DS Assignment

NAME:Krithika P Suwarna

SRN: PES1201701369

1. **void sortedInsertDLL(int data) : Given a sorted DLL of integers, write a method that inserts a new node at the proper place in the DLL. Assume HNODE and TNODE are the sentinel nodes of the DLL.**

```
void sortedInsertDLL(DLList *list, int data){
    //printf("HELLO\n");
    int flag=0;
    Node *temp = (Node *) malloc(sizeof(Node));
    Node *toinsert = (Node *)malloc(sizeof(Node));
    toinsert->next=NULL;
    toinsert->prev=NULL;
    toinsert->data=data;
    temp = list->head;
    while(temp != NULL)
    {
        if(temp->data<=data && temp->next!=NULL &&
temp->next->data>=data){
            toinsert->next=temp->next;
            toinsert->prev=temp;
            temp->next=toinsert;
            toinsert->next->prev=toinsert;
            //printf("HELLO\n");
            flag=1;
            return ;
        }
        temp = temp->next;
    }
    if(!flag){
        list->tail->next=toinsert;
        toinsert->next=NULL;
        toinsert->prev=list->tail;
    }
}
```

2. addAfter(int b, int data); Write a method that adds a new node containing data after a node containing the value b.

```
int addAfter(s_list* list,int b, int data){
    int flag=0;
   // printf("HELLO");
    if(list->head==NULL) //empty list case
```

```
        return 0;
    node *temp = (node *)malloc(sizeof(node));
    temp->link=NULL;
    node *toinsert = (node *)malloc(sizeof(node));
    toinsert->data=data;
    toinsert->link=NULL;
    temp = list->head;
    while(temp!=NULL && temp->data!=b){ //DONOT USE ||
        if(temp->data==b)
            flag=1;
        temp = temp->link;
    }
    if(temp!=NULL){
        toinsert->link=temp->link;
        temp->link=toinsert;
    }
    else if(flag==0 && temp==NULL)
        return 0;
    return 1;
}
```

3. Write a RemoveDuplicates() function which takes a list sorted in increasing order and
deletes any duplicate nodes from the list. Ideally, the list should only be traversed once.

```
void RemoveDuplicates(s_list *list){
    if(list->head==NULL)
        return ;
    node *temp=(node *)malloc(sizeof(node));
    temp->data=0;
    temp->link=NULL;
    temp = list->head;
    while(temp!=NULL && temp->link!=NULL){
        while(temp!=NULL && temp->link!=NULL &&
temp->data==temp->link->data ){
            if(temp->link->link==NULL){ //if last node
                node *temp2=(node *)malloc(sizeof(node));
                temp2 = temp->link;
                free(temp2);
                temp->link=NULL;
            }
            else{
                node *temp2=(node *)malloc(sizeof(node));
                temp2 = temp->link;
                temp->link=temp2->link;
                free(temp2);
            }
        }
        temp = temp->link;
    }
}
```

4. replace (int b, int data) ; Write a method that replaces node that contains b with data.

```
void replace(s_list *list,int b, int data){
    if(list->head==NULL)
        return ;
    node *temp=(node *)malloc(sizeof(node));
    // temp->data = 0;
    // temp->link=NULL;
    temp = list->head;
    while(temp!=NULL){
        if(temp->data==b){
            temp->data=data;
            return ;
        }
        temp=temp->link;
    }
}
```

5. An array called twoStacks[n] is used to represent two stacks. Stack1 has top1=0 starting at first element of the array. Stack2 has top2=n starting at the last element of the array. Write algorithm for push, pop, operations for both the stacks.

