**1 LECTURE**

TOPICS COVERED

WHAT IS LL? \*\*\* don’t tamper the head and always return the head

USES?

MEMORY SPACE?

ARRAY TO LL

TRAVERSAL OF LL

COUNT ELEMENTS IN LL

SEARCH ELEMENT IN LL

**// ARRAY TO LL STEPS**

1. Assign the temp,head,mover
2. Create the node for each array elements
3. Point the mover to next temp
4. Move the mover to the temp

**// Traversal os LL**

1. Assign the head to the temp
2. Till the temp != null
3. Print the element
4. Point the temp to next element

**// Count the elements in LL**

1. Assign the head to the temp,count =0
2. Till the temp != null
3. Count ++
4. Point the temp to next element

**// Search element in LL**

1. Assign the head to temp
2. Till the temp != null
3. If temp’s data == target
4. Return 1
5. Temp point to the next element
6. Outside the loop return 0

**2nd LECTURE**

TOPICS COVERED

DELETION OF LL

1. HEAD
2. TAIL
3. POSITION
4. POSITION VALUE

INSERTION OF LL

1. HEAD
2. TAIL
3. POSITION
4. POSITION VALUE

**DELETION OF LL**

**// HEAD**

1. Base condition check the head is null or having the single element
2. Retun null
3. Assign the head to the next element
4. Return the head

**// TAIL**

1. Assign the head to the temp
2. Check for 0th element
3. Head == null (0 ) || head.next == null (1)
4. Return null;
5. Go till the second last != null
6. Temp points to the next
7. Else temp points to null

**// POSITION**

1. Check the for base condiion
2. K ==1 call the head
3. Assign temp to head,
4. Assign prev = null
5. Count = 0
6. Go till the last element
7. Increase the count
8. Check the count == k
9. Prev next = prev next next
10. Else move to the prv = temp
11. Move the temp to the next element

**// POSITION VALUE**

1. Same as the position change the k to the element
2. Head.data == el
3. Same as the head.data

**INSERTION OF LL**

**// HEAD**

1. Base condition head is null
2. Create the new node
3. If not create the simple node called temp
4. Point the temp to the head

**// TAIL**

1. Base condition if head Is null
2. Create the new node and point to null
3. Go till the last that is temp.next != null
4. Create the node and it’s next points to the null
5. Temp;s next points to the nn

**// POSITION**

1. Base condition head is null and k ==1
2. Create the new node and points to null
3. If the length is greater than 1 call the INSERT HEAD FUNCTION
4. If the k I tail call the INSERT TAIL FUNCTION
5. If k is between
6. Keep the count 0
7. Go till the last temp != null
8. Increase the count by 1
9. If count == k-1 that Is befor the kth node
10. Create the new node with the value
11. Remember nn’s next points to the temp’s next
12. Temp’s next points to the nn
13. Else temp = temp.next

**// POSITION VALUE BEFORE**

1. Base condition head is null not k ==1 becoz value Is found for the null
2. If value is in the head head’s data = given data
3. Create the new node points to the head
4. While temp’s next != null
5. Inside the loop temp’s next data == val
6. Create the node and pontes to the temp’s next
7. Temp’s next points to the nn
8. Else the temp = temp.next

**3rd Lecture**

TOPICS COVERED

ARRAY TO DLL

**DELETION OF DLL**

HEAD OF THE NODE

TAIL OF THE NODE

KTH VALUE OF THE NODE

KTH VALUE OF THE NODE

**INSERTION OF DLL**

HEAD OF THE NODE

TAIL OF THE NODE

KTH POSITION

KTH VALUE POSITION

**// ARRAY TO DLL**

1. Use the first element as the head
2. Remember the head using prev variable
3. Loop till arr.length
4. Create the new node called temp
5. Point the temp’s next to the null
6. Point the temp’back to the prev
7. Prev,s next points to the temp
8. Return the new head or the new DLL

