DAA Assignment :- 02

PRASHUK JAIN
Section :- CE
Roll No :- 30
U. Roll No :- 2019578

Q13:- Your computer has a RAM (Physical Memory) of 2GB and your are given an array of 4 GB for sorting. Which sorting algorithm you are going to use for this purpose and why? Also explain the concept of External and Internal sorting.

Ans:- It is not possible to load the entire array into the memory for sorting using internal sorting In this case we would need to used external merge sort algorithm that operate on disk instead of memory
External sorting is a technique used to sort large data sets that can not be held in memory at once
It involves a combination of internal and external sorting technique
The most commonly used external sorting algorithm is external merge sort In this algorithm the data is split into smaller parts that can be fit into memory and each chunk is sorted using an internal sorting such as quick quick or heap sort The sorted chunks are then merged together in a series of passes, where the data is read from the disk, merged & written back to disk. Internal sorting is the type of sorting when no external memory is required

Q13:- 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

Ans:-
```
#include <iostream>
Using namespace std;
void modified (int a[], int n);
int main ()
{
    int n;
    cout << "Enter limit";
    cin >> n;
    int a[n];
    for (int i=0; i<n; i++)
      {
          cin >> a[i];
      }
    modified (a,n);
    for(int i=0; i<n; i++)
      {
          cout << a[i];
      }
}
void modified (int a[], int n)
{
    int t, f, c=0;
    for( int i=0; i<n; i++)
      {
          f=0;
          for(int j=0; j<n-1; j++)
            {
                if(a[j]>a[j+1])
                {
                    f=1;
                    swap(a[j], a[j+1]);
            c++;
         }
      }
      if(f==0)
      break;
   }
   cout << c <<endl;
}
```

Q12:- Selection sort is not stable can you write a version of stable selection sort

A:-

```
#include <iostream>
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];
     for ( int K= min ; K>i; K--)
      {
         a[K] = a[K-1];
      }
     a[i] = Key ;
    }
}
```

```
int main ()
{
  int n, i;
  cout << "Enter limit";
   cin >> n;
   int a[n]
   for (i=0; i<n; i++)
     cin >> a[i];

   StableSelectionSort (a, n);
   for (i=0; i<n; i++)
   {
     cout << a[i];
   }
  return 0;
}
```

Q11:- 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?

Ans The recurrence relation for the best and worst cases of Merge and quick sort is:-

Merge sort :-

Best case :- $T(n) = 2T(n/2) + \theta(n)$

worst case :- $T(n) = 2T(n/2) + \theta(n)$

Quick sort :-

Best case :- $T(n) = 2T(n+2) + \theta(n)$

Worst case :- $T(n) = T(n-1) + \theta(n)$

The main difference between the complexities of the two algorithm is that quick sort has a higher worst case time complexity than Merge sort. This is because in the worst case, Quick sort may repeatedly choose a pivot that is already the largest or smallest element in the subarray leading to unbalanced partitions and an $O(n^2)$ time complexity. Merge sort, on other hand always divide the input array into two halves and then combines them, ensuring a balanced partition and an $O(n \log n)$ time complexity in the worst case.

In terms of similarities, both Merge Sort and Quick sort are divide - and - conquer algorithm that sort an array by dividing it into smaller subarrays, sorting the subarray and then merging or combing them. Both algorithm have a best - case time complexity of $O(n\log n)$, making them efficient for sorting large database

Q10:- In which case Quick sort will give the best and the worst case time complexity?

A:- In the best case scenario, Quick sort will have a time complexity of $O(n \log n)$ when the pivot choosen for each partition divides the input array into two almost equal-sized subarrays. This will result in a balanced partitioning and a quick sorting process

In the worst case scenario, Quick sort will have a time complexity of $O(n^2)$ when the pivot choosen for each partition results in unbalanced partition, with one partition having all elements greater and smaller than the pivot. In such cases, each partitioning step will only reduce the size of the input array by one element resulting in a partitioning steps and a time complexity of $O(n^2)$.

Q8:- Which sorting is best for practical use? Explain

A:- Quick sort is widely used sorting algorithm that has an average time complexity of $O(n \log n)$ and is often faster than other popular sorting algorithm. Quicksort is particularly efficient for larger database & can be easily implemented in place to save memory. However, its worst case time complexity is $O(n^2)$ which can occur when the input data is already sorted.

