MODULE 1
ASYMPTOTIC NOTATIONS FOR COMPLEXITY OF ALGORITHMS.

1. Big "oh" [O]
$f(n) = O(g(n))$ iff there exist 2 +ve constants c and n_0 such that
$|f(n)| \le c.|g(n)|$ for all $n \ge n_0$
f(n) = computing time of some algorithm. • When we say that the computing time of an algorithm is O(g(n)),
we mean that its execution takes no more than a constant timeg(n).

□Properties of Big "oh"
• If the time complexity of f(n) is O(g(n)) and the time complexity ofg(n) is O(h(n)), then f(n) has a time complexity of O(h(n)) • If f(n)= O(h(n)) and g(n) = O(h(n)), then f(n)+g(n)= O(h(n)) • an•
has a time complexity of O(n•
) where, a is constant. • In Big Oh, g(n) is the upper bound of f(n) • Rate of growth – 1, logn, n, nlogn, n•
, n•
, 2•
. These functions aregeneral functions which is same as g(n)

2. Omega (Ω) • $f(n) = \Omega(g(n))$ iff there exist +ve constants c and n_0 such that
$f(n) \ge c.g(n)$ for all n, $n \ge n_0$
• Here g(n) is the lower bound of f(n)
Eg: 10n•
+4n +2
f(n) ≥ c. g(n)
f(n) ≥ n•
for n≥1TC = Ω (• Ł Ł)

3. Theta (θ) • $f(n) = \theta(g(n))$ iff there exist +ve constants c_1 , c_2 and n_0 such that
$c_1.g(n) \le f(n) \le c_2.g(n)$ for all n, $n \ge n_0$
• Gives average case TC
Eg:3n +2
TC = θ (n)
4. Little oh (o) • for f(n)=o(g(n)), then lim
• →□
• (•)
• (•) =0 where g(n)≠0
• TC will be one added to the greatest power of the given polynomial
Eg: f(n)= 3n+1
TC = o(•
•
)
f(n) = 2n•
+4n +5
TC = o(•
•
)
5. Little Omega (ω) • for f(n)=ω(g(n)), then lim
• →□
• (•)
• (•) =0 where f(n)≠0
Eg: 4n•
+2n
TC = ω(n)
3n+2
TC= ω(1)

MODULE 2:

□**Application of stack**
• When a function call occurs , the return address is stored in stack• Number system conversion
• Maintaining undo list for word document application
• Sorting
• Expression evaluation
• Expression conversion
• String reversal • Used for implementing subroutines in general programminglanguage.

DOUBLE ENDED QUEUE :
PUSH :

Algorithm Push_DQ(DQ, item)
1. Start
2. If front = 0 then
3. ahead = SIZE -1
4. else
5. if (front = SIZE -1)
or (front = -1)
6. ahead = 0
7. else
8. ahead = front -1
9. end if
10. end if
11. If ahead = rear then
12. Print 'Deque is
full'
13. Exit
14. else
15. front = ahead
16. DQ[front] =item
17. end if
18. Stop

POP:
Algorithm Dequeue_CQ(CQ)
1. Start
2. If front = -1 then
3. Print "CQ is empty"
4. Exit
5. Else
6. item = CQ[front]
7. If (front = rear)
8. front = rear = -1
9. else
10. front = (front+1) MOD LENGTH
11. End if
12. End if
13. Stop

□□**Application of Queue**
1) In a multiuser system, task form a queue waiting to be executedone after another.
2) In computer networks, the packets that arrive from various linesare kept in a queue
3) For performing breadth First Search (BFS) of a graph, queue isused
4) Queues are generally used for ordering events on first in first outbasis

MODULE 2

Binary Search Algo
1. Step 1: set beg = lower_bound, end = upper_bound, pos = - 1
2. Step 2: repeat steps 3 and 4 while beg <=end
3. Step 3: set mid = (beg + end)/2
4. Step 4: if a[mid] = val
5. set pos = mid
6. print pos
7. go to step 6
8. else if a[mid] > val
9. set end = mid - 1
10. else
11. set beg = mid + 1
12. [end of if]
13. [end of loop]
14. Step 5: if pos = -1
15. print "value is not present in the array" 16. [end of if]
17. Step 6: exit
The time complexity of the binary search algorithm is O(log n). The
best-case time complexity would be O(1) when the central index would
directly match the desired value. Binary search worst case differs from that.
The worst-case scenario could be the values at either extremity of the list or values not in the list.

