Name: Om Vivek Gharge

PRN: 2020BTEIT00041

Q1.

CODE:

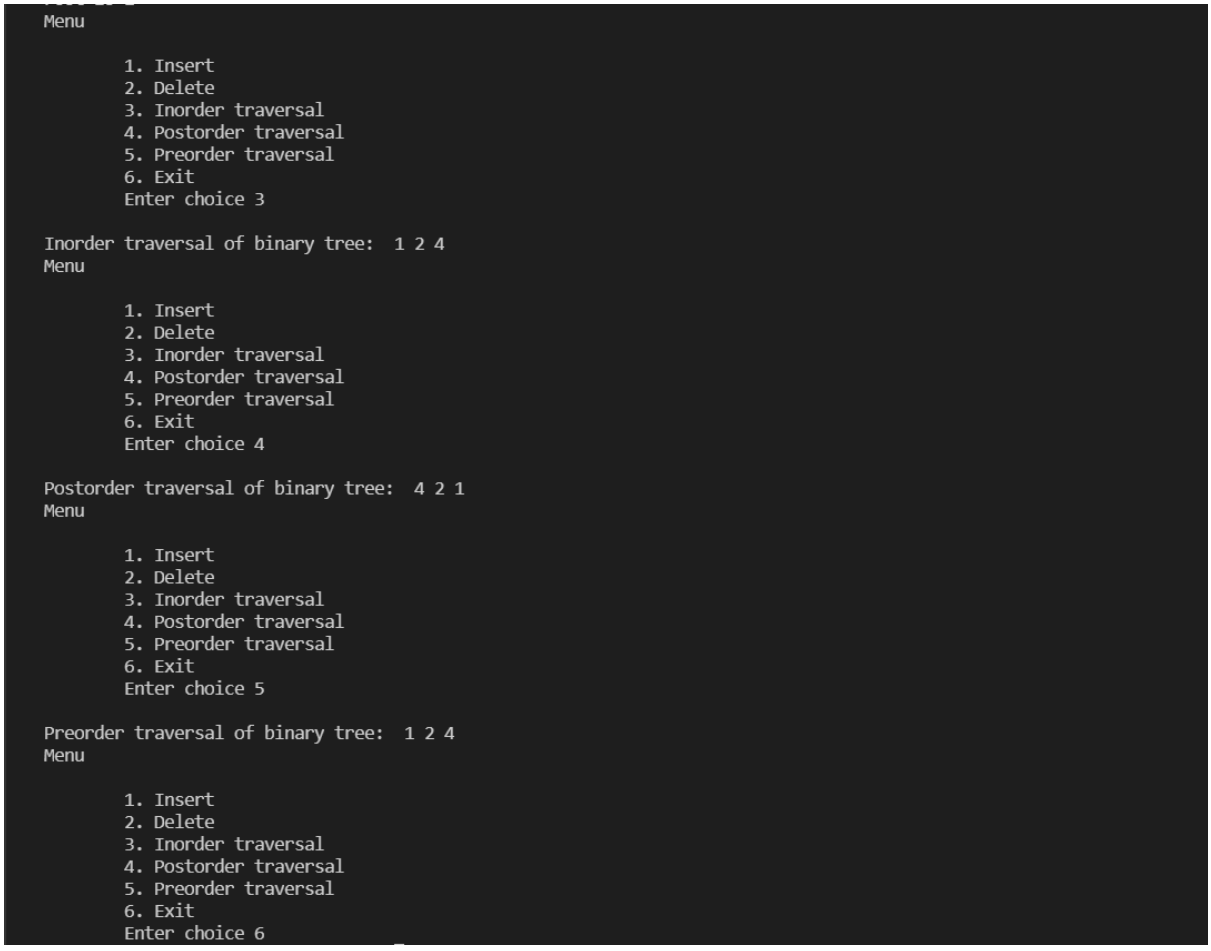
```c
/*
Name: Om Vivek Gharge
PRN: 2020BTEIT00041
Q1.
*/

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
struct tree
{
    int info;
    struct tree *left;
    struct tree *right;
};
struct tree *insert(struct tree *root, int x)
{
    if (!root)
    {
        root = (struct tree *)malloc(sizeof(struct tree));
        root->info = x;
        root->left = NULL;
        root->right = NULL;
        return (root);
    }
    if (root->info > x)
        root->left = insert(root->left, x);
    else
    {
        if (root->info < x)
            root->right = insert(root->right, x);
    }
    return (root);
}
void delete_tree(struct tree *ptr)
{
    if (ptr)
    {
        delete_tree(ptr->left);
```

```
40          delete_tree(ptr->right);
41          free(ptr);
42      }
43  }
44  void inorder(struct tree *root)
45  {
46      if (root != NULL)
47      {
48          inorder(root->left);
49          printf(" %d", root->info);
50          inorder(root->right);
51          return;
52      }
53      return;
54  }
55  void postorder(struct tree *root)
56  {
57      if (root != NULL)
58      {
59          postorder(root->left);
60          postorder(root->right);
61          printf(" %d", root->info);
62      }
63      return;
64  }
65  void preorder(struct tree *root)
66  {
67      if (root != NULL)
68      {
69          printf(" %d", root->info);
70          preorder(root->left);
71          preorder(root->right);
72      }
73      return;
74  }
75  int main()
76  {
77      struct tree *root;
78      int choice, data, item_no;
```

```c
    root = NULL;
    while (1)
    {
        printf("\n Menu\n");
        printf("\n \t 1. Insert");
        printf("\n\t 2. Delete");
        printf("\n\t 3. Inorder traversal");
        printf("\n\t 4. Postorder traversal");
        printf("\n\t 5. Preorder traversal");
        printf("\n\t 6. Exit ");
        printf("\n\t Enter choice ");
        scanf(" %d", &choice);
        if (choice < 1 || choice > 5)
            break;
        switch (choice)
        {
        case 1:
            printf("\n Enter data ");
            scanf("%d", &data);
            root = insert(root, data);
            printf("\n root is %d", root->info);
            break;
        case 2:
            delete_tree(root);
            break;
        case 3:
            printf("\n Inorder traversal of binary tree: ");
            inorder(root);
            break;
        case 4:
            printf("\n Postorder traversal of binary tree: ");
            postorder(root);
            break;
        case 5:
            printf("\n Preorder traversal of binary tree: ");
            preorder(root);
        }
    }
    return 0;
}
```

