5-9-24

Sorting

Page No.
Date

Bubble Sort:

A Simple Comparison-based sorting algorithm that repeatedly steps through the list, Compares adjacent elements, and swaps them if they are in the wrong order.

Example:-

array

0	1	2	3	4
+2	0	45	11	-9

temp = 2

i = 4 (indexNum.(n-1))

j = 0 + 1 = 1 (increase index Number step by step)

-2 0 45 11 -9

array[j] > array[j+1]

⇒ Working left to Right and J + +

⇒ Working slow.

temp = a[i] = 2 (temp)

a[i] = a[j] = 0 (temp)

a[j] = temp

2

Note:-

Also has a time Complexity of $O(n^2)$ but is often slower than Selection Sort due to its repeated swapping mechanism, even though it can be optimized with a flag to detect sorted arrays.

Page No
Date

Code

```
#include<stdio.h>
int main(){
    int i,j, temp, a[10], n;
    Printf("How many elements you want:\n");
    Scanf("%d", &n);
    Printf("Enter %d element.\n", n);
      For(i=0; i<n; i++)
    Scanf("%d", &a[i]);
    For(i=0; i<n; i++)
    {  for(j=i+1; j<n; j++)
      {  if(a[i] > a[j])
        {
          temp = a[i];
          a[i] = a[j];
          a[j] = temp;
        }
      }
}
Printf(" after Shorting\n");
 For(i=0; i<n; i++)
 {
   Printf("\n%d", a[i]);
 }
 return 0;
 }
```

Output

How Many elements you want:
5
Enter 5 element.
5
3
2
1
4

After Sorting

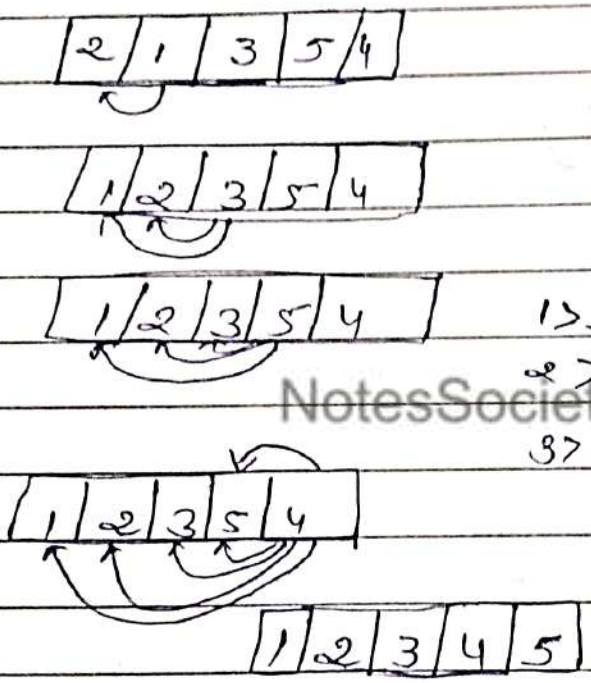
1
2
3
4
5

Insertion Sort:

a Simple Sorting algorithm that works by building a Sorted array one element at a time.

→ It is Considered an "in-Place" Sorting algorithm, meaning it doesn't require any additional memory space beyond the original array.

⇒ We Start with Second element of the array as first.



→ Working Right to left and J--.

→ Working Fast.

Note:

Generally the faster for small or nearly Sorted datasets, with a time complexity of $O(n^2)$ in the average and Worst Cases, but performs better on Partially Sorted data.

```
#include<stdio.h>
int main()
{
  int i, j, temp, a[10], n;
  printf("How many elements you want\n");
  scanf("%d", &n);
  printf("Enter %d elements\n", n);
  for (i=0; i<n; i++)
  scanf("%d", &a[i]);
  for (i=0; i<n; i++)
  { j=i;
  while (j>0 && a[j-1]>a[j])
  {
    temp = a[j];
    a[j] = a[j-1];
    a[j-1] = temp;
    j--;
  }
} printf("after shorting\n");
  for (i=0; i<n; i++)
{
  printf("\n%d", a[i]);
}
  return 0;
}
```

output

How many elements you want
5
Enter 5 element
3
2
5
1
7
after Shorting
1
2
3
5
7

24-9-24

Page No.
Date

DSA :-

DSA Stands for Data Structures and Algorithms.

→ it's the study of How to Solve problem using Step-by-step methods. The ultimate goal is to make programs run faster and less memory. Understanding DSA helps in building better Software and Solving Complex problems.