**// DELETION OF HEAD**

1. Base codition
2. Check for the null or single element
3. Return null
4. Remember the head use the prev variable
5. Move the head to head’s next
6. Point the prev’s next to the new head
7. Head’s back points to the prev
8. Return the new head

**// DELETION OF TAIL**

1. Base condition check
2. Head == null return null
3. Head’s next == null (i,e) last element or single element
4. Return null
5. Remember the head using prev variable
6. Traverse till last element
7. Temp = temp’s next
8. Prev points to the temp’s back
9. Prev’s next points to the null
10. Temp’s back = null;

**// DELETION OF KTH POSITION**

1. Initialize the variables front and back to head
2. Base condition check
3. Head == null or head’s next == null
4. Return null
5. K lies between head and tail node
6. Initialize the count = 0
7. Traverse till last
8. Increment the count by 1
9. If count equals to the k
10. Prev = temp’s back
11. Front = temp’s next
12. Prev’ next points to the front
13. Front’s back points to the prev
14. Temp’s next = null
15. Temp’s back = null
16. Else move the temp to the temp’s next
17. Return new head
18. If k is tail node (i,e) Dll have the prev and front element
19. Traverse till the second last element
20. Front points to the temp’s next
21. Front’s back points to the null
22. Temp’s next points to the null
23. Return the new head

**// DELETION OF GIVEN VALUE**

1. First condition is the give value is never head
2. Same as before

**// INSERTION OF HEAD NODE**

1. Create the new node called temp with the value
2. Temp’s next points to the head
3. Head’s back points to the temp
4. Temp’s back points to null
5. Return new head called temp

**// INSERT AT TAIL NODE - AFTER**

1. Base condition head == null
2. Create the new node and points next and back to null
3. If head’s next == null that is one node
4. Create the new node with the value
5. Nn’s next points to the null
6. Head’s next points to the nn
7. Nn’s back points to the head

**// INSERT AT KTH POSITION - BEFORE**

1. Base condition head == null k ==1 : length = 0
2. Create the new node with the value and points it’s next and back to null
3. k == 1 : length = n
4. create the new node with the value
5. temp’s next points to the head
6. head’s back points to the temp
7. temp’s back = null
8. if k lies between head and tail node
9. initialize prev front count = 0
10. traverse till last
11. count ++
12. If count == k that is temp is on the kth position
13. Point the prev to the temp’s back
14. Point the front to the temp’s next
15. Create the new node with the value
16. Nn’s next points to the front
17. Front’s back points to the nn
18. Prev’s next points to the nn
19. Nn’s back points to the prev
20. Return the new head

**// INSERT VALUE AT THE GIVE NODE – BEFORE**

1. Same as the kth position one difference is temp.data == given data

**LECTURE 4**

TPOIC COVERED

**REVERSE THE DLL**

**//REVERSE THE DLL - BRUTE FORCE**

1. Using stack only swapping the data not the links
2. Assign the temp equals to the head
3. Traverse till the last element
4. Push all the data into the stack
5. Temp moves to temp’s next
6. Reassign the temp = head
7. Traverse till the last
8. Store the temp data = stack.peek (first element in the stack)
9. Pop it out
10. Temp moves to temp’s next

**// REVERSE THE DLL – OPTIMAL**

1. Swapping the link
2. Assign the last = null current = head
3. Traverse till the last element
4. Remember the last that is current back befor swapping
5. Currents back points to the current next (Reverse)
6. Current next points to the last
7. Current moves to the current back (After swapping)
8. Return the last bach as new head

**LECTURE 5**

TOPIC COVERED

**ADD TWO NUMBERS IN THE LL**

1. Create the dummy node assign it value to -1
2. Assign the current to dummy node
3. Assign t1 = LL1
4. Assign t2 = LL2
5. Initialize carry to 0
6. Traverse t1 t2 not equal to the null
7. Sum = carry
8. Check if t1 is not null add the sum + t1.data
9. Check if t2 is not null add the sum + t2.data
10. Create the new node and put the sum data modulo 10
11. If there any carry by last digit concept /10
12. Current next points to the new node
13. Check if t1 t2 is not null move t1 t2 to the next node
14. Out side while loop add the carry if have the value create the nn and put the value
15. Point the current next to the nn
16. Return the dummy next as the head;

