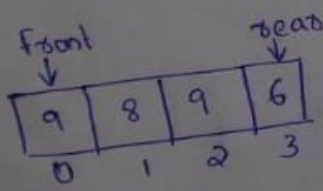
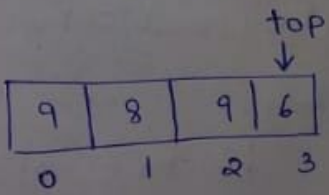
1. ~~A st~~

a). Stack is a linear data structure, it follows LIFO (Last in First out) principle. The elements are inserted and deleted only from one side of list in a stack.

Queue is also a linear data structure, it follows FIFO (First in First Out) principle. The elements are inserted from one side (rear) and elements are deleted from other side (front)

Queue	Stack
→ Insertion operation is called enqueue	→ Insertion operation is push operation
→ Deletion operation is dequeue	→ Deletion operation is pop
→ Insertion occurs at rear of the list and deletion occurs at front of list	→ Both insertion and deletion occurs ~~at th~~ only at one end (top)
→ It follows FIFO (First in First Out)	→ It follows LIFO (Last in First Out)





b) similarities

→ Both binary tree and binary Search tree has root node and follows a hierarchical structure

→ Maximum child nodes is two for both binary Search tree and binary tree.

Dissimilarities

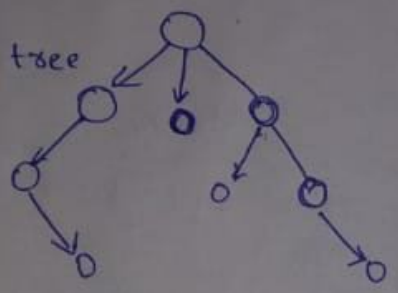
→ Binary search tree has a specific order for the arrangement of data elements, while binary tree lacks a specific order for arrangements of data elements

→ Since Binary Search trees are Sorted binary trees it provide faster insertion, deletion and traversal than binary tree

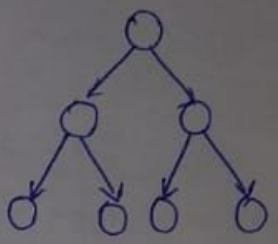
→

→

Binary tree



Binary Search tree



c) Any subset of Universal set in 0 and 1 is called bit string It is used to define a sequence of bits having the values either 0 and 1

eg:- Universal set = $\{1, 2, 3, 4, 5\}$

then

bit string of $U = \{1, 1, 1, 0, 1\}$

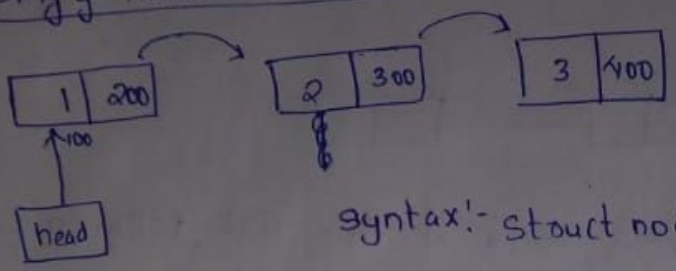
(d) → Make set is used to create a new set whose only member is pointed to x [Make. set(x)]

→ A disjoint set data structure is used to maintain a collection of disjoint set.

→ Union unites dynamic sets that contain x and y. That is creates $S_x \cup S_y$ from S_x and S_y

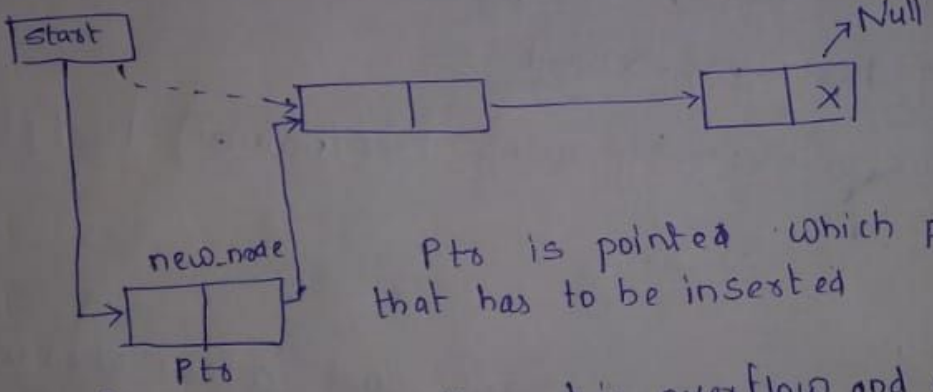
- Findset(x): It returns a pointer to the representation of a set containing x. It also determines which set is in which a particular element is in.