OUTPUT:


Menu
1. Insert
2. Delete
3. Inorder traversal
4. Postorder traversal
5. Preorder traversal
6. Exit
Enter choice 1
Enter data 1
root is 1
Menu
1. Insert
2. Delete
3. Inorder traversal
4. Postorder traversal
5. Preorder traversal
6. Exit
Enter choice 1
Enter data 2
root is 1
Menu
1. Insert
2. Delete
3. Inorder traversal
4. Postorder traversal
5. Preorder traversal
6. Exit
Enter choice 1
Enter data 4
root is 1
Menu
1. Insert
2. Delete
3. Inorder traversal
4. Postorder traversal
5. Preorder traversal
6. Exit


Menu
1. Insert
2. Delete
3. Inorder traversal
4. Postorder traversal
5. Preorder traversal
6. Exit
Enter choice 3
Inorder traversal of binary tree: 1 2 4
Menu
1. Insert
2. Delete
3. Inorder traversal
4. Postorder traversal
5. Preorder traversal
6. Exit
Enter choice 4
Postorder traversal of binary tree: 4 2 1
Menu
1. Insert
2. Delete
3. Inorder traversal
4. Postorder traversal
5. Preorder traversal
6. Exit
Enter choice 5
Preorder traversal of binary tree: 1 2 4
Menu
1. Insert
2. Delete
3. Inorder traversal
4. Postorder traversal
5. Preorder traversal
6. Exit
Enter choice 6

Q2.

