10 In which cases Quick sort will give the best and the worst case time complexity?

Best Case → The best case occurs when the partition process always picks the middle element as pivot. Following is the recurrence relation for best case

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

Worst Case → The worst case occurs when the partition process always picks greatest or smallest element as pivot. In above partition strategy is considered where last element is always picked as pivot, the worst case would occur when the ~~above~~ array is already sorted in increasing or decreasing order.

11 Write Recurrence Relation of Merge and Quick sort in best and worst case? What are the similarities and differences between complexities of two algorithms and why?

Quick Sort → Recurrence Relation – Best Case ↓

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

Worst Case → $T(n) = T(n-1) + \theta(n)$

Merge Sort → Recurrence Relation – Best Case ↓

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

Worst Case → $T(n) = 2T(n/2) + \theta(n)$

TIME COMPLEXITY

| | Best Case | Worst Case |
|---|---|---|
| Quick Sort | $O(n \log n)$ | $O(n^2)$ |
| Merge Sort | $O(n \log n)$ | $O(n \log n)$ |

Shraddha

Date.
Page No.

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

```
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;
    }
}
```

Shraddha

Date.
Page No.

13 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 sorted.

```
void bubble_sort(int a[], int n)
  {
    for (int i=0 ; i<=n-2 ; ++i)
      { boolean flag = false;
        for (int j = 0 ; j<=n-i-2 ; ++j)
          {
            if (A[j] > A[j+1])
              {
                int temp = A[j];
                A[j] = A[j+1];
                A[j+1] = temp;
                flag = true;
              }
          }
        if (flag == false)
          break;
      }
  }
```

Shraddha

Date
Page No.

14 Your computer has a RAM (physical memory) of 2GB and you are given an array of 4GB of sorting. Which algorithm you are going to use for this purpose and why? Also, explain the concept of External and Internal Sorting

For such a sorting, we will use external sorting technique because external sorting algorithm can handle large amount of memory which cannot fit in the main memory.

EXTERNAL SORTING TECHNIQUE → In External sorting techniques, only a part of the array resides in the RAM during execution, therefore it can handle large memory array which cannot fit into the main memory

example → K-way Merge Sort.

INTERNAL SORTING TECHNIQUE → In these sorting techniques, the whole array needs to reside in the main memory i.e. RAM during execution.

example → Bubble Sort, etc.

Shraddha