Sakshi Sahu
CST SPL-1
Roll NO:03

PAGE NO.
DATE: / /

DAA Tutorial 3

QUS1 write a linear search pseudo code to search an element in a sorted array with minimum comparisions.

```
int linear_search (int A[], int n, int t)
{
  if (abs (A[0]-t) > abs (A[n-1]-t))
      for (i = n-1 to 0; i--)
        if (A[i] == t) { return i ; }

  else
      for (i=0 to n-1; i++)
        if (A[i] == t)
          return i;
}
```

Ans 2) Iterative Insertion sort.

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

PAGE NO.
DATE: / /

Insertion sort is also called online sorting algorithm because it will work if the element to be sorted are provided one at a time with the understanding that the algorithm must keep the sequence sorted as more element are added in.

Ans3) Complexity of all the sorting algorithms are given below :

sorting	Best case	worst case
Bubble sort	$O(n^2)$	$O(n^2)$
Selection sort	$O(n^2)$	$O(n^2)$
Insertion sort	$O(n^2)$	$O(n^2)$
Count sort	$O(n)$	$O(n+k)$
Quick sort	$O(n \log n)$	$O(n^2 \ldots)$
merge sort	$O(n \log n)$	$O(n \log n)$
Heap sort	$O(n \log n)$	$O(n \log n)$

Ans 4

sorting	Inplace	Stable	online
Bubble	✓	✓	✗
Selection	✓	✗	✗
insertion	✓	✓	✓
Count	✗	✓	✗
QUICK	✓	✗	✗
merge	✗	✓	✗
Heap	✓	✗	✗

Iterative

Ans 5) Recursive / pseudo code for binary search

Iterative

```
int binarysearch ( int arr [ ], int x )
{
    int l = 0, r = arr.length - 1;
    while (l <= r)
    {
        int m = l + (r - l) / 2;

    if (arr[m] == x)
        return m;
    if (arr[m] < x)
        l = m + 1;
    else
        r = m - 1;

    }
    return -1;
}
```

Recursive

```
int binarysearch ( int arr[ ], int l , int r;
                   int x )
{
    if (r >= l)
    {
        int mid = l + (r - l) / 2;
        if (arr[mid] == x)
            return mid;
```

```
else if (arr[mid] > x)
    return binarysearch(arr, l, mid-1, x);
else
    return binarysearch(arr, mid+1, r, x);
}
return (-1);
}
```

* Linear Search

Iterative : Time complexity = O(n)
space complexity = O(1)

Recursive : Time complexity = O(n)
space complexity = O(n)

* Binary Search

Iterative : Time complexity = O(log n)
space complexity = O(1)

Recursive : Time complexity = O(log n)
space complexity = O(log n)

PAGE NO.
DATE: / /

Ans 6

$T(n)$
↓
$T(n/2)$
↓
$T(n/4)$
↓
.
.
$T(n/2^R)$

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

Ans 7

```
int n;
int A[n];
int key;
int i=0, j = n-i;
while (i<j)
{
  if ((A[i] + A[j]) == key)
      break;
  else if ((A[i] + A[j]) > key)
      j--;

  else
      i++;
}
cout << i << " " << j;
```