Algorithm For Inserting An Element Into A Linear Queue
Assume FRONT=REAR= -1
Step1 : Start
Step 2: If REAR=MAX.SIZE then
2.1 : Print Queue is Full
2.2 : Exit
Step 3 : Else
3.1: If FRONT= -1 and REAR = -1 then
FRONT =REAR= 0.
3.2: Q[REAR] = ITEM
3.3 : REAR= REAR + 1

Algorithm For Deleting An Element From A Linear Queue
Input : Queue with elements with FRONT and REAR pointer. Output: Deleted element ,ITEM.
Data Structure: Array representation of queue.
Step 1: If FRONT= -1 then
1.1: Print Queue is Empty
1.2: Exit
Step 2 : Else
2.1: ITEM= Q[FRONT].
2.2: If FRONT=REAR then
FRONT=REAR=-1
2.3: Else
FRONT=FRONT+1
Step 4: End if
Step 5: Stop

Algorithm to insert (enqueue) an element into circular Queue

```
{
rear=(rear+1)mod n;
if (front==rear) then
print("Queue is full");
else
queue[rear]=x;
}
```

Algorithm to delete (dequeue) an element from circular Queue

```
{
if (front==rear) then
print("Queue is empty");
else
{
front=(front+1)mod n;
item=queue[front];
}
}
```

MODULE 3

DYNAMIC MEMORY ALLOCATION

Memory Allocation Process

• Global variables, static variables and program instructions get theirmemory in permanent storage area whereas local variables arestored in a memory area called Stack.
• The memory space between these two region is known asHeaparea. This region is used for dynamic memory allocationduringexecution of the program. The size of heap keep changing.
• The process of allocating memory at runtime is known as dynamicmemory allocation. Library routines known as memorymanagement functions are used for allocating and freeing memoryduring execution of a program.

LINKED LIST NODE ADD&DEL:

Algorithm: InsertAtBeginning:

```
Step 1: IF AVAIL = NULL
          Write OVERFLOW
          Go to Step 7
       [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = HEAD
Step 6: SET HEAD = NEW_NODE
Step 7: EXIT
```

Algorithm: InsertAtEnd

```
Step 1: IF AVAIL = NULL
          Write OVERFLOW
          Go to Step 10
       [END OF IF]
Step 2: SET NEW_NODE = AVAIL
Step 3: SET AVAIL = AVAIL -> NEXT
Step 4: SET NEW_NODE -> DATA = VAL
Step 5: SET NEW_NODE -> NEXT = NULL
Step 6: SET PTR = HEAD
Step 7: Repeat Step 8 while PTR -> NEXT != NULL
Step 8:     SET PTR = PTR -> NEXT
       [END OF LOOP]
Step 9: SET PTR -> NEXT = NEW_NODE
Step 10: EXIT
```

Algorithm: DeleteFromBeginning

```
Step 1: IF HEAD = NULL
          Write UNDERFLOW
          Go to Step 5
       [END OF IF]
Step 2: SET PTR = HEAD
Step 3: SET HEAD = HEAD -> NEXT
Step 4: FREE PTR
Step 5: EXIT
```

Algorithm: DeleteFromEnd

```
Step 1: IF HEAD = NULL
          Write UNDERFLOW
          Go to Step 8
       [END OF IF]
Step 2: SET PTR = HEAD
Step 3: Repeat Steps 4 and 5 while PTR -> NEXT != NULL
Step 4:     SET PREPTR = PTR
Step 5:     SET PTR = PTR -> NEXT
       [END OF LOOP]
Step 6: SET PREPTR -> NEXT = NULL
Step 7: FREE PTR
Step 8: EXIT
```

Doubly linked list

• In a single linked list, every node has a link to its next node in thesequence. So, we can traverse from one node to another node onlyin one direction and we can not traverse back.
• We can solve this kind of problem by using a double linked list.
• In a double linked list, every node has a link to its previous nodeand next node
• So, we can traverse forward by using the next field andcantraverse backward by using the previous field.
• Every node in a double linked list contains three fields
• In double linked list, the first node must be always pointed by head.
• Always the previous field of the first node must be NULL.
• Always the next field of the last node must be NULL.

