Date ___/___/_____

Set 5

1. Define complexity and describe the different types of complexities used to evaluate algorithm 3M

⇒ Compexity of an algorithm refers to the amount of resources (like time and memory) required by the algo.

Types of complexities ! Set 3. Q.2

2. Write short notes of types of recursion with example. 3M

⇒ Set-2 Q.6

3. Derive index formula for a 2-D array in row major order. 4M

Set-1 Q.3

4 Write a brief note on searching techniques and explain binary search. 4M.

⇒ Searching techniques :-

→ Searching means finding the location or presence of a specific element (key) in a list or an array.

e.g. if we have an array [1, 2, 3, 4, 5] and we have to find 4. then searching tells us that 4 is at 3 index.

⇒ There are two main types of searching.

1. Linear Search: checks all element one by one untill the desired element is found. Works for all types of array means no need to sort array. TC = $O(n^2)$

2. Binary search: Repeatedly divides into two parts to find element quickly. Works on only sorted array. TC = $O(\log(n))$.

Binary search : Set 1 Q no 4

5. Explain Quick sort algorithm and give a real life example where it can be used. 5M

Set 2. Q 5 and Q.10

6. Write an iterative algorithm for calculating Fibbonacci numbers and explain it 5M

⇒ The fibboncci series is the sequence of numbers where each number is the sum of previous two numbers.

$$f(n) = f(n-1) + f(n-2)$$
$$f(0) = 0$$
$$f(1) = 1$$

Algorithm:

⇒ Start

⇒ Read integer (number of terms)

⇒ Initialize a=0, b=1.

⇒ Print a, b.

```
⇒ loop for i=2 to n-1
      → c = a+b
      → print c
      → a = b
      → b = c
⇒ Stop.
```

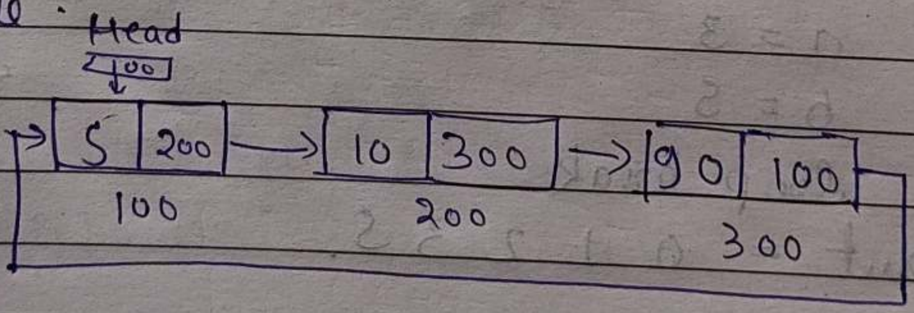
```
⇒ Example for n = 6.
        print  0 , 1     // 0 1
  i=2   c = a+b = 0+1 = 1
        print c          // 0 1 1
        a = 1
        b = 1
  i = 3  c = 2
        print c     // 0 1 1 2
        a = 1
        b = 2
  i = 4  c = 3
        print c .// 0 1 2 3
        a = 2
        b = 3
  i = 5  c = 5
        print c   // 0 1 2 3 5
        a = 3
        b = 5
  i = 6  loop break
  output = 0 1 2 3 5.
```

code

```
void fibb(int n) {

    int a=0; int b=1; int c;
    cout<<a<<" "<<b<<" ";

    for(int i=2; i<n; i++){

        ~~int~~ c = a+b;
            cout<<c<<" ";
            a=b;
            b=c;
    }
}
```

7. Explain circularly linked list and its applications. Write the algorithm for deletion in a circular linked list. 8 M.

⇒ A circular linked List is a variation of a Linked list in which the last node points back to the first node, forming a circle.



Here.

→ Each node contains data and a next pointer

→ The next pointer of the last node points to the head instead of NULL.

⇒ Types of circular Linked List.

i) Singly circular Linked List
⇒ Each node has one pointer (next) pointing to the next node, and last node points to head.

ii) Doubly Circular Linked List
Each node has two pointers (prev and next) and links form a full circular chain in both direction.

Application of circular LL

i) Operating systems:
⇒ Used to manage CPU scheduling where processes are cyclically executed.

ii) Buffer Management:
Used in data buffers, such as audio/video streaming where old data is overwritten in a circular manner.

iii) Gaming
⇒ To keep track of player turns in a continuous cycle.

iv) Music or playlist Apps
⇒ To loop through playlist continuously.

Algorithm: (Pseudo code)

```
1.  If head = null.
        print "Lis is empty"
       and return.
2.  Initialize:
            current = head
            previous = null
3.  If head -> data == key and head -> next == head
        (only one node in list)
              free(head)
              head = null
              return.
4.  If had -> data == key
        (delete head node)
       Find the last node (say temp) such that
           temp -> next == head
           temp -> next = head -> next
           head = head -> next
           free(current)
           return
5.  Else
         Repeat while current -> next ! = head:
              previaus = current
              current = current -> next
          If current -> data == key:
               priviaus -> next = current -> next
               free(Current)
              return.
6.    If curent -> next == head
         print "Node not found"
```

8(a) Compare recursive and iterative approaches with reference to stack usage. 4m

⇒ Both recursion and iteration are methods used to perform repetetive tasks in programming.

⇒ They handle repetetion differently especially in terms of stack usage and memory management.

→ Recursion

Recursion is process where a function calls itself directly or indirectly untill a base condition is met.

Stack usage:

→ Each recursive function call is stored in the system call stack.

→ The stack keeps track of:

→ Function parameters

→ Return addresses

→ Local vars

→ When the base condition is reached, the stack starts unwinding.

e.g.

```
int fact (int n) {
    if (n==0) { return 1; }

    return n* fact(n-1);
}
```

⇒ Each call to fact() is pushed onto the function call stack.

⇒ Iteration

⇒ Iteration uses looping constructs (like for, while, do-while) to repeat the set of statements until a condition becomes false.

→ Stack usage!

⇒ No function call stack is created for iteration

⇒ Uses a single memory block for loops and control vars.

⇒ The stack is used only once for the function itself, not for each iteration.

e.g.

```
int fact (int n){
    int res = 1;

    for(int i = 1; i<=n; i++)
        res *= i;
    return res;
}
```

⇒ only one frame is created in the stack for this function.

8 (b) Write a recursive program to compute factorial of a number. 4 M

Set 3 Q no 6

9.) Write the algorithm for merge sort and explain its divide and conquer approach. Discuss its significance in large dataset sorting. 10 M

Set 1 Q 9

Its significance in large dataset sorting.

1.) Efficiency:
Merge sort consistently performs in O(nlogn), even in the wors case making it reliable for large data set.

2.) Stability!
Maintains the relative order of equal elements, useful in database and record sorting.

3.) External sorting!
Works well for data stored on disks or external devices because it accesses data sequentially.

4. Parallel Processing:
⇒ The divide and conquer structure allows easy parallelization of sublists for faster processing.

5. Predictable Performance
⇒ Unlike Quick Sort, performance doesn't degrade due to bad pivot choices.

⇒ Advantages of Merge sort.
→ Stable sorting algorithm.
→ Suitable for large data Bases.
→ Predictable O(n log n) performance.
→ Excellent for linked lists and external sorting.

⇒ Disadvantages:
→ Requires extra memory (O(n)) for temporary arrays.
→ Recursive calls increases overhead for small data sets.