CODE:

```c
#include <stdio.h>
#include <stdlib.h>
struct BST
{
    int data;
    struct BST *left;
    struct BST *right;
};
typedef struct BST NODE;
NODE *node;
NODE *createtree(NODE *node, int data)
{
    if (node == NULL)
    {
        NODE *temp;
        temp = (NODE *)malloc(sizeof(NODE));
        temp->data = data;
        temp->left = temp->right = NULL;
        return temp;
    }
    if (data < (node->data))
    {
        node->left = createtree(node->left, data);
    }
    else if (data > node->data)
    {
        node->right = createtree(node->right, data);
    }
    return node;
}
NODE *search(NODE *node, int data)
{
    if (node == NULL)
        printf("\nElement not found");
    else if (data < node->data)
    {
        node->left = search(node->left, data);
    }
    else if (data > node->data)
    {
```

```
41          node->right = search(node->right, data);
42      }
43      else
44          printf("\nElement found is: %d", node->data);
45      return node;
46  }
47  void inorder(NODE *node)
48  {
49      if (node != NULL)
50      {
51          inorder(node->left);
52          printf("%4d", node->data);
53          inorder(node->right);
54      }
55  }
56  void preorder(NODE *node)
57  {
58      if (node != NULL)
59      {
60          printf("%4d", node->data);
61          preorder(node->left);
62          preorder(node->right);
63      }
64  }
65  void postorder(NODE *node)
66  {
67      if (node != NULL)
68      {
69          postorder(node->left);
70          postorder(node->right);
71          printf("%4d", node->data);
72      }
73  }
74  NODE *findMin(NODE *node)
75  {
76      if (node == NULL)
77      {
78          return NULL;
79      }
```

```
    if (node->left)
        return findMin(node->left);
    else
        return node;
}
NODE *del(NODE *node, int data)
{
    NODE *temp;
    if (node == NULL)
    {
        printf("\nElement not found");
    }
    else if (data < node->data)
    {
        node->left = del(node->left, data);
    }
    else if (data > node->data)
    {
        node->right = del(node->right, data);
    }
    else
    {
        if (node->right && node->left)
        {
            temp = findMin(node->right);
            node->data = temp->data;
            node->right = del(node->right, temp->data);
        }
        else
        {
            temp = node;
            if (node->left == NULL)
                node = node->right;
            else if (node->right == NULL)
                node = node->left;
            free(temp);
        }
    }
    return node;
}
```

```c
void main()
{
    int data, ch, i;
    NODE *root = NULL;
    while (1)
    {
        printf("\n1.Insertion in Binary Search Tree");
        printf("\n2.Search Element in Binary Search Tree");
        printf("\n3.Delete Element in Binary Search Tree");
        printf("\n4.Inorder\n5.Preorder\n6.Postorder\n7.Exit");
        printf("\nEnter your choice: ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("\nEnter value ");
            scanf("%d", &data);
            root = createtree(root, data);
            break;
        case 2:
            printf("\nEnter the element to search: ");
            scanf("%d", &data);
            root = search(root, data);
            break;
        case 3:
            printf("\nEnter the element to delete: ");
            scanf("%d", &data);
            root = del(root, data);
            break;
        case 4:
            printf("\nInorder Traversal: \n");
            inorder(root);
            break;
        case 5:
            printf("\nPreorder Traversal: \n");
            preorder(root);
            break;
        case 6:
            printf("\nPostorder Traversal: \n");
            postorder(root);
            break;
        case 7:
            exit(0);
        default:
            printf("\nWrong option");
            break;
        }
    }
}
```

OUTPUT:

```
1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Delete Element in Binary Search Tree
4.Inorder
5.Preorder
6.Postorder
7.Exit
Enter your choice: 1

Enter value 1

1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Delete Element in Binary Search Tree
4.Inorder
5.Preorder
6.Postorder
7.Exit
Enter your choice: 1

Enter value 2

1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Delete Element in Binary Search Tree
4.Inorder
5.Preorder
6.Postorder
7.Exit
Enter your choice: 1

Enter value 3

1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Delete Element in Binary Search Tree
4.Inorder
5.Preorder
6.Postorder
7.Exit
Enter your choice: 2
```

```
Enter the element to search: 2

Element found is: 2
1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Delete Element in Binary Search Tree
4.Inorder
5.Preorder
6.Postorder
7.Exit
Enter your choice: 4

Inorder Traversal:
   1   2   3
1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Delete Element in Binary Search Tree
4.Inorder
5.Preorder
6.Postorder
7.Exit
Enter your choice: 5

Preorder Traversal:
   1   2   3
1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Delete Element in Binary Search Tree
4.Inorder
5.Preorder
6.Postorder
7.Exit
Enter your choice: 6

Postorder Traversal:
   3   2   1
1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Delete Element in Binary Search Tree
4.Inorder
5.Preorder
6.Postorder
7.Exit
Enter your choice: 7
```

Q3.