State :

i) Speed : it helps programs run faster.

ii) Solving problems : it gives you way to fix tricky issues.

iii) Saving Space : good data organization uses less memory.

iv) Better Performance : it makes apps work well with lots of data.

v) Job interview :- Many tech companies ask DSA questions, so it help you get hired.

vi) Basic knowledge :- It's the foundation for learning more about Computers and programming

- in short, DSA helps you write better and faster programs!

2. Four operations performed on DSA:

i) Insertion: Adding new data to a structure, like putting a new item in a list.

ii) Deletion: Removing data, such as taking an item out of list.

iii) Searching: finding specific data, like looking for a name in a list.

iv) Traversal: Going through all the data to see or use each item, like reading every name in a list.

3. Linear and Non-linear:

Linear:

Data is in a straight line.

Ex:- Array or List

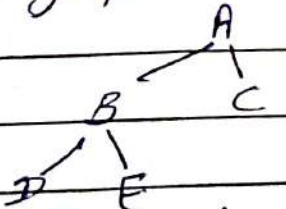
[5, 10, 15]

You can easily go from one number to the next, like counting.

Non linear:-

Data is present in a more complex way, not in a straight line.

Ex: trees or graph



So, linear is like a list, and non-linear is like branching structure.

- Difference between Linear and Non linear:

i) Structure
Linear: Data is in a straight line (like a list)
Non linear: Data is branched or connected (like a tree)

ii) Access Method
Linear: you access one by one.
Non linear: you can jump around to different parts.

iii) Examples
Linear: Arrays, linked lists.
Non linear: Trees, graphs.

iv) Memory Usage
Linear: Usually uses less memory and is easier to manage.
Non linear: Can use more memory due to complex connections.

v) Complexity:
Linear: Simpler to understand and use.
Non linear: More complex but allows for better relationships between data.

- linear is straightforward, while non-linear is more flexible and complex!

- Sorting and types:

Sorting:
The process of arranging data in a specific order, usually from smallest to largest.

Types of sorting:

- Bubble Sort: Compares adjacent / nearest items and swaps them if they are in the wrong order. it keeps repeating until the whole list is sorted.

- Selection Sort: finds the smallest item in the list and moves it to the front. it then looks for the next smallest and repeats the process.

- insertion Sort: Builds a sorted list one item at a time by taking each new item and placing it in the correct position.

- Merge Sort: Divides the list into smaller parts, sorts those parts, and then merges them back together in order.

- Quick Sort: Picks a "pivot" item and sorts the other items into two groups: those smaller than the pivot and those large, then sorts those groups.

Date

Searching and types:

The process of finding a specific item or value in a list or collection of data.

Types:

i) Linear Search: Checks each item one by one until it finds the target. It's simple but can be slow for large lists.

ii) Binary Search:- Working only on Sorted lists. It repeatedly divides the list in half, Checking the middle item to find the target faster.

iii) Jump Search:- Jumps ahead/next by a fixed number of steps and then does a linear search within a smaller range. it's faster than linear but needs a Sorted list.

iv) Interpolation Search:- Similar to binary search but better for uniformly distributed data. It estimates where the target might be and Searching around the point.

Page No.
Date

⇒ Time Complexity :

A way to describe How the running time of an algorithm Changes as the size of the input increase.

⇒ it help us understand if a algorithm is fast or slow.

⇒ we can say its time Complexity is "O(n)" n is number of item.

⇒ Space Complexity :

Space complexity Measures how much memory an algorithm needs as the size of the input increase. it helps us understand how much extra space the algorithm will use while running.

- if an algorithm only uses a small amount of memory
  → "O(1)" (constant space)

- if it needs more memory that grows with the number
  - "O(n)" (linear space)

Date

⇒ hierachy to classify DSA :

1 Data Structures.

- Linear Data Structures
  - i) Arrays
  - ii) Linked lists
  - iii) Stacks
  - iv) Queues

- Non Linear
  - Trees
    - i) Binary Tree
    - ii) Binary Search Tree
    - iii) AVL Tree
    - iv) Heaps
  - Graphs

2) Algorithms
  - Sorting Algo.
    - i) Bubble Sort
    - ii) Insertion Sort
    - iii) Selection Sort
    - iv) Merge Sort
    - v) Quick Sort
  - Searching Algo.
    - i) Linear Search
    - ii) Binary Search
  - Graph Algo.
    - i) Depth - First Search (DFS)
    - ii) Breadth - First Search (BFS)
    - iii) Dijkstra's Algo.
  - Dynamic Programming
  - Recursion