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 20

int twostack[SIZE];
int top1=-1;
int top2=SIZE;

void push1(int n);
void pop1();
void push2(int n);
void pop2();

int main(){
        int flag=1,elem=0;
        while(flag){
                int opt;
                scanf("%d",&opt);
                switch(opt){
                        case 1:
                                        scanf("%d",&elem);
                                        push1(elem);
                                        break;
                        case 2:
                                        scanf("%d",&elem);
                                        push2(elem);
```

```
                                        break;
                        case 3:
                                        pop1();
                                        break;
                        case 4:
                                        pop2();
                                        break;
                        default:
                                        flag=0;
                }
        }
}

void push1(int n){
        if(top1<top2-1)
                twostack[++top1]=n;
        else
                printf("STACK 1 OVERFLOW!\n");
}

void push2(int n){
        if(top1<top2-1)
                twostack[--top2]=n;
        else
                printf("STACK 2 OVERFLOW!\n");
}

void pop1(){
        if (top1 >= 0 )
           twostack[top1--];
    else
         printf("Stack UnderFlow\n");
}

void pop2(){
        if (top2 < SIZE)
         twostack[top2++];
    else
         printf("Stack UnderFlow\n");
}
```

6. Write a program that reads in a positive integer and prints the binary representation of that integer. Hint: divide the integer by 2 and use stack or queue.

include<stdio.h>

#include<stdlib.h>

#define SIZE 20

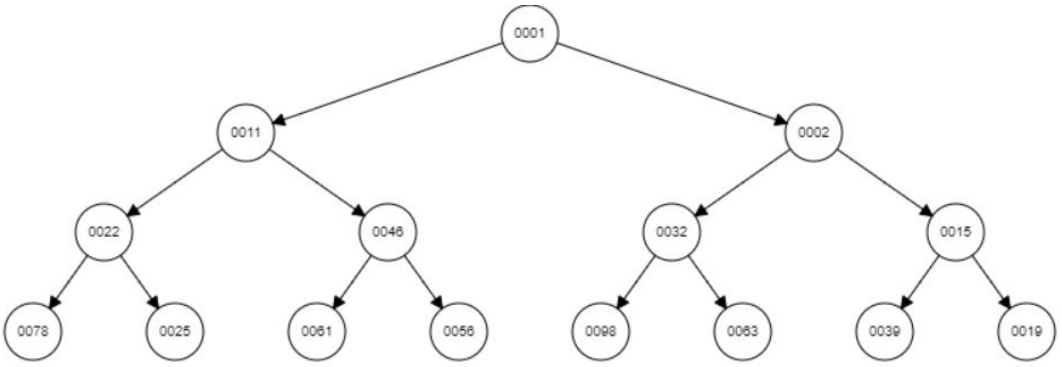
```
typedef struct Stack
{
    int stk[SIZE];
    int t;
}stk;

int empty(stk *s);
int full(stk *s);
int push(stk *s,int i);
int pop(stk *s);
void disp(stk *s);
void convert(int x);
int main(){
    int num,x,i;
    scanf("%d",&num);
    for (i = 0; i < num; i++) {
        scanf("%d",&x);
        printf("%d",x);
        convert(x);
        printf("\num");
    }
}
void intialize(stk *s){
    s->t=0;
}
int empty(stk *s){
    if(s->t==-1)
        return 1;
    else
        return 0;
```

```
}

int full(stk *s){
    if(s->t>20)
        return 1;
    else
        return 0;
}

int push(stk *s,int i){
    if(full(s))
        return 0;
    s->stk[(s->t)++]=i;
    //printf("HERE");
    return 1;
}

int pop(stk *s){
    if(empty(s))
        return 0;
    s->t -=1;
    return 1;
}

void disp(stk *s){
    if(empty(s))
        return ;
    for(int i = s->t; i>=0;i--)
        printf("%d",s->stk[i]);
    //printf("HERE");
}
```