**LECTURE 6**

TOPIC COVERED

**MIDDLE OF LL**

**// BRUTE FORCE**

1. Check for the length even or odd
2. If even call even length function
3. If odd call odd length function
4. Even function :
5. Val = n/2 it’s integer and floor decreasecount - 0
6. Traverse till the last
7. Count ++
8. If count == val
9. Return temp’s next
10. Else move the temp = temp.next
11. Odd function :
12. Val = n/2 ceil value increase
13. Count = 0
14. Traverse till the last
15. Count ++
16. If count == val
17. Return the temp
18. Else move the temp = temp’s next

**// OPTIMAL**

1. Initialize slow = head
2. Initialize fast = head
3. Traverse till the fast.next != null and fast != null
4. Slow to one move
5. Fast to double move
6. Return slow

**LECTURE 7**

TOPIC COVERED

**REVERSE LL ITERATIVELY AND RECURSIVELY**

**// ITERATIVERLY – BRUTE FORCE**

1. Using data only
2. Initialize the stack and temp = head
3. Traverse till the last not null
4. Push the temp’s data to the stack
5. Move the temp to temp’s next
6. Reinitialize the temp to head
7. Traverse till the last not null
8. Pop the temp.data at the peek of the stack
9. Temp = temp’s next

**// ITERATIVELY – OPTIMAL**

1. Initialize the prev = null
2. Temp = head and front = head
3. Traverse till the last not null
4. Front = temp’s next
5. Temp’s next points to the prev
6. Move the prev to the temp
7. Move the temp to the front

**RECURSIVELY**

1. Base condition head == null or head.next == null
2. Return the head
3. New head = call the reverse function recursively with the head’s next
4. Remember the front = head’s next
5. Front’ next points to the head
6. Head’s next points to the null
7. Return the new head

**LECTURE 8**

TOPICS COVERED

**DETECT THE LOOP**

**// BRUTE FORCE**

1. Using hash map
2. Initialize the hasp map node is the key and integer is the value node means full object
3. Initialize the temp = head
4. Traverse till the last not null
5. Check for the one step like if the temp’s key is already in the hash set
6. If not put the node in the hash set
7. Move the temp to the temp.next

**// OPTIMAL CODE**

1. Using slow and fast pointers
2. Initialize the slow = head
3. Initialize the fast = head
4. Traverse the fast not equal to the null and fast’s next not equal to the null
5. Slow moves to the one point
6. Fast moves to the double point
7. If slow == fast means it is in the same loop of the element
8. Return true
9. If not is in equal means it is linear data structure return false

**LECTUER 9**

TOPICS COVERED

**STARTING POINT OF THE LOOP**

**// BRUTE FORCE**

1. Using hash map
2. Initialize the temp = head and hast map set
3. Traverse till the last that is temp != null
4. If check the node is already in the hash set
5. Return the temp
6. If not put the temp to the hash set
7. Move the temp to the next
8. Return null if not present

**// OPTIMAL CODE**

1. Using slow and fast pointer
2. Traverse till the fast is not null and fast’s next not null
3. Slow moves to the one point
4. Fast moves to the double point
5. If slow is equal to the fast there is a loop
6. 2nd job is to find the starting point
7. Initialize the slow equal to the head
8. Traverse till slow is equal to the fast
9. If not move the slow to one and fast to one
10. If not means return slow
11. If not outside the loop
12. Return the null