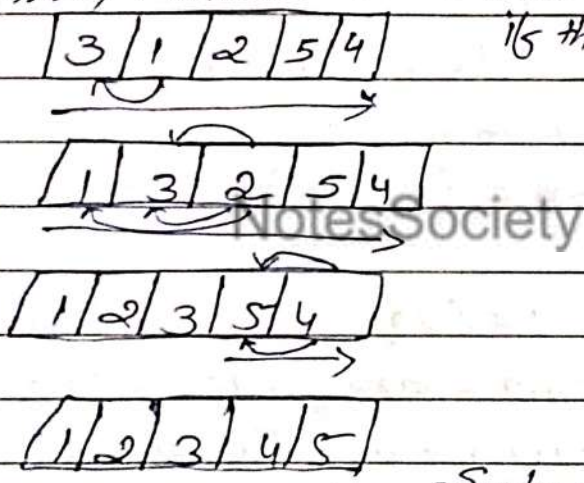
Page No.
Date

Selection Sort :

Selection Sort is a Comparison-based Sorting algorithm. it Sorts an array by repeatedly selecting the Smallest (or largest) element from the unsorted portion and swapping it with the first unsorted element.

→ This process continues Until the entire array is Sorted.

→ Select firstly mini (smallest element) in array and swap Bevious element if this element is small.



Sorted Array.

Note:

Has a time Complexity of $O(n^2)$ in all Cases, but typically has more Comparisons than insertion Sort, making it lower in practice.

⇒ insertion Sort > Selection Sort > bubble Sort

- merge > Quick > insertion > Selection > bubble.

Date

Code :

```
#include <stdio.h>
void main(){
    int n, arr[10], i, j, Position, temp;
printf("How Many Numbers you want to Sort:");
Scanf("%d", &n);
Printf("Okay, Enter %d Numbers:\n", n);
  for(i=0; i<n; i++)
    Scanf("%d", &arr[i]);
for(i=0; i<n-1; i++){
    Position=i;
for(j=i+1; j<n; j++){
  if(arr[Position] > arr[j])
    Position = j;
  }
If(Position != i){
      temp = arr[i];
      arr[i] = arr[Position];
      arr[Position] = temp;
    }
}
  Printf("After Sorting:\n");
  for(i=0; i<n; i++)
  Printf("%d\n", arr[i]);
}
```

Page No.
Date

Output

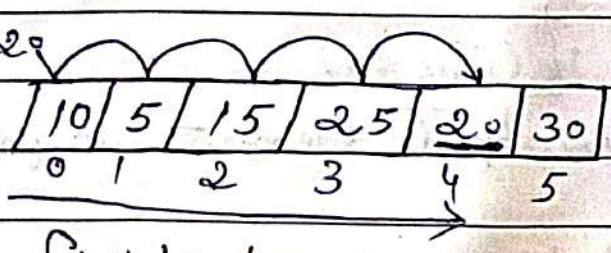
How many Numbers you want to Sort: 4
Okay, Enter 4 Numbers:
2
1
3
4
After Sorting:
1
2
3
4

- Searching algorithms are essential tools in Computer Science used to locate specific item within a Collection of data.
These algorithms are designed to efficiently navigate through data structures to find the desired information, making them fundamental in various applications such as databases, web search engines, and more.

⇒ Searching is the fundamental process of locating a specific element or item within a collection of data.
→ This collection of data can take various forms, such as arrays, lists, trees, or other structured representations.

Linear Search:

It's defined as a sequential search algorithm that starts at one end and goes through each element of a list until the desired element is found, otherwise, the search continues till the end of the dataset.



Code

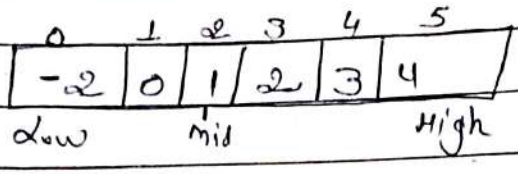
```
#include <stdio.h>
int main() {
        int a[10], i, n, data, flag=0, count=0;
printf("How many elements you want to enter:\n");
scanf("%d", &n);
printf("Enter the %d element\n", n);
for(i=0; i<n; i++)
scanf("%d", &a[i]);
printf("Enter the data which will find\n");
scanf("%d", &data);
for (i=0; i<n; i++)
{
  if (a[i] == data)
{
printf("The Searching element is found on %d position",
                                        i+1);
        flag = 1;
        count++;
printf("The element %d are found on %d time",
                                        data, count);
        // break;
    }
} if (flag=0)
printf("element not found\n");
    return 0;
}
```

output

How many elements you want to enter:
5
Enter the 5 element
3
2
1
5
4
Enter the data which will find
1
The Searching Element is found on 3 position
The Element 1 are found on 1 time.