```
void convert(int x){
    if(x==0){
        printf("0");
        return ;
    }
    stk STACK;
    stk *s = &STACK;
    int temp=x;
    intialize(s);
    //printf("HERE");
    while(temp>0){
        //printf("HERE");
        push(s,temp%2);
        temp = temp/2;
    }
    //push(s,1);
    pop(s);
    disp(s);
}
```

7. Construct a min-heap with the following data elements : { 32, 78, 1, 22, 46, 98, 39, 11, 25, 61, 56, 63, 2, 15, 19}. Show the contents of either the heap arrray or the heap tree at each step.

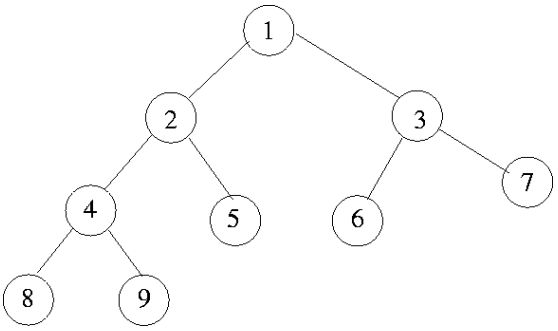
The final min-heap is –



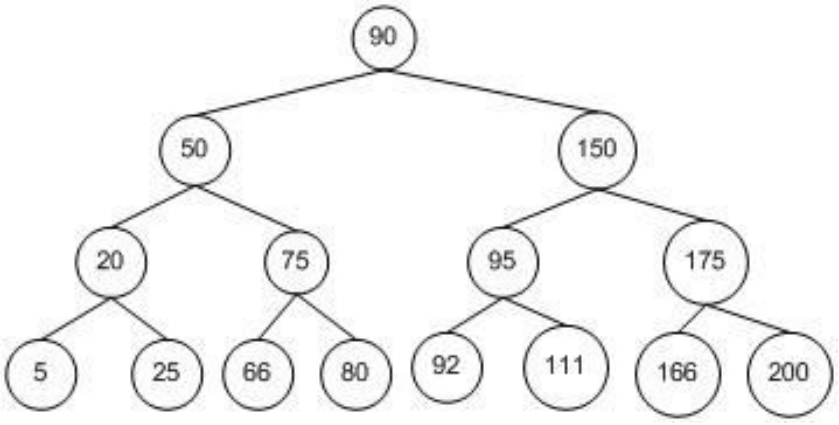
1. 32 in root of the empty heap.
2. Insert 78 at the bottom left of the heap. 78 is greater than 32, so no swapping needed.
3. Insert 1 at the bottom right of the heap. 1 is less than 32, so swap 32 with 1.
4. Insert 22 at the bottom left of the heap. 22 is less than 32, so swap 32 with 22.
5. Insert 46 at the bottom right of the heap. 46 is less than 78,so swap 78 with 46.
6. Insert 98 at the bottom left of the heap. 98 is greater than the other numbers, so no swapping needed.
7. Insert 39 at the bottom right of the heap. 39 is less than 46,so swap 46 with 39.
8. Insert 11 at the bottom left of the heap. 11 is less than 22, so swap 22 with 11.
9. Insert 25 at the bottom right of the heap. 25 is less than 32, so swap 32 with 25.
10. Insert 61 at the bottom left of the heap. 61 is less than 98, so swap 98 with 61.
11. Insert 56 to the bottom right of the heap. 56 is less than 61,so swap 61 with 56.