CODE:

```c
/*
    Name: Om Vivek Gharge
    PRN: 2020BTEIT00041
    Q3.
*/

#include <stdio.h>
#include <stdlib.h>
#define bool int
struct tNode
{
    int data;
    struct tNode *left;
    struct tNode *right;
};
struct sNode
{
    struct tNode *t;
    struct sNode *next;
};
void push(struct sNode **top_ref, struct tNode *t);
struct tNode *pop(struct sNode **top_ref);
bool isEmpty(struct sNode *top);
void inOrder(struct tNode *root)
{
    struct tNode *current = root;
    struct sNode *s = NULL;
    bool done = 0;
    while (!done)
    {
        if (current != NULL)
        {
            push(&s, current);
            current = current->left;
        }
        else
        {
            if (!isEmpty(s))
            {
```

```
            {
                current = pop(&s);
                printf("%d ", current->data);
                current = current->right;
            }
            else
                done = 1;
        }
    }
}
void push(struct sNode **top_ref, struct tNode *t)
{
    struct sNode *new_tNode =
        (struct sNode *)malloc(sizeof(struct sNode));
    if (new_tNode == NULL)
    {
        printf("Stack Overflow \n");
        getchar();
        exit(0);
    }
    new_tNode->t = t;
    new_tNode->next = (*top_ref);
    (*top_ref) = new_tNode;
}
bool isEmpty(struct sNode *top)
{
    return (top == NULL) ? 1 : 0;
}
struct tNode *pop(struct sNode **top_ref)
{
    struct tNode *res;
    struct sNode *top;
    if (isEmpty(*top_ref))
    {
        printf("Stack Underflow \n");
    }
    else
    {
```

```
        top = *top_ref;
        res = top->t;
        *top_ref = top->next;
        free(top);
        return res;
    }
}
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);
}
int main()
{
    struct tNode *root = newtNode(1);
    root->left = newtNode(2);
    root->right = newtNode(3);
    root->left->left = newtNode(4);
    root->left->right = newtNode(5);
    printf("\n Inorder traversal is:");
    inOrder(root);
    return 0;
}
```

OUTPUT:

Q4.

CODE:

```c
/*
    Name: Om Vivek Gharge
    PRN: 2020BTEIT00041
    Q4.
*/
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
struct Node
{
    int data;
    struct Node *left, *right;
};
struct Stack
{
    int size;
    int top;
    struct Node **array;
};
struct Node *newNode(int data)
{
    struct Node *node = (struct Node *)malloc(sizeof(struct Node));
    node->data = data;
    node->left = node->right = NULL;
    return node;
}
struct Stack *createStack(int size)
{
    struct Stack *stack = (struct Stack *)malloc(sizeof(struct Stack));
    stack->size = size;
    stack->top = -1;
    stack->array = (struct Node **)malloc(stack->size * sizeof(struct Node *));
    return stack;
}
int isFull(struct Stack *stack)
{
    return stack->top - 1 == stack->size;
}
int isEmpty(struct Stack *stack)
{
```

```
    return stack->top == -1;
}
void push(struct Stack *stack, struct Node *node)
{
    if (isFull(stack))
        return;
    stack->array[++stack->top] = node;
}
struct Node *pop(struct Stack *stack)
{
    if (isEmpty(stack))
        return NULL;
    return stack->array[stack->top--];
}
struct Node *peek(struct Stack *stack)
{
    if (isEmpty(stack))
        return NULL;
    return stack->array[stack->top];
}
void postOrderIterative(struct Node *root)
{
    if (root == NULL)
        return;
    struct Stack *stack = createStack(MAX_SIZE);
    do
    {
        while (root)
        {
            if (root->right)
                push(stack, root->right);
            push(stack, root);
            root = root->left;
        }
        root = pop(stack);
        if (root->right && peek(stack) == root->right)
        {
            pop(stack);
            push(stack, root);
            root = root->right;
        }
        else
        {
            printf("%d ", root->data);
            root = NULL;
        }
    } while (!isEmpty(stack));
}
int main()
{
    struct Node *root = NULL;
    root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);
    root->right->left = newNode(6);
    root->right->right = newNode(7);
    printf("Post order traversal of binary tree is :");
    postOrderIterative(root);
    return 0;
}
```

OUTPUT:

```
Post order traversal of binary tree is :4 5 2 6 7 3 1
```

Q5.