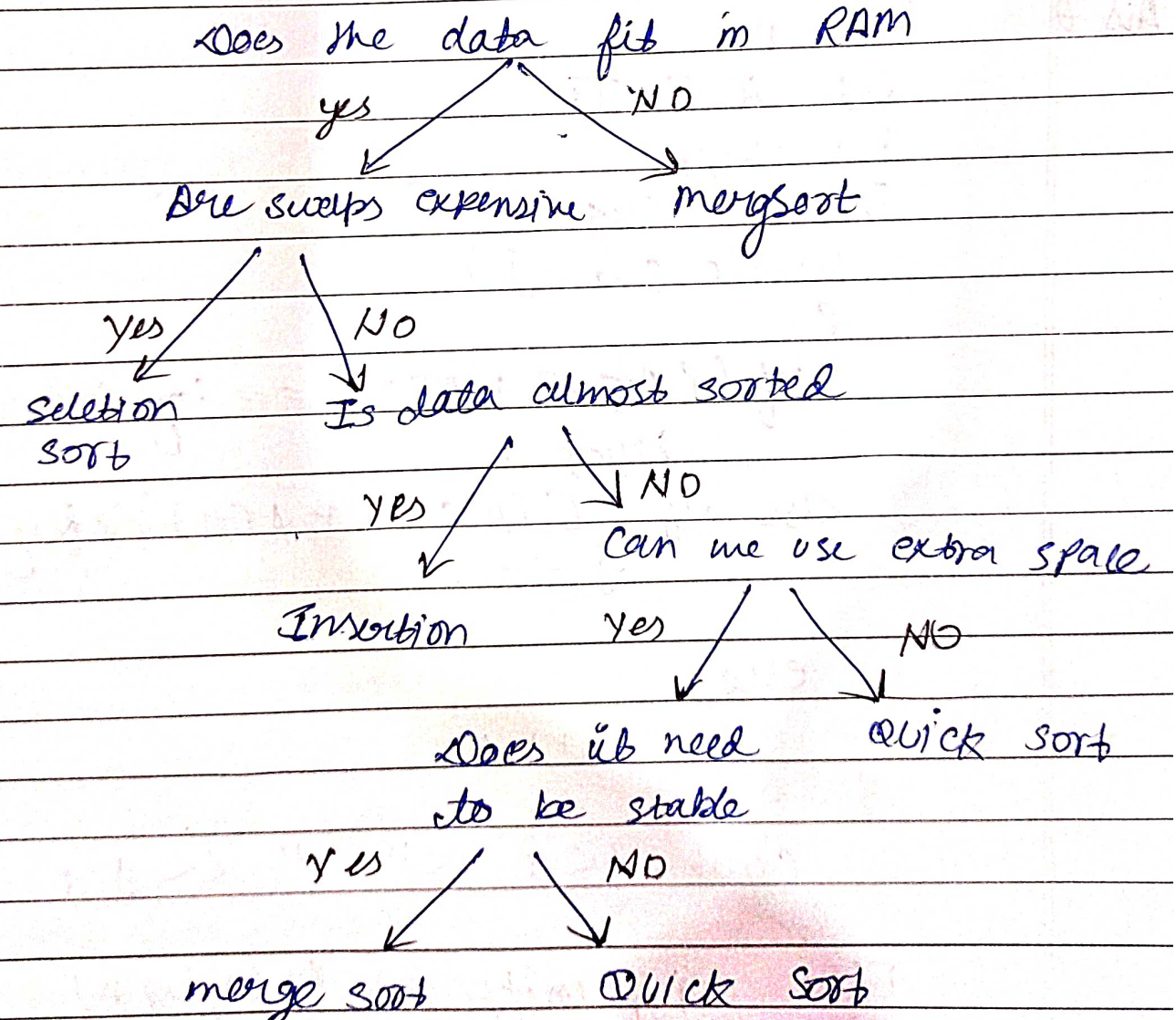
Time complexity = $O(n \log n)$

PAGE NO.
DATE: / /

Ans 8 There is no best sorting algorithm. It depends on the situation or the type of any array given.

Factor affecting sorting algorithm are:

