Tutorial-3

Kulmeet Singh
13-F
2016827

Q1. Write linear search pseudo code to search an element in a sorted array with minimum comparisons.

Sol:-

```
for(int i=0; i<n; i++)
  {
    if(a[i]==key value)
    {
      cout<<" element found ";
      break;
    }
  }
```

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?

Sol:- iterative →

```
void insertion_sort(int a[], int n)
{
    for(int i=1; i<n; i++)
    {
        int j=i-1;
        int t= a[i];
      while(j>=0 && a[j]>t)
      {
         a[j++]= a[j];
           j--;
      }
       a[j+1]= t;
    }
}
```

Recursive :-

```
void insertion_sort(int a[], int n)
{
    if(n<=1)
    {
     return;
    }
    insertion_sort(a, n-1);
    int last = a[n-1];
    int j = n-2;
    while(j>=0 && a[j]>last)
    {
        a[j+1] = a[j];
        j--;
    }
    a[j+1] = last;
}
```

Insertion sort is called online sort because it doesn't need to know anything about what values it will sort and the information is requested while the algorithm is running.

* Other sorting Algorithm :-

1) Bubble Sort
2) Quick Sort
3) Merge Sort
4) Selection Sort
5) Heap Sort

lexity of all the sorting Algorithm that has been discussed lectures.

Sorting	Best	Worst	Average
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$
Quick	$O(n \log n)$	$O(n^2)$	$O(n \log n)$
Merge	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Heap	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Q4. Divide all the sorting algorithms into inplace / stable / online sorting.

Inplace Sorting	Stable Sorting	Online Sorting
Bubble Sort	Merge Sort	Insertion Sort
Selection Sort	Bubble Sort	
Insertion Sort	Insertion Sort	
Quick Sort	Count Sort	
Heap Sort		

Q5. Write Recursive / Iterative pseudo code for Binary Search. What is the time and space complexity of linear and Binary Search (Recursive & Iterative).

Sol:- iterative →

```
int Bin_Search (int a[], int low, int high, int key)
{
    while (low <= high)
    {
        int mid = low + (high-low)/2;
```

```
    if(a[mid]==key)
     return mid;
    else if(key < a[mid])
     high = mid-1;
    else
      low = mid+1;
   }
  return -1;
}
```

Recursive -

```
int Bin_Search(int a[], int low, int high, int key)
{
   if(low < 0)
     return -1;
  if(low < high){
  int mid = low + (high - low)/2;
  if(a[mid]==key)
   return mid;
else if(key < a[mid])
    Bin_Search(a, low, mid-1, key);
else
    Bin_Search(a, mid+1, high, key);
 }
return -1;
}
```

Complexity :-

1) Linear Search →
Iterative : $O(n)$
Recursive : $O(n)$

2) Binary Search →
Iterative : $O(\log n)$
Recursive : $O(\log n)$

Space Complexity :-

1) Linear Search $O(1)$

2) Binary Search $O(1)$

Q6. Write Recurrance Relation for Binary Recursive Search.

$$T(n) = T(n/2) + 1 \quad - (1)$$

$$T(n/2) = T(n/4) + 1$$

Putting value in eq. (1)

$$T(n) = T(n/4) + 1 + 1 \quad - (2)$$

$$T(n/4) = T(n/8) + 1 \quad - (3)$$

Putting value in eq. (2)

$$T(n) = T(n/8) + 1 + 1 + 1$$

$$\vdots$$

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

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

$\boxed{k = \log_2 n}$

So, $T(n) = T(n/n) + \log n$

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

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

Q7. Find two indexes such that $a[i] + a[j] = k$ in minimum time complexity.

Sol:-

```
for (int i=0; i<n; i++)
{
    for (int j=0; j<n; j++)
    {
        if (a[i]+a[j]==k)
        {
            cout << i << " " << j;
        }
    }
}
```

Q8. Which sorting is best for practical use? Explain.

Sol:- Quick sort is the fastest general purpose sort. In most practical situations, Quick sort is the method of choice as stability is important and if space is available then merge sort might be best.

Q9. What do you mean by number of inversions in an array? Count the no. of inversions in array,

$A[] = \{2, 2, 31, 8, 10, 20, 6, 4, 5\}$ using Merge sort.

→ A pair $(a[i], a[j])$ is said to be inversion if $a[i] > a[j]$ and if $i < j$

∴ Total no. of inversions in given array are 31 using merge sort.

which cases Quick sort will give the best and the worst case Time Complexity?

Sol:- * Worst case ($O(n^2)$) → The worst case occurs when the picked pivot is always an extreme (smallest or largest) element. This happens when input array is sorted or Reverse sorted array and either first or last element is picked as pivot.

* Best case ($O(n \log n)$) → The best case occurs when we will select pivot element as a median element.

Q11. Write Recurrance Relation of Merge Sort and Quick Sort in best and worst case? What are the similarities and differences between Complexities of two Algorithm and why?

Sol:-

Merge Sort:- 1) Best case; $T(n) = 2T(n/2) + O(n)$

2) Worst case: $T(n) = 2T(n/2) + O(n)$

i.e. $O(n \log n)$

Quick Sort:- 1) Best case; $T(n) = 2T(n/2) + O(n)$ → $O(n \log n)$

2) Worst case; $T(n) = T(n-1) + O(n)$ → $O(n^2)$

→ In Quick sort the array of elements is divided into parts repeatedly until it is not possible to divide it further. It is not necessary to divide half.

→ In Merge Sort the elements are split into 2 out again (n/2) again and again until one element is left.

Q12. Selection Sort is not stable by default but can you write a version of stable selection.

Sol:-

```
for(int i=0; i<n-1; i++)
{
    int min=i;
for(int j=i+1; j<a; j++)
{
    if(a[min]>a[j])
    {
      min=j;
    }
  int key=a[min];
  while(min>i)
  {
     a[min] = a[min-j];
      min--;
  }
 a[i]=key;
}
}
```

...t scans array even when array is sorted. Can you ...the Bubble Sort so that it does not scan the whole ...once it is sorted.

A better version of Bubble Sort, known as Bubble Sort, includes a flag that is set if exchange is made after an entire pass over the array. If no exchange is made, then it should be the array is already in order because no two element need to be switched.

In that case sort is →

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