**LECTURE 10**

TOPIC COVERED

**LENGTH OF THE LOOP**

**// BRUTE FORCE**

1. Using hashing
2. Initialize the count = 0 and temp to head
3. Traverse till the temp is not null
4. Before putting into the hash check one time if it is exits
5. If not put the element into the hash and move to the temp to the temp’s next
6. If the element is present before the in the hash
7. Initialize the dum to the temp
8. Move the temp to the next and initialize the count to 1
9. Traverse till the temp is not = dum
10. Increment the count by one
11. Move the temp to the temp’s next
12. Return the count
13. If there is no loop return the null

**// OPTIMAL CODE**

1. Using slow and fast poiners
2. Initialize the slow and fast to head
3. Traverse till fast not equal to the null and fast’s next not equal to the null
4. Move the slow to slows next
5. Move the fast to fast’s next.next
6. If slow and fast are in the same position indicated that there Is a loop
7. Count = 0
8. Move the slow to the next and count to 1
9. Traverse till slow and fast are in the same position
10. Increment the count by 1
11. Return the count
12. If no loop is detected return the null

**LECTURE 11**

TOPICS COVERED

**PALINDROME OR NOT**

**// BRUTE FORCE**

1. Using stack
2. Initialize the stack and temp to the head
3. Traverse till the temp is not null
4. Push all the element in the stack
5. Reinitialize the temp to the head
6. Traverse till the temp is not null
7. Check the temp’s data is not equal to the stack’s peek element
8. Return false
9. If not move the temp to temp’s next
10. Outside the loop return the true

**// OPTIMAL CODE**

1. Using slow and fast pointers
2. 1st step finding the half
3. 2nd step revere the ll
4. 3rd step comparing the value
5. 4th step again reverse the ll
6. Initailize the slow and fast to the head
7. Traverse till the fast’s next not equal to the null and fast.next.next not equal to the null
8. Move the slow and pointers
9. Initialize the newhead = reverse function(slow’s next)
10. First to head and second to the new head
11. If first’s data != second’s data
12. Again return reverse the ll and return the false
13. If ll is palindrome reverse the ll and return the true

**12 LECTURE**

TOPIC COVERED

**SEGREGATE THE EVEN AND ODD INDEX NODE**

**// BRUTE FORCE**

1. Initailize the temp to head and i = 0 two arraylist
2. If I is even push the temp’s data to the even arraylist
3. Else put the temp’s data to the odd arraylist
4. Reinitialize the temp to the head
5. Index to 0
6. Traverse till the temp is not last
7. Get the temp’s data from the respective odd array list once it is null
8. Get the temp’s data from the respective even array list
9. Return the head;

**13 LECTURE**

TOPIC COVERED

**REMOVER NTH NODE FROM BACK**

**// BRUTE FORCE**

1. Using reverse technique
2. Base condition head is not ot head.next is not null
3. Initialize the count = 0, temp = head , prev = head
4. If k ==1 move the head to head’s next;
5. Write the reverse function return the new head use the store variable
6. Traverse till the last temp’s next not null
7. Increment the count ++
8. If count is equal to the n
9. Point the prev’s next to the temp’s next
10. Reverse the function and return the new head
11. Return the head

**// BRUTE FORCE ME**

1. Count the length
2. If count == n means head
3. Return the new head
4. If not store the res by count – n
5. Traverse till the last element
6. Decrease the result by -1
7. If res == 0 that means that temp is in the prev node to the delted
8. Remmener the temp and point the temp’s next to the deletetnode next
9. Else move the temp
10. Return the head

**// OPTIMAL CODE**

1. Using slow and fast pointer for n times
2. Initialize the slow to head
3. Traverse till the fas is nextt is not null that is not the tail
4. Move the slow to next and fast to next
5. Rem the slow’s next
6. And point the slow’s next to the slow’s next of next
7. If fast is == null that is in the last element and that in head
8. Return the head of next becoz the head is deleted
9. Else return the previous head