i) Runtime
ii) space
iii) stable
iv) No. of swaps
v) will the data fit in RAM



PAGE NO.
DATE: / /

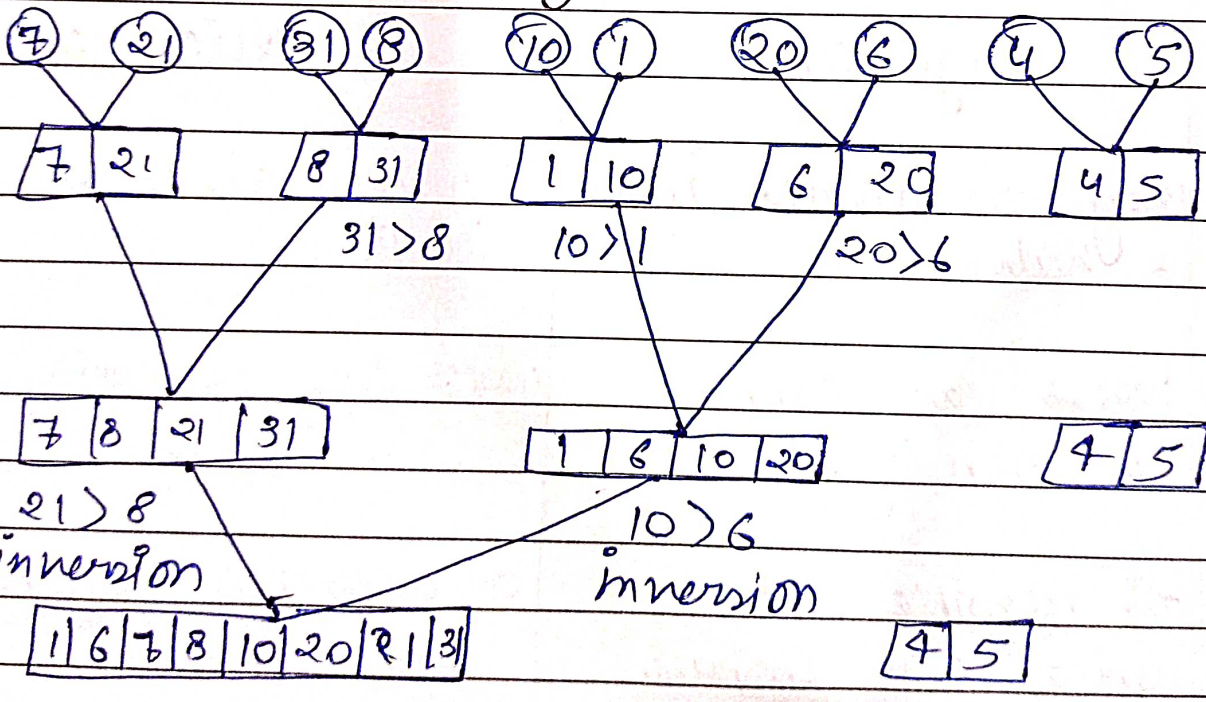
Ans (9) Inversions is an array indicates how far the array is from being sorted. If the array is ~~form~~ already sorted, the inversion count is 0, but if the array is sorted is reverse order. then the inversion is maximum.

conversion for inversion

$a[i] > a[j]$ and $i < j$

7	21	31	8	10	1	20	6	4	5

Dividing the array



7>4, 6>5, 7>4, 7>5, 8>4, 8>5, 10>4, 10>5, 20>4, 20>5
21>4, 21>5, 31>4, 31>5

inversion count = 31 Ans

Ans 10 Best case

merge sort = $2T(n/2) + n$
quick sort = $2T(n/2) + n$

worst case

merge sort = $2T(n/2) + n$
quick sort = $T(n-1) + n$

Similarities : They both work on concept of divide & conquire algorithm

Difference

	merge sort	quick sort
1)	They are divided into 2 halves	They are divide in any ratio
2)	worst case compexity is $O(n \log n)$	worst case complexity is $O(n^2)$
3)	It require extra space NOT INSpace	It does not require extra space
4)	It is external sorting	It is internal sorting
5)	work consistently on any size of data set	work fast on small data set

PAGE NO.
DATE: / /

Ans 10) Best case

Time Complexity = $o(n \log n)$

Best case occur when the partition process always picks the middle element as pivot

* worst case

Time complexity = $O(n^2)$

when the array is sorted in ascending or decending order.

Ans 12)

```
void selection (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)
          {
              min --;
          }
          A[i] = key;
      }
}
```

PAGE NO.
DATE: / /

Ans 13)

```
void bubblesort ( int A[], int n )
{
  int i, j;
  int f = 0;

for (i = 0; i < n; i++)
  {
    for ( j = 0; j < n-1; j++)
    {
      if (A[j] > A [j+1])
        {
          swap ( A[j], A [j+1]
            j = 1;
        }
    }
if ( f == 0)

    break ;

}
```

Ans 15 * your computer ——— ——— ——— ——— ——— ——— ——— ——— ——— sorting

Ans 13 When the data set is large enough to fit inside RAM, we use merge sort because it uses the divide and conquer approach in which it keeps dividing the array into smaller parts

until it can no longer be splitted.

It then merge the array divided in n parts. Therefore at a time only a part of array is taken on RAM

External sorting

It is use to sort massive ammount of data. It is required when the data does not fit inside RAM and insted they meet reside in the slower external memory.

During sorting, chuncks of small data that can fit in main memory are read, sorted and written out to a temporary file.

During merging, the sorted subfiles are combine into a single large file

Internal sorting

It is a type of sorting which is used when the enter collection of data is small enough to reside within RAM. Then there is not need of external memory for program execution.