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

A:-

```
# include <iostream>
using namespace std;
void search (int a[], int n, int key);
int binary (int a[], int l, int r, int key);
 int main ( )
 {
  int n, key;
  cout <<"Enter limit";
   cin >>n;
   int a[n]
  for (int i=0; i<n; i++)
   {
     cin >> a[i];
   }
   cout <<" Enter key ";
    cin >>key;
    search( a, n, key);
  }
 void search (int a[], int n, int key)
  {
     int k, j;
    for(int i=0; i<n; i++)
     {
       k = key - a[i];
       j= binary(a, 0, n, k);
       if(j>-1 && j!=i)
        {
         cout <<i<<j;
          break;
        }
        }
      }
    }
int binary (int a[], int l, int r, int key)
 {
   int mid;
   while (l<=r)
   {
    mid = (l+r)/2;
    if (a[mid]==key)
    {return mid;
    }
    else if (a[mid]>k)
     r=mid-1;
     else
     l= mid+1;
    }
   return -1;
 }
```

Q6:- Write recurrence relation for binary recursive search

Ans Recurrence relation :- $T(n) = T(n/2) + 1$

Q5:- Write recursive/iterative pseudo code for binary search What is the time and space complexity of linear and Binary search (Recursive and Iterative)

Ans:- Recursive Pseudo code for binary search:-

```
int binary search(int a[], int l, int h, int key)
{
     if (l>h)
     return -1;
    int mid = (l+h)/2;
     if (A[mid] == key)
     return mid;
else if( key > A[mid])
return binarySearch(a, mid+1, h, key);
return binarysearch(a, l, mid-1, key);
}
```

Now Iterative binary search:-

```
int binarysearch(int a[], int l, int h, int key)
 {
   while(l<=h)
    {
    int mid = (l+h)/2;
    if(A[mid]==key)
     return mid;
     else if( A[mid]> key)
       h= mid-1;
      else
        l= mid+1;
     }
    return -1;
 }
```

	Time complexity			Space complexity
	Best	Average	Worst	
Binary search (Recursive)	O(1)	O(logn)	O(logn)	O(1)
Binary search (Iterative)	O(1)	O(log n)	O(log n)	O(1)
Linear search (Recursive)	O(1)	O(n)	O(n)	O(n)
Linear search (Iterative)	O(1)	O(n)	O(n)	O(1)

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

Ans:-

	Inplace	Stable	Online
Bubble sort	Yes	Yes	NO
selection Sort	Yes	NO	NO
Insertion Sort	Yes	Yes	Yes
Quick sort	Yes	NO	NO
Merge Sort	NO	Yes	NO
Heap Sort	Yes	NO	Yes
Count Sort	NO	Yes	NO

Q3:- Complexity of all the sorting algorithms

Ans:-

	Best case	worst case	Average case	Space complexity
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Merge sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
Quick sort	$O(n \log n)$	$O(n^2)$	$O(n \log n)$	$O(1)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$
Count Sort	$O(n+K)$	$O(n+K)$	$O(n+K$	$O(n+K)$

Q2:: write pseudo code for iterative and recursive insertion sort Insertion sort is called online sorting. why? What about other sorting algorithm?

Ans: Insertion sort iterative code:-

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

Recursive Insertion sort code :-

```
void  recursiveInsertionSort( int a[], int n)
  {
     if (n<=1)
     return;
   RecursiveInsertionSort(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 sometimes called as "online sorting algorithm" because it can sort list of elements as they are being received one at a time, without having to wait for the entire list to be received or processed first

Q1: Write linear search pseudocode to search an element in a sorted array with minimum comparsions.

A:-
```
#include <iostream>
using namespace std;
int linear seach (int a[], int key, int n)
  {
  for (int i=0; i<n; i++)
    {
     if (a[i]==key)
      {
       return i;
      }
     else if (a[i] > key)
       return-1;
    }
   return-1;
  }
```

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

Ans:- The inversion count for any array is the number of steps it will take for the array to be sorted, or how far away any array is from being sorted

* If array = {7,21,31,8,10,1,20,6,4,5} :-

In this there are 27 inversions in this array using merge sort