Binary Search:

It's used in a sorted array by repeatedly dividing the search interval in half.



mid = l + h/2

→ it works by repeatedly dividing the search interval in half until the target value is found or the interval is empty.

Code

```
Scanf("%d", &data);
l = 0
r = n-1;
While (l<=r){
    mid = (l+r)/2;
  if (a[mid] == data){
  Printf("%d element found at position %d\n",
                                data, mid+1);
  return; }
  else if (data > a[mid]){
        l = mid+1;
    } else {
          r = mid-1;
      } }
  Printf("%d element not found\n", data);
}
```

Output

How many elements do you want : 4
Enter 4 elements in Sorted order:
1
2
3
55
Enter the data to find : 55
55 element found at position 4

★ Using Count

```
#include<stdio.h>
void main(){
        int a[100], i, n, data, count=0;
        int l, r, left, right, mid;

printf("How may elements do you want:");
scanf("%d", &n);
printf("Enter the %d elements in Sorted order:\n"
                                        , n);
for(i=0; i<n; i++)
scanf("%d", &a[i]);
printf("Enter the data to find:");
scanf("%d", &data);
l=0;
r=n-1;
```

```
while (l <= r) {
    mid = (l+r)/2;
    if (a[mid] == data) {
        count++; // Counting
        // check duplicates
        left = mid-1;
        while (left >= l && a[left] == data) {
            count++;
            left--;
        }
        // check duplicates
        right = mid+1;
        while (right <= r && a[right] == data) {
            count++;
            right++;
        }
        printf("%d element found at position %d\n", data, mid+1);
        printf("The element %d was found %d times\n", data, count);
        return;
    } else if (data > a[mid]) {
        l = mid+1;
    } else {
        r = mid-1;
    }
}
printf("%d element not found\n", data);
}
```

Page No.
Date

Output

How Many elements do you want: 5
Enter the 5 elements in Sorted order:
1
2
2
2
4
Enter the data to find: 2
2 element found at position 3
The element 2 was found 3 times.

Stack

- Push / Pop / Display

```
#include <stdio.h>
#include <stdlib.h>
#define n 5
        (max size of stack)
int stack[n];
int top = -1;
void push(){
        int x;
        printf("Enter data:");
        scanf("%d", &x);
    if (top == n-1){ → (last index num.)
        printf("Stack overflow\n");
    } else{
        top++;
        stack[top] = x;
        printf("%d Pushed to Stack\n", x);
    }
}
void pop(){
        if (top == -1){
        printf("Stack Underflow\n");
    } else{
        int Popped_value = stack[top];
        top--;
    printf("%d Popped from Stack\n", Popped_value);
    }
}
Void display(){
        if (top == -1){
    Printf("Stack is empty\n"); }
```

```
else { printf("Stack elements:");
    for (int i = top; i >= 0; i--) {
    printf("%d", stack[i]);
    }
    Printf("\n");
}
}
int main() {
    int choice;
    while (1) {
    Printf ("\n 1. Push \n 2. Pop \n 3. Display \n 4. Exit\n");
    Printf("Enter your choice:");
    Scanf("%d", &choice);
    Switch (choice) {
        Case 1:
            Push();
            break;
        Case 2:
            Pop();
            break;
        Case 3:
            display();
            break;
        Case 4:
            Exit(0);
        default:
        Printf("Invalid choice \n");
        }
    }
    return 0;
}
```

Output

1. Push
2. Pop
3. Display
4. Exit

Enter your Choice : 1
Enter data : 5
5 pushed to stack

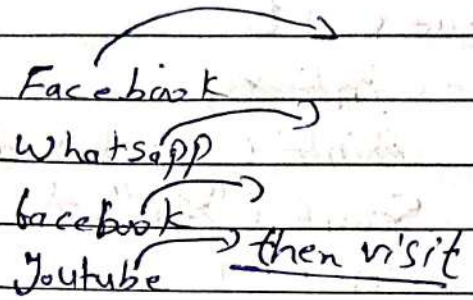
.
.
.