12. Insert 63 to the bottom left of the heap. 63 is less than 98,so swap 98 with 63.
13. Insert 2 to the bottom right of the heap. 2 is less than 11,so swap 11 with 2.
14. Insert 15 to the bottom left of the heap. 15 is less than 22, so swap 22 with 15.
15. Insert 19 to the bottom right of the heap. 19 is less than 22, so swap 22 with 19.

8. **Give an example of a (a) Proper Binary Tree (b) Binary Search Tree (3) Heap**

   (a) Proper Binary Tree

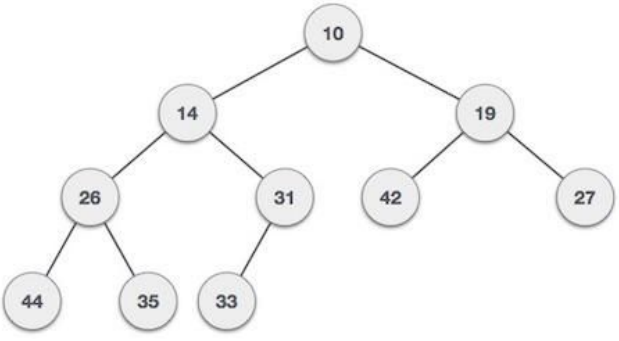


(b) Binary Search Tree



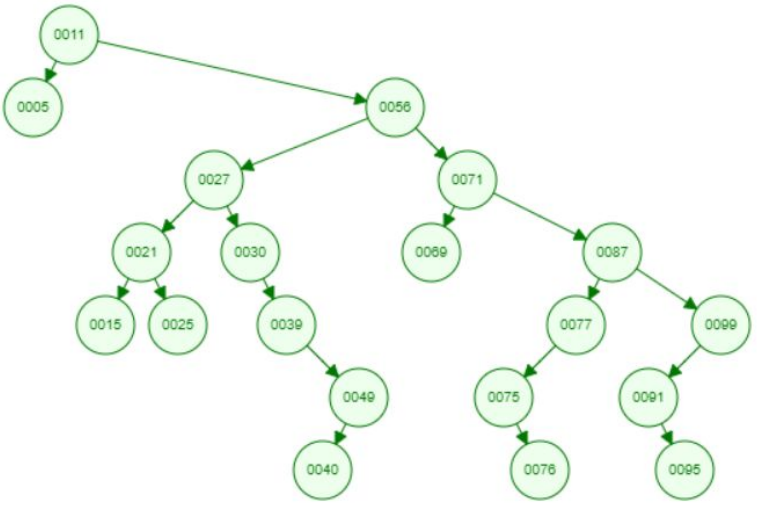
(c)Heap

9. Create a BST the following { 11, 56, 27, 5, 49, 25, 87, 40, 75, 91, 95, 76}. 75, 21, 76

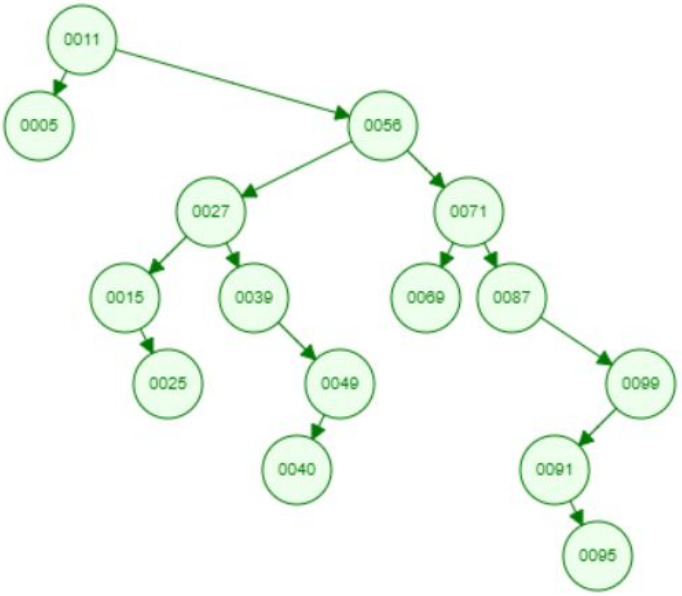


by inserting elements : 30, 21, 39, 71, 77, 99, 15, 69, Delete 30, 77,

After inserting –



After deleting –



10. **Why is Priority Queue implemented using heap performs better than that of array or list?**

When priority queue is implemented using arrays and linked list, moving the elements of the queue back and forth based on priority is tedious and takes a lot of time and memory. Everytime an element is inserted, the elements of the list or array need to be rearranged according to priorty. The same process has to be done with heaps, but it is much easier to do so, and less time consuming as you can use a max or min heap based on your requirement. The element with the highest priority is moved to the front of the queue and the one with the lowest priority is moved behind. Therefore priority queue using heaps performs better.