CODE:

```c
/*
    Name: Om Vivek Gharge
    PRN: 2020BTEIT00041
    Q5.
*/
#include <stdio.h>
#include <stdlib.h>
struct node
{
    struct node *left;
    struct node *right;
    int data;
};
struct stack
{
    struct node *data;
    struct stack *next;
};
void push(struct stack **top, struct node *n);
struct node *pop(struct stack **top);
int isEmpty(struct stack *top);
int tree_traversal(struct node *root)
{
    struct node *temp = root;
    struct stack *s_temp = NULL;
    int flag = 1;
    while (flag)
    {
        if (temp)
        {
            printf("%4d", temp->data);
            push(&s_temp, temp);
            temp = temp->left;
        }
        else
        {
            if (!isEmpty(s_temp))
            {
                temp = pop(&s_temp);
                temp = temp->right;
```

```
            }
            else
                flag = 0;
        }
    }
}
void push(struct stack **top, struct node *n)
{
    struct stack *new_n = (struct stack *)malloc(sizeof(struct stack));
    new_n->data = n;
    new_n->next = (*top);
    (*top) = new_n;
}
int isEmpty(struct stack *top)
{
    if (top == NULL)
        return 1;
    else
        return 0;
}
struct node *pop(struct stack **top_n)
{
    struct node *item;
    struct stack *top;
    top = *top_n;
    item = top->data;
    *top_n = top->next;
    free(top);
    return item;
}
struct node *create_node(int data)
{
    struct node *new_n = (struct node *)malloc(sizeof(struct node));
    new_n->data = data;
    new_n->left = NULL;
    new_n->right = NULL;
    return (new_n);
}
int main()
{
    struct node *root;
    root = create_node(8);
    root->left = create_node(5);
    root->right = create_node(4);
    root->left->left = create_node(7);
    root->left->right = create_node(6);
    printf("\n Preorder traversal is:");
    tree_traversal(root);
    return 0;
}
```

OUTPUT:

```
Preorder traversal is:   8   5   7   6   4
```

Q6.

CODE: A)

```c
/*
    Name: Om Vivek Gharge
    PRN: 2020BTEIT00041
*/

#include <stdio.h>

void insert(int A[], int n)
{
    int temp, i;
    i = n;
    temp = A[i];

    while (temp < A[i / 2] && i > 1)
    {
        A[i] = A[i / 2];
        i = i / 2;
    }
    A[i] = temp;
}

void Delete(int A[], int n)
{
    int val, i, temp, j, x;

    val = A[1];
    // x = A[1];
    A[1] = A[n];
    A[n] = val;
    i = 1;
    j = 2 * i;

    while (j <= n - 1)
    {
        if (j < n - 1 && (A[j + 1] < A[j]))
```

```
        {
            j = j + 1;
        }

        if (A[i] > A[j])
        {
            temp = A[i];
            A[i] = A[j];
            A[j] = temp;
            i = j;
            j = 2 * i;
        }
        else
        {
            break;
        }
    }

    printf("\n The deleted element is: %d ", val);
}

int main()
{

    int size, choice;
    printf("Enter the size of an array: ");
    scanf("%d", &size);
    int B[size];
    printf("\n\nNote: Enter the the first element of array as '0' because the heap DS will start from 1st index\n");
    printf("\nInsert the elements into the array: \n");
    for (int i = 0; i < size; i++)
    {
        scanf("%d", &B[i]);
    }

    while (choice != 0)
    {
        printf("\nEnter your choice: \n");
        printf("1. Creation of a heap\n2. Deleting a element from a heap\n3. Displaying a heap\n");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("\nCreating a heap...\n");
            for (int i = 2; i <= sizeof(B) / sizeof(B[0]) - 1; i++)
            {
                insert(B, i);
            }

            break;

        case 2:

            printf("Deleting...\n");
            Delete(B, sizeof(B) / sizeof(B[0]) - 1);
            printf("The deleted element will be stored at the free space of given array.");
            break;

        case 3:

            printf("Heap: [ ");
            for (int i = 1; i <= sizeof(B) / sizeof(B[0]) - 1; i++)
            {
                printf("%d ", B[i]);
            }
            printf("]\n");

            break;
        case 0:
```

```
104             case 0:
105
106                 break;
107             }
108         }
109
110         return 0;
111     }
```