Contn...

Date

→ Stack work on "last in first out / First in last out."

Example: you open first "1. youtube, 2. Instagram, 3. Whatsapp and Last facebook, Suppose you want to check "youtube" so you go back and back and then check last.



→ In stack from last side value push and pop.

Note:

The top variable is initalized to -1. This indicates that the stack is empty.

Page No.
Date

Queue (enqueue, dequeue, Display) Simple.

```
#include <stdio.h>
#include <stdlib.h>
#define n 5

int queue[n];
int Front = -1;
int rear = -1;

void enqueue() {
    int x;
    Printf("Enter data: ");
    Scanf("%d", &x);
if (rear == n-1) {
    Printf("Queue overflow\n");
} else {
    if (Front == -1) {
        Front = 0; // if empty
    }
    rear++;
    queue[rear] = x;
    Printf("%d added to queue\n", x);
}
}

void dequeue() {
    if (Front == -1 || Front > rear) {
    Printf("Queue underflow\n");
} else {
    int dequeued_value = queue[Front];
    Front++;
Printf("%d removed From queue\n", dequeued_value);
```

```
if (front > rear) {
        front = rear = -1 ; // if empty
} } }
void Display () {
        if (front == -1 || front > rear) {
printf (" Queue is empty \n" );
} else {
  Printf ("Queue element : ");
  For (int i = front ; i <= rear ; i++)
{ printf ("%d ", queue [i] );
} Printf ("\n");
} }
int main () {
      int choice ;
While (1) {
printf ("\n 1. Enqueue \n 2. Dequeue \n 3.
        Display \n 4. Exit \n");
Printf ("Enter your choice:" );
Scanf ("%d", & choice );
switch (choice) {
  Case 1;
    enque ();
  break;
Case 2 :
  dequeue ();
  break;
Case 3:
display ();
break;
Case 4:
Exit(0);
default:
Printf (" Invalid choice \n");
}
}
  return 0;
}
```

Output

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice : 1

Enter data : 2

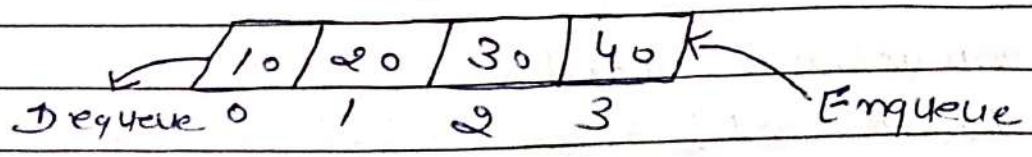
2 added to queue

1. Enqueue
2. Dequeue
3. Display
4. Exit

Queue

Page No.
Date

→ Queue working on "Fifo".



Note: In Queue and Stack item/data is Remove from memory but Not Empty memory /Not del. data only Remove but memory location is present there.

Inline Queue

```
#include <stdio.h>
#define MAX 5

int Queue_arr[MAX];
int Front = -1;
int rear = -1, max;

void insert(){
        int item, max;
  if ((front == 0 && rear == MAX-1) || Front == rear+1)
  { printf("overflow.\n");
    return;
    }

  // if the queue is empty
  if (front == -1) {
        Front = 0; // initialize front and rear
rear = 0; }
  if (rear == max-1)
    rear = 0;
  else {
  rear = (rear+1) % MAX; }
  Printf("Enter an element:");
  Scanf("%d", &item);
  Queue_arr[rear] = item; // insert the item
  Printf("Inserted %d\n", item);
   }
void dequeue(){
        // Check for underflow
  if (front == -1) {
        printf("Underflow.\n");
  return;
   }
```

```
printf("The deleted element is %d\n", Queue_arr
                                        [Front]);
// if the queue becomes empty after dequing
if (front == rear) {
    Front = -1; // RESET The QUEUE
    rear = -1;
}
if(front == max-1) // Wrap around FOR FRONT.
 front = 0;
else
front = front + 1;
}

void display() {
        if (front == -1) {
        printf ("Queue is empty \n");
    return;
}
printf("Queue elements:");

int i = Front;
 while (1) {
 printf ("%d ", Queue_arr[i]); // Print the
                                value at the current positio
if (i == rear) break; // BREAK if we've reached
                        The rear.
i = (i+1) % MAX; // WRAP Around
}
  printf ("\n");
}
```

```
int main(){
    int choice;
    while(1){
    printf("1. Enqueue\n 2. Dequeue\n 3. display\n");
    printf("Enter your choice:");
    scanf("%d", &choice);
switch(choice){
    case 1: //Enqueue
    insert();
    break;

    case 2: //Dequeue
    dequeue();
    break;

    case 3: // display
    display();
    break;
default:
    printf("Invalid choice.\n");
    }
    }

    return 0;
}
```

