2.Construct B-Tree an order of 5 with a set of 100 random elements stored in array. Implement searching, insertion and deletion operations

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 4
#define MIN 2
struct BTreeNode {
   int keys[MAX + 1], count;
   struct BTreeNode *children[MAX + 1];
};

struct BTreeNode *root;
// Function prototypes
struct BTreeNode *createNode(int key, struct BTreeNode *child);
void insert(int key);
void insertNode(int key, int pos, struct BTreeNode *node, struct BTreeNode *child);
void splitNode(int key, int *pval, int pos, struct BTreeNode *node, struct BTreeNode *child,
struct BTreeNode **newNode);
int setValue(int key, int *pval, struct BTreeNode *node, struct BTreeNode **child);
void search(int key);
void deletion(int key);
void removeVal(struct BTreeNode *node, int pos);
void doRightShift(struct BTreeNode *node, int pos);
void doLeftShift(struct BTreeNode *node, int pos);
void mergeNodes(struct BTreeNode *node, int pos);
void adjustNode(struct BTreeNode *node, int pos);
int predecessor(struct BTreeNode *node);
int successor(struct BTreeNode *node);
void traversal(struct BTreeNode *node);

int main() {
   int i, r, num;
   int array[100];

   // Generating 100 random elements
   srand(time(NULL));
   for (i = 0; i < 100; i++) {
      array[i] = rand() % 1000;
      insert(array[i]);
   }
   printf("B-Tree of order 5 constructed with 100 random elements:\n");
   traversal(root);
   printf("\n");
   // Searching
   printf("Enter element to search: ");
```

```
    scanf("%d", &num);
    search(num);
    // Deletion
    printf("Enter element to delete: ");
    scanf("%d", &num);
    deletion(num);
    printf("B-Tree after deletion:\n");
    traversal(root);
    printf("\n");
    return 0;
}
struct BTreeNode *createNode(int key, struct BTreeNode *child) {
    struct BTreeNode *newNode;
    newNode = (struct BTreeNode *)malloc(sizeof(struct BTreeNode));
    newNode->keys[1] = key;
    newNode->count = 1;
    newNode->children[0] = root;
    newNode->children[1] = child;
    return newNode;
}
void insert(int key) {
    int flag, i;
    struct BTreeNode *child;

    flag = setValue(key