OUTPUT:

```
Enter the size of an array: 5


Note: Enter the the first element of array as '0' because the heap DS will start from 1st index

Insert the elements into the array:
0 1 2 3 4

Enter your choice:
1. Creation of a heap
2. Deleting a element from a heap
3. Displaying a heap
3
Heap: [ 1 2 3 4 ]

Enter your choice:
1. Creation of a heap
2. Deleting a element from a heap
3. Displaying a heap
2
Deleting...

 The deleted element is: 1 The deleted element will be stored at the free space of given array.
Enter your choice:
1. Creation of a heap
2. Deleting a element from a heap
3. Displaying a heap
3
Heap: [ 2 4 3 1 ]
```

B)

```c
/*
    Name: Om Vivek Gharge
    PRN: 2020BTEIT00041
*/

#include <stdio.h>

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

void insert(int A[], int n)
{
    int temp, i;
    i = n;
    temp = A[i];

    while (temp > A[i / 2] && i > 1)
    {
        A[i] = A[i / 2];
        i = i / 2;
    }
    A[i] = temp;
}

void Delete(int A[], int n)
{
    int val, i, temp, j, x;

    val = A[1];
    // x = A[1];
    A[1] = A[n];
    A[n] = val;
    i = 1;
    j = 2 * i;

```

```c

    while (j <= n - 1)
    {
        if (j < n - 1 && (A[j + 1] > A[j]))
        {
            j = j + 1;
        }

        if (A[i] < A[j])
        {
            temp = A[i];
            A[i] = A[j];
            A[j] = temp;
            i = j;
            j = 2 * i;
        }
        else
        {
            break;
        }
    }

    printf("\n The deleted element is: %d ", val);
}

int main()
{

    int size, choice;
    printf("Enter the size of an array: ");
    scanf("%d", &size);
    int B[size];
    printf("\n\nNote: Enter the the first element of array as '0' because the heap DS will start from 1st index\n");
    printf("\nInsert the elements into the array: \n");
    for (int i = 0; i < size; i++)
    {
        scanf("%d", &B[i]);
    }
```

```c
    while (choice != 0)
    {
        printf("\nEnter your choice: \n");
        printf("1. Creation of a heap\n2. Deleting a element from a heap\n3. Displaying a heap\n");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("\nCreating a heap...\n");
            for (int i = 2; i <= sizeof(B) / sizeof(B[0]) - 1; i++)
            {
                insert(B, i);
            }

            break;

        case 2:

            printf("Deleting...\n");
            Delete(B, sizeof(B) / sizeof(B[0]) - 1);
            printf("The deleted element will be stored at the free space of given array.");
            break;

        case 3:

            printf("Heap: [ ");
            for (int i = 1; i <= sizeof(B) / sizeof(B[0]) - 1; i++)
            {
                printf("%d ", B[i]);
            }
            printf("]\n");

            break;
        case 0:

            break;
        }
    }

    return 0;
}
```

OUTPUT:

```
Enter the size of an array: 5


Note: Enter the the first element of array as '0' because the heap DS will start from 1st index

Insert the elements into the array:
0
1 2 3 4

Enter your choice:
1. Creation of a heap
2. Deleting a element from a heap
3. Displaying a heap
2
Deleting...

 The deleted element is: 1 The deleted element will be stored at the free space of given array.
Enter your choice:
1. Creation of a heap
2. Deleting a element from a heap
3. Displaying a heap
3
Heap: [ 4 2 3 1 ]
```