Page No.
Date

Output

1. Enqueue
2. Dequeue
3. display
Enter your choice: 1
Enter an element:
Inserted 3
1. En..
2. De..
3. Display.
Enter your choice: 2
The deleted element is 3
"
"
"
Enter your choice:

Insert circular queue

```
#include <stdio.h>
#define MAX 5
int Queue_arr[MAX];
int front = -1;
int rear = -1;
void insert() {
    int item;
    // Check for overflow
    if ((front == 0 && rear == MAX-1) || (front
    == rear+1)) {
    printf("overflow.\n");
    return;
    }
        // If the QUEUE is Empty
    if (front == -1) {
    front = 0; // initialize front and rear
    rear = 0; // Also initialize rear
    } else {
        // MOVE REAR TO THE NEXT POSITION
    rear = (rear+1) % MAX; // wrap around using
                           modulo
    }

printf("Enter an Element: ");
scanf("%d", &item);
Queue_arr[rear] = item; // INSERT the Item
printf("Inserted %d\n", item); }

void dequeue(){
        // CHECK FOR UNDERFLOW
if (front == -1){
printf("Underflow.\n");
return; }
```

```
printf("The deleted element is %d\n", Queue_arr
                                        [Front]);
    //IF THE QUEUE become Empty after dep.

    if(front == rear) {
        Front = -1; //Reset the Queue
        rear = -1;
    }else {
            //MOVE Front to The next position
    Front = (front+1) % MAX;
        }
    }
Void display() {
            if(front == -1) { // Queue is empty
    printf("Queue is empty\n");
    return;
    }
printf("Queue elements");
    int i = front;
  while(1) {      //Print the value at the current posi.
        printf("%d", Queue_arr[i]);
  if(i == rear) break; // STOP WHEN we
                                reach the rear
    i = (i+1) % MAX; //MOVE to the next elem.
  }
    printf("\n");
    }
    int main() {
        int choice;
        while(1) {
  printf("1. Enqueue\n 2. Dequeue\n3. Display\n
                4. Exit\n");
```

```
Printf("Enter your choice:");
Scanf("%d", &choice);

switch (choice){
 Case 1:
      insert();
      break;
Case 2:
   dequeue();
   break;
Case 3:
display();
break;
Case 4:
return 0;

default:
      Printf("Invalid choice:\n");
   }
}
return 0;
}
```

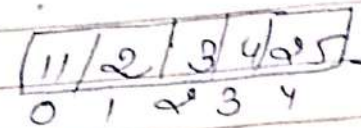
Page No.
Date

OUTPUT

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice : 1
Enter an element : 3
Inserted 3
1. "
2. "
3. "
Enter your choice : 2
The deleted element is 3
1 "
2 "
3 "
Enter your choice : 3
Que is empty
1 "
"
"
"

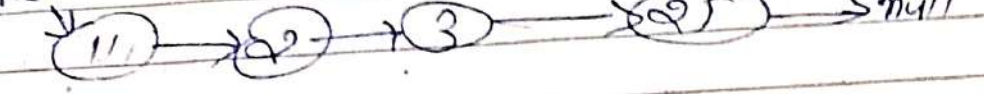
30-9-24

Linked List

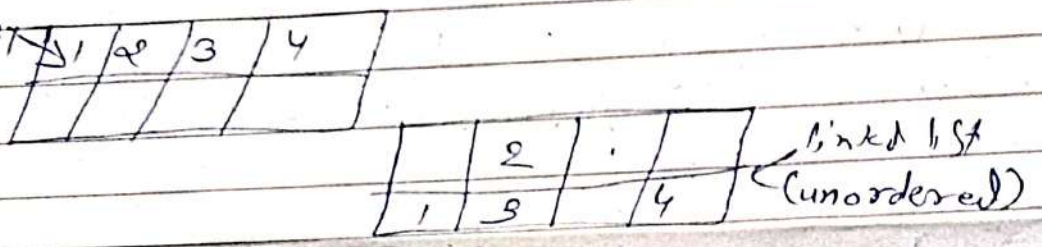


→ array looking like array of indexes but Linked list not looking same

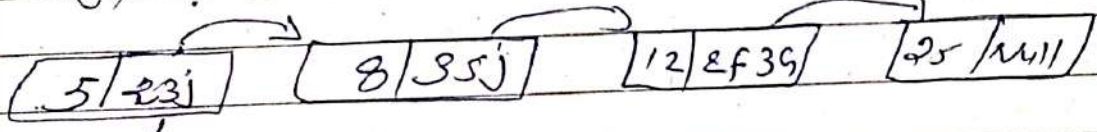
Linked list looking like



→ In array we are place in memory data continusly but in link list not continusly way possible.



