## Stacks for trees

* /\* A binary tree tNode has data, pointer to left child

   and a pointer to right child \*/

struct tNode

{

   int data;

   struct tNode\* left;

   struct tNode\* right;

};

* /\* Structure of a stack node. Linked List implementation is used for

   stack. A stack node contains a pointer to tree node and a pointer to

   next stack node \*/

struct sNode

{

  struct tNode \*t;

  struct sNode \*next;

};

/\* UTILITY FUNCTIONS \*/

* /\* Function to push an item to sNode\*/

void push(struct sNode\*\* top\_ref, struct tNode \*t)

{

  /\* allocate tNode \*/

  struct sNode\* new\_tNode =

            (struct sNode\*) malloc(sizeof(struct sNode));

  if(new\_tNode == NULL)

  {

     printf("Stack Overflow \n");

     getchar();

     exit(0);

  }

  /\* put in the data  \*/

  new\_tNode->t  = t;

  /\* link the old list off the new tNode \*/

  new\_tNode->next = (\*top\_ref);

  /\* move the head to point to the new tNode \*/

  (\*top\_ref)    = new\_tNode;

}

* /\* The function returns true if stack is empty, otherwise false \*/

bool isEmpty(struct sNode \*top)

{

   return (top == NULL)? 1 : 0;

}

* + /\* Function to pop an item from stack\*/

struct tNode \*pop(struct sNode\*\* top\_ref)

{

  struct tNode \*res;

  struct sNode \*top;

  /\*If sNode is empty then error \*/

  if(isEmpty(\*top\_ref))

  {

     printf("Stack Underflow \n");

     getchar();

     exit(0);

  }

  else

  {

     top = \*top\_ref;

     res = top->t;

     \*top\_ref = top->next;

     free(top);

     return res;

  }

}

/\* Helper function that allocates a new tNode with the

   given data and NULL left and right pointers. \*/

struct tNode\* newtNode(int data)

{

  struct tNode\* tNode = (struct tNode\*)

                       malloc(sizeof(struct tNode));

  tNode->data = data;

  tNode->left = NULL;

  tNode->right = NULL;

  return(tNode);

}

## Queues using array

int queue\_arr[MAX];

int rear = - 1;

int front = - 1;

insert()

{

    int added\_item;

    if (rear == MAX - 1)

    printf("Queue Overflow\n");

    else

    {

        if (front == - 1)

        /\*If queue is initially empty \*/

        front = 0;

        printf("Input the element for adding in queue : ");

        scanf("%d", &added\_item);

        rear = rear + 1;

        queue\_arr[rear] = added\_item;

    }

} /\*End of insert()\*/

del()

{

    if (front == - 1 || front > rear)

    {

        printf("Queue Underflow\n");

        return ;

    }

    else

    {

        printf("Element deleted from queue is : %d\n", queue\_arr[front]);

        front = front + 1;

    }

} /\*End of del() \*/

## Queues using linked list

struct node

{

    int info;

    struct node \*next;

} \*front, \*rear;

rear = NULL;

front = NULL;

void enqueue(int elt)

{

    struct node \*p;

    p = (struct node\*)malloc(sizeof(struct node));

    p->info = elt;

    p->next = NULL;

    if (rear == NULL || front == NULL)

    front = p;

    else

    rear->next = p;

    rear = p;

}

int dequeue()

{

    struct node \*p;

    int elt;

    if (front == NULL || rear == NULL)

    {

        printf("\nUnder Flow");

        getch();

        exit(0);

    }

    else

    {

        p = front;

        elt = p->info;

        front = front->next;

        free(p);

    }

    return (elt);

}

## Stacks for arrays

int stack[MAX\_SIZE];

void push();

int pop();

void traverse();

int is\_empty();

int top\_element();

int top = -1;

void push(int value)

{

top++;

stack[top] = value;

}

int pop()

{

int element;

if ( top == -1 )

return top;

element = stack[top];

top--;

return element;

}

int is\_empty()

{

if ( top == - 1 )

return 1;

else

return 0;

}

## Iterating over a hash map

Map<Integer, Integer> map = **new** HashMap<Integer, Integer>();  
  
//iterating over keys only  
**for**(Integer key : map.keySet()) {  
    System.out.println("Key = " + key);  
}  
  
//iterating over values only  
**for** (Integer value : map.values()) {  
    System.out.println("Value = " + value);  
}

## Merge Sort Implementation

void merge(int \* array\_of\_integers, int p, int q, int r){

int n1 = q - p + 1;

int n2 = r - q;

int i, j, k;

int left\_array[n1 + 1];

int right\_array[n2 + 1];

for(i = 0; i < n1; i++)

left\_array[i] = array\_of\_integers[p + i];

for(j = 0; j < n2; j++)

right\_array[j] = array\_of\_integers[q + j];

i = 0;

j = 0;

for(k = p; k < r; k++){

if(left\_array[i] <= right\_array[j]){

array\_of\_integers[k] = left\_array[i];

i++;

} else {

array\_of\_integers[k] = right\_array[j];

j++;

}

}

}

void merge\_sort(int \* array\_of\_integers, int p, int r){

if(p < r){

int q = (p+r)/2;

merge\_sort(array\_of\_integers, p, q);

merge\_sort(array\_of\_integers, q + 1, r);

merge(array\_of\_integers, p, q, r);

}

}

## Quick Sort Implementation

void exch(int a[],int i,int j){

int s=a[i];

a[i]=a[j];

a[j]=s;

}

int partition(int a[],int l,int h);

void quick(int a[],int l,int h){

if (h<=l) return ;

int j=partition(a,l,h);

quick(a,l,j-1);

quick(a,j+1,h);

}

int partition(int a[],int l,int h){

int i=l-1;

int j=h;

int v=a[l];

while(true){

while( a[++i]<v);

while(a[--j]>v) if (j==i) break;

if (i>=j) break;

exch(a,i,j);

}

exch(a,i,h);

return i;

}

int main(){

int a[]={12,43,13,5,8,10,11,9,20,17};

int n=sizeof(a)/sizeof(int);

quick(a,0,n-1);

for (int i=0;i<n;i++){

cout<<a[i]<<" ";

}

return 0;

}

## Heap Sort Implementation

int hsort[25],n,i;

void adjust(int,int);

void heapify();

void main()

{

int temp;

clrscr();

printf("\n\t\t\t\tHEAP SORT");

printf("\n\t\t\t\t\*\*\*\* \*\*\*\*\n\n\n");

printf("\nenter no of elements:");

scanf("%d",&n);

printf("\nenter elements to be sorted\n\n");

for(i=1;i<=n;i++)

scanf("%d",&hsort[i]);

heapify();

for(i=n;i>=2;i--)

{

temp=hsort[1];

hsort[1]=hsort[i];

hsort[i]=temp;

adjust(1,i-1);

}

printf("\nSORTED ELEMENT\n\n");

for(i=1;i<=n;i++)

printf("%d\n",hsort[i]);

getch();

}

void heapify()

{

int i;

for(i=n/2;i>=1;i--)

adjust(i,n);

}

void adjust(int i,int n)

{

int j,element;

j=2\*i;

element=hsort[i];

while(j<=n)

{

if((j<n)&&(hsort[j]<hsort[j+1]))

j=j++;

if(element>=hsort[j])

break;

hsort[j/2]=hsort[j];

j=2\*j;

}

hsort[j/2]=element;

}