**14 LECTURE**

**DELETE MIDDLE NODE OF LL**

**// BRUTE FORCE BY ME**

1. Count the length of the LL and store it in the count variable
2. If count is even function call the delete even function
3. Else call the odd delete function
4. Even function call Initialize the value by dividing the value and floor of that number
5. Initialize the temp to head and prev = null
6. Traverse till the last in not null
7. Decrease the value by 1
8. Check the value is zero
9. Preserve the front to temp’s next
10. Prev’s next points to the front return the head
11. Else move the prev to temp and temp to temp’s next
12. Same for odd delete function use floor instead of the ceil us the only difference

**// BRUTE FORCE BY STRIVER**

1. Find the length of the ll
2. Find the middle by length /2 + 1 of integer division
3. Initialize the temp to head
4. Traverse till the not null
5. Decrease the res by 1
6. If Check the res == 0
7. Preserve the front by pointing temp’s next
8. Prev’s next points to the front
9. Return he head
10. Else move the prev to temp and temp to temp’s next
11. Return null

**// OPTIMAL CODE**

1. Using slow and fast pointers
2. Initialize the slow and fast to head prev , front = null
3. Traverse till the last fast is not null and fast != null
4. Preserve the prev n=by pointing the slow
5. Move the slow to slow’s next
6. Move the fast to two next
7. After pointing the middle node
8. Front points to the slows next
9. Prev’s next points to the front
10. Return head

**15 LECTURE**

**SORT THE LL**

**// BRUTE FORCE**

1. Using array list
2. Initialize the temp to head
3. Traverse till the last temp is not null
4. And add the temp’s data to the arraylist
5. Move the temp to temp’s next
6. Sort the arraylist by using built in function
7. Reinitialize the temp to head and I = 0
8. Traverse till the temp is not null and I is less then arraylist size
9. put the temp’s data from the respective arraylist data
10. move the temp to temp’s next
11. return head

**// OPTIMAL CODE**

1. using merge sort
2. 1st step is base condition
3. Head is null or head’s next Is null
4. Return head
5. 2nd step finding the middle use the middle variable
6. Finding the middle function there is a difference is fast = head.next
7. 3rd step is separate by pointing
8. Left head is head
9. Right head is middle’s next
10. Separate by null is middle’s next is null
11. 4th step is dividing the right and left
12. By using the variable(lefthead)
13. By using variable(rigthhead)
14. 5th step is returning the merge list is sorting
15. Create the dummy node by -1
16. Point the temp to dummy
17. Left is not null && right is not null
18. If left’s data <= right’s data is equal
19. Point the temp’next to left
20. Move the temp to left
21. And move the left to left’s next
22. Else do it same for the right
23. After left and right is reaching null and there is a element
24. If left use else use the right previous step

**16 LECTURE**

**SORT O 1 2 IN LL**

**// BRUTE FORCE**

1. Initialize the three arraylist zero one two
2. Initialize the temp to head
3. Traverse till the temp is not null
4. Put all the data in the respective arraylist
5. That is o means zero arraylist
6. I means one arraylist
7. 2 means two arraylist
8. Reinitialize the temp to head and initialize the I = 0
9. And calculate the respective arraylist size
10. For zero Traverse till the last and and I less than n0 that is size
11. Put the temp’s data in the ll
12. Reinitialize the I = 0
13. For one Traverse till the last and and I less than n1 that is size
14. Put the temp’s data in the ll
15. Reinitialize the I = 0
16. For two Traverse till the last and and I less than n2 that is size
17. Put the temp’s data in the ll
18. Return head

**17 LECTURE**

**FINDING INTERSECTION POINT OF LL**

**// BRUTE FORCE**

1. Using hasing map data structure
2. Initialize the temp till not null and hash
3. Put all the node in the hash map data structure
4. Reinitialize the temp to head
5. Traverse till the last
6. If check the map already contains the key
7. Return that temp
8. Else traverse tull the last
9. If any node present return temp