Singly linked list.



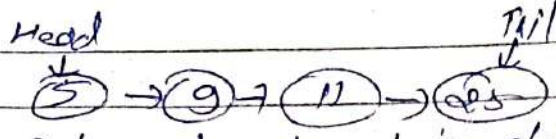
~~Address~~ Hold next elemts

Advantg of linked list

- we are in arry give size first for example a[10] but linked list no need give size first it's adjust dynamically auto allocation new memory.

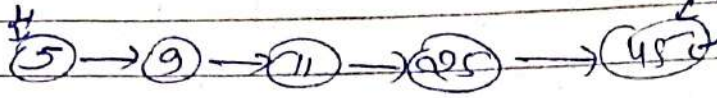
Note: we can use linked list there number is unknown.

- In linked list runtime we can easyly grow and shrink.
- No-pre allocation of space



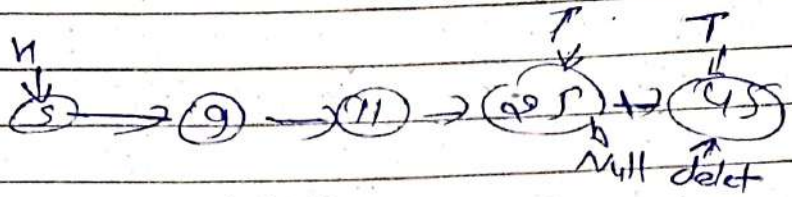
→ insert a element in last.

Insert



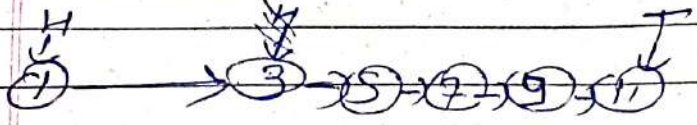
O(1)

Delet

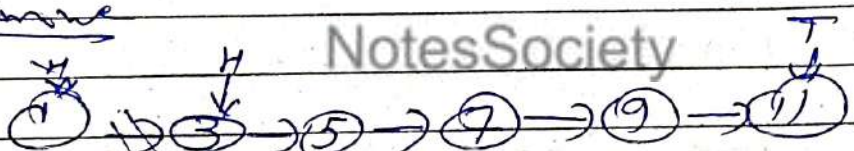


O(n)

Beginning Insert a data

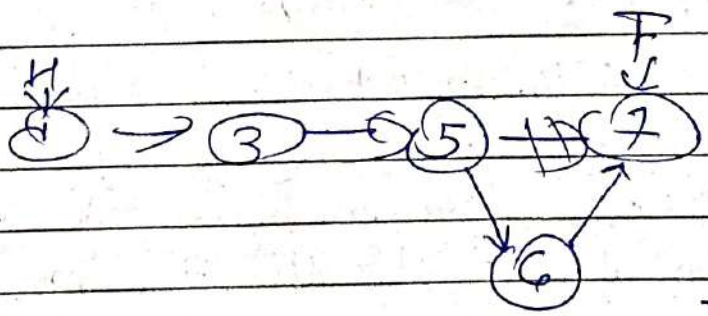


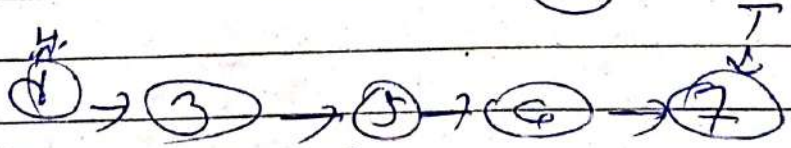
Remove



O(1)

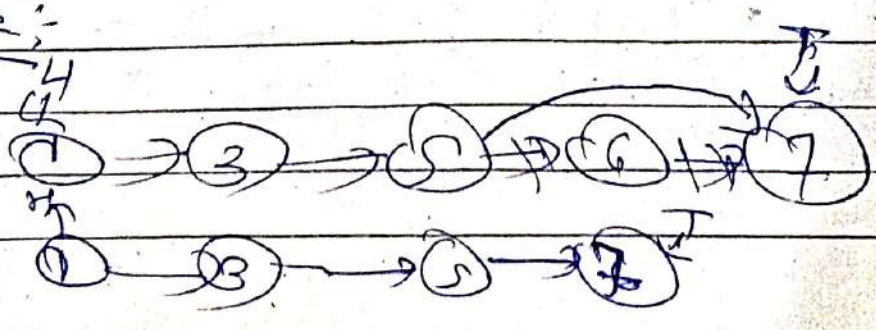
middle





O(n)

Delete:-



Page
Date

linklist

```
#include <stdio.h>
#include <stdlib.h>

Struct node{
        int info;
        Struct node *link;
    };
Struct node *Start = Null, *q;

Void Create (){
        int data;
        Struct node *temp;
        temp = (Struct node *) malloc (Sizeof (Struct node));
        if (temp == Null) {
        Printf ("Memory not allocated \n");
        return;
        }
Printf ("Enter an element:");
Scanf ("%d", &data);

temp -> info = data;
temp -> link = Null;

if (Start == Null) {
        Start = temp;
    } else {
        q = Start;
        While (q -> link != Null) {
        q = q -> link;
        }
        q -> link = temp;
    }
}
```

```
void display ()
{
    if (start == NULL)
    {
        printf ("The list is empty \n");
        return;
    }
    q = start;
    printf ("Linked list elements: ");
    while (q != NULL)
    {
        printf ("%d", q -> info);
        q = q -> link;
    }
    printf ("\n");
}

int main()
{
    int choice;
    while (1)
    {
        printf ("1. Create node \n");
        printf ("2. Display list \n");
        printf ("3. Exit \n");
        printf ("Enter your choice:");
        scanf ("%d", &choice);

        switch (choice)
        {
            case 1:
                create ();
                break;
            case 2:
                display ();
                break;
            case 3:
                exit (0);
            default:
                printf ("Invalid choice \n");
        }
    }
    return 0;
}
```

Page No.
Date

Count element

```
#
#
Struct Node{
    int data;
    Struct Node *link;
};
void Count (Struct Node *Start){
  Struct Node *a = Start;
  Struct Node *q = a;
  int Count = 0;
  while (q != Null){
     Count++;
     q = q->link;
    }
Printf("Count element is %d \n", Count);
}
Struct Node *CreateNode (int data){
  Struct Node *newNode = (Struct Node*) malloc(sizeof
                                    (Struct Node));
NewNode -> data = data;
NewNode -> link = Null;
return Newnode;
  }
  // MAIN FUNCTION
int main(){
  Struct Node *Start = CreateNode(10);
  Start -> link = CreateNode(20);
  Start -> link = CreateNode(30);
Start -> link -> link -> link = CreateNode(40);
  Count(Start);
  return 0;
  }
```

Page No.
Date

Search function link list

```
Struct Node {
    int info;
    Struct Node *link;
    };
void Search (Struct Node *start, int item) {
    Struct Node *q = start;
    int count = 1;
    int flag = 0;
    While (q != Null) {
    if (q->info == item) {
printf ("The item %d found at position %d \n", item, count);
flag = 1;
    return;
    }
    count ++;
    q = q -> link;
    }
     if (flag == 0) {
     printf ("No item found \n");
     }
     }
     // Create NODE
int main() {
Struct Node *start = malloc (sizeof (Struct Node));
start -> info = 10;
start -> link = malloc (sizeof (Struct Node));
start -> link -> info = 20;
start -> link -> link = Null;
// Find node
    int item = 20;
    Search (start, item);
    return 0;
    }
```

Priority Queue

Date

```
Struct node { int prio;
              int info;
              Struct node *link;
            } Front = Null, *q;

Priority ()
        { int p, data;
          Stout node *tmp;
          tmp = malloc ——
          Printf("Enter the data & Priority");
          Scanf("%d %d", &data &p);
tmp -> info = data;
tmp -> prio = p;
  if (Front == Null || p < Front -> prio)
  { tmp -> link = Front;
        Front = tmp;
        return;
  }
q = Front;
while (q -> link != Null && q -> link -> prio < p)
        q = q -> link;
tmp -> link = q -> link;
q -> link = tmp;
}
```