2. Singly Linked List



syntax:- struct node
{
int data;
struct node * next;
}

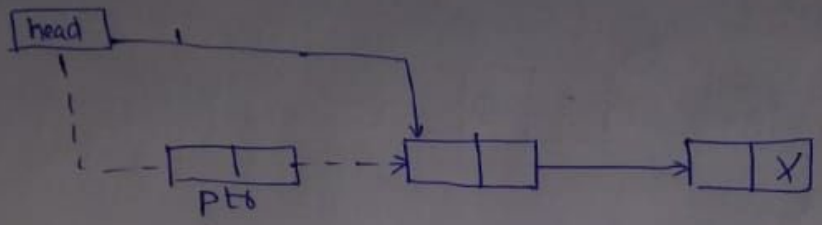
Insertion in beginning



Ptr is pointed which points to the node that has to be inserted

① if ptr == null, then it is overflow and exit the procedure
② Set new_node = pointer and set ptr = ptr → next
③ then make new_node → data = value
④ set new_node → Next = head
⑤ finally, we need to make the new node as the first node of the list by using head = new_node and then exit.

Deletion in Singly Linked List



In order to delete the first node of the list, following steps is needed.

① Check whether head = Null, if yes, it is an underflow, exit the procedure

② First set ptr = head, and point head to the next ~~the head~~ by head = ptr → head

③ Then free the pointer by using ~~free(pointer)~~ free(ptr)

④ then exit.

3. Amortized analysis

It is a method for calculating the cost associated with a data structure. In Amortized analysis we average the time required to perform a sequence of data structure operations over all the operations performed. It considers the average performance of each operation in worst case.

Data structures likes hash tables, disjoint set and splay trees are analyzed using Amortized analysis.

The common technique used for amortized analysis are

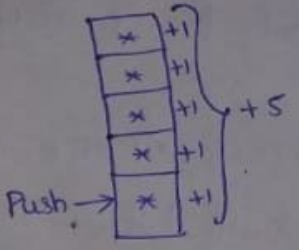
① aggregate method
② Accounting method
③ Potential method.

Accounting Method

In accounting method each operation aissgned a charge called amortized cost. Here some operations may be charged more or less than acctual cost

$$\text{Actual cost} = \frac{\text{No: of steps taken}}{\text{time required ton run}}$$

Eg:- Conider performing push operation in a stack as given below



For each step we are planned to given extra 1 cost

Therefore actual cost = 5, but Amortized cost = $5 + 5(1) = 10$

If an operations, amortized cost exceeds its actual cost, we asign the difference called a credit to specific object in data structure

Credit = Amortized cost - Actual Cost = $10 - 5 = 5$

That is, Amotized cost = Actual Cost + Credit

Credit cannot be a negitive in any operation.

Aggregate Method

In this method, we show that for all n, a sequence of n operations takes a worst case time T(n) in total

In worst case, the average cost / amortized cost per operation = $\frac{T(n)}{n}$

That is, this amortized cost applies to each operation, even there are several type of operations in the sequence.

4. A collision happens when a hash function generate same address for a different key.

There are two collision Resolution techniques

① Open addressing ② Seperate chaining

1] Open addressing

In open addressing the key which cause collision is placed in the hash table itself but the location will be other than its hash address.

If the address is already occupied we will try to inser in thie next location on the hash table. There are 3 methods namely

① linear probing
② Quadratic probing
③ double hashing.

Linear probing

If the address of given hash function is a [already occupied] then move to next location → a+1

Quadratic probing

Colliding keys are store away from intial point to reduce cluster

Double hashing

It uses an idea of applying a second hash function to key when a collision occurs.

2) Seperate Chaining

In this method Linked list is maintained for the element that have the same address. All elements having this same has address will be stored in a seperate linked list and starting address will be i