**// OPTIMAL CODE**

1. Using two pointers
2. 1st step count the length of the two ll
3. Initialize the resA for LL1 and resB for LL2
4. If res A greater than zero that is n1 is larger
5. Traverse till the resA is 0
6. Move the temp to temp’s next
7. Else move the temp to temp’s next that is n2 is greater
8. 3rd step traverse t1 abd t2 is not null
9. If t1 == t2 means they are in the intersection point
10. Return that temp
11. Else return the null

**18 LECTURE**

**ADD ONE TO LL**

**// BRUTE FORCE**

1. 1st step reverse the LL using reverse function
2. Initialize the temp to head,rem,sum = 0,carry = 1
3. Traverse till the temp is not null
4. Add the temp’s data to the carry
5. Get the rem by %
6. Get the last quotient by / 10
7. Put the temp’s data to the rem
8. Move the temp to temp’s next
9. If there is carry > 0
10. Reverse the LL
11. Create the new node and put the carry’s data
12. Point the nn.next to head
13. Return the nn
14. Outside the while loop reverse the LL
15. Return the head

**19 LECTURE**

**DELETE ALL OCCURRENCE**

**// BRUTE FORCE**

1. Initialize the temp to head
2. Traverse till the temp is not null
3. Check the temp.data == x
4. Also check for temp== head
5. Update the head = head.next
6. Outside the loop
7. Initialize the next = temp.next
8. Prev = temp.back
9. If next is not null meand one link update that is nn.prev = prevnode
10. If prev is not null there is next node is null one link swap update that is prev.next = nn
11. Else move the temp to temp’s next
12. Return head

**20 LECTURE**

**PAIR SUM OF LL**

**//OPTIMAL CODE**

1. Initialize the left to head right to tail by finding the tail
2. Hint is arrayinside the arraytlist
3. Traverse till left data is less than right’s data
4. If left’s data + right’s data == target
5. Initialize the arraylist means pair
6. Pair.add(left.data)
7. Pair.add(right.data)
8. Add the result. Add(pair)
9. Move the left = left.next
10. Right to right.prev
11. If sum > target move the right to prev
12. Else move the left to left’s next
13. Return result
14. ArrayList<ArrayList<Integer>> result = new ArrayList<>(); hint

**LECTURE 21**

**REMOVER DUPLICATES**

**// BRUTE FORCE**

1. Initialize the temp to head;
2. Traverse till the temp Is not null
3. Check if temp’s data == temp’next data if yes
4. Point the temp next to temp.next.next
5. Else move the temp totemp next
6. Return head

**// OPTIMAL CODE**

1. Initialize the temp to head
2. Traverse till the temp is not null AND Temp’s next is not null
3. Nn points to temp’s next
4. While nn is not null nad nn.data == temp’s data if yes means
5. Move the temp to temp’s next
6. If not temp.next points to the nn
7. If nn is not null nn.prev points to the temp
8. Else move the temp to temp’s next
9. Return head

**22 LECTURE**

**REVERSE BY K GROUP**

**// OTIMAL CODE**

1. Ist step finding the kth node
2. If kth is null
3. Check the prev.next = temp
4. Close the that prev case
5. And return break
6. 2nd step separate the ll
7. Nn poita to the kth next
8. Kth next points to the null
9. Reverse the separated ll
10. If temp == head
11. Update head = kth node
12. Else prev.next = kthnode
13. Prev = temp
14. Temp = nn
15. Retrun head;

**23 LECTURE**

**ROTATE LL BY K**

**// brute force**

1. First find the length and tail for the the problem
2. Length initialize the 1 becausse the tail is not going last
3. Base condition if head Is null nad head.next = null or k ==0
4. Return head
5. If k is multiple of length return head
6. If not get the k by k % len
7. Tail.next = head
8. Find the new tail by find tail function
9. Cnt = 1 and traverse till the temp is not null
10. Check the cnt is equal to the k
11. Break else move the temp to temp’s next
12. Increament the count by one
13. Return the temp
14. If not head = nn.next
15. Nn.next = null
16. Return head