CIRCULAR SINGLY LINKED LIST

• In a circular Singly linked list, the last node of the list contains apointer to the first node of the list. We can have circular singlylinked list as well as circular doubly linked list.
• We traverse a circular singly linked list until we reach the samenode where we started.
• The circular singly liked list has no beginning and no ending.
• There is no null value present in the next part of any of the nodes.
• Circular linked list are mostly used in task maintenance in operatingsystems.
• There are many examples where circular linked list are being usedin computer science including browser surfing where a record ofpages visited in the past by the user, is maintained in the form ofcircular linked lists and can be accessed again on clicking theprevious button.

CIRCULAR DOUBLY LINKED LIST

• Circular doubly linked list is a more complex type of data structurein which a node contain pointers to its previous node as well as thenext node.
• Circular doubly linked list doesn't contain NULL in any of the node.
• The last node of the list contains the address of the first node ofthe list.
• The first node of the list also contain address of the last nodein itsprevious pointer
• Due to the fact that a circular doubly linked list contains three partsin its structure therefore, it demands more space per nodeandmore expensive basic operations.
• However, a circular doubly linked list provides easy manipulation ofthe pointers and the searching becomes twice as efficient.

MEMORY ALLOCATION SCHEME

FIRST FIT - In the first fit approach is to allocate the first freepartition or hole large enough which can accommodate the process. It finishes after finding the first suitable free partition.
BEST FIT - The best fit deals with allocating the smallest freepartition which meets the requirement of the requesting process. This algorithm first searches the entire list of free partitions andconsiders the smallest hole that is adequate. It then tries to find ahole which is close to actual process size needed.
WORST FIT - In worst fit approach is to locate largest available freeportion so that the portion left will be big enough to be useful. It isthe reverse of best fit.

MODULE 4

TREE

• A tree is a nonlinear data structure. Data elements are related in ahierarchal manner (parent-child relationship)

• The first node of the tree is called Root node.

• Final nodes are called Leaf nodes.

• Leaf nodes are also called terminal nodes

• A child has a single parent but a parent has more than onechildren. So we can say that , here, a one to many relationship isexist.

• In a general tree, A node can have any number of children nodesbut it can have only a single parent.

TREE REPRESENTATION

Tree can be represented using two ways
1. Using Array
2. Using Linked List

If a complete binary tree with n node, then depth = • $\log_2 n$• +1 , in an array representation,

□Parent (i) is at • i/2•

□Lchild (i) is at 2i

□Rchild (i) is at 2i+1

Algorithm to search an element in Binary search tree

```
Search (root, item)
Step 1 - if (item = root → data) or (root = NULL)
return root
else if (item < root → data)
return Search(root → left, item)
else
return Search(root → right, item)
END if
Step 2 - END
```

BST INSERT ALGO

```
If node == NULL
    return createNode(data)
if (data < node->data)
    node->left  = insert(node->left, data);
else if (data > node->data)
    node->right = insert(node->right, data);
return node;
```

GRAPH TRAVERSAL:

BFS ALGO:

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until QUEUE is empty

Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).

Step 5: Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2
(waiting state)
[END OF LOOP]].

DFS Algorithm

Step 1: SET STATUS = 1 (ready state) for each node in G

Step 2: Push the starting node A on the stack and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until STACK is empty

Step 4: Pop the top node N. Process it and set its STATUS = 3 (processed state)

Step 5: Push on the stack all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)
[END OF LOOP]

Step 6: EXIT

GRAPH REPRESENTATION IN MEMORY THUND NAMMALH VARAKKANAM.

MODULE 5

HASHING FUNCTIONS
1. Mid Square
2. Division
3. Folding
4. Digit Analysis

□**Mid Square** – Let k=3205. The hash function squares the k .thatis, $k^2 = (3205)^2$
= 102 |72| 025

• Take middle value. This middle value is the address of bucket. Midsquare is applied , when only the bucket size is a power of 2.

□**Division**

f(x) = x % m

• For reducing the collision we use prime numbers for m. • The range of bucket address is 0 to m-1 (m is a constant, ie, hashtable size)

□**Folding**

• We divide the key into some parts and add each parts

Eg: 30 |25| 0
= 30+25+0 = 55

This 55 is take as a bucket address. This method is also called shiftfolding

Folding at boundaries

• Here we also divide the key into some parts. We take thealternative reverse of the number and add it

Eg: 30 |25| 0
= 30+52+0 = 82

• 82 is taken as a bucket address

□**Digit Analysis**

• This method is particularly useful in the case of a static filewhereall the identifiers in the table are known in advance. • Each identifier x is interpreted as a number using some radix "r"

• This hashing function is distribution dependent • All of the inputs that must be hashed are known in advance• Here we make a statistical analysis of digits of the key, and selectthose digits (of fixed position) which occur quite frequently• Then reverse or shift the digits to get the address

Eg: If the key is 9861234 . If the statistical analysis has revealed thefact that the third and fifth position digits occur quite frequently,
then we choose the digits in these positions from the key. So we get62. Reversing it , we get 26 as the address

Heap Sort

```
HeapSort(arr)
BuildMaxHeap(arr)
for i = length(arr) to 2
swap arr[1] with arr[i]
heap_size[arr] = heap_size[arr] ? 1
MaxHeapify(arr,1)
End
BuildMaxHeap(arr)
heap_size(arr) = length(arr)
for i = length(arr)/2 to 1
MaxHeapify(arr,i)
End
MaxHeapify(arr,i)
L = left(i)
R = right(i)
if L ? heap_size[arr] and arr[L] > arr[i]
largest = L
else
largest = i
if R ? heap_size[arr] and arr[R] >
arr[largest]
largest = R
if largest != i
swap arr[i] with arr[largest]
MaxHeapify(arr,largest)
End
```

Algorithm:mergesort(low,high)

```
if (low<=high){
mid=(low+high)/2
mergesort(low,mid) //divide
mergesort(mid+1,high) //divide
merge(low,mid,high) //conquer
}
Merge(low,mid,high){
1.Assign i=low,j=mid+1,k=low. 2. While
(i<=mid && j<=high)
2.1 . If a[i]<a[j]{
b[k]=a[i]
i=i+1
k=k+1
}
2.2 else{
b[k]=a[j]
j=j+1
k=k+1
}
3. While(i<=mid)
3.1: b[k]=a[i]
3.2: i=i+1
3.3:k=k+1
4. While(j<=high)
4.1: b[k]=a[j]
4.2: j=j+1
4.3:k=k+1
5. Assign i=low
6.While i<=high
6.1. a[i]=b[i]
6.2 . i=i+1
```

```
Algorithm
Quicksort(low,high){
if(low<high){
j=partition(A,low,high)
Quicksort(low,j-1)
Quicksort(j,high-1) }
partition(A,low,high){
1. piv=A[0].
2. Assign l=low,h=high
3. While(l<=h)
3.1 .while(A[l]<pivot)
l++;
3.2 .while(A[h]>pivot)
h--;
3.3. if(l<h)
swap (A[l],A[h])
4. swap(piv,A[h])
5. Return h
```

GRAPH REPRESENTATION IN MEMORY

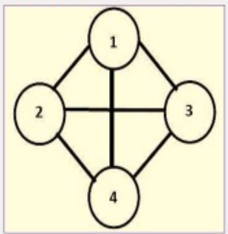
The most commonly used representations are

- **Adjacency Matrix**
- **Adjacency List**
- **Adjacency Multilist**
- **Sequential Representation**

1. Adjacency Matrix

- The graph can be represented in a matrix form.
- Let G=(V, E) be a graph with "n" vertices. The size of the adjacency matrix G is **n×n**.

- A[i , j] =1 indicate that an edge is present between the vertex v_i and v_j.
- A[i , j] = 0 indicate that an edge is not present in between the vertex v_i and v_j.



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

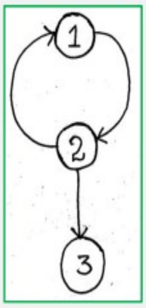
n=4

Size of matrix = 4×4

Degree of vertex v_1 = number of ones in row or column in 1. That is 3

- If the graph is undirected, the obtaining matrix is a symmetric matrix. So we can store the upper portion or the lower portion of the matrix in memory.

Consider the following directed graph



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \end{array}$$

The matrix is not symmetric. The row sum of the matrix indicate the "out degree" and the column sum indicate "in degree"

Out degree of V1 = 1
In degree of V1 = 1
Out degree of V2 = 2
In degree if V2 = 1

- If we represented a graph (Directed or undirected) in the adjacency matrix form, the diagonal elements are always zero. So we can avoid the searching of diagonal elements.
- Number of search = n^2-n
- Time complexity = **$O(n^2)$**
- The searching time can be further reduced as O(n+e) where, n is number of vertices and e is number of edges. For reducing the time complexity of adjacency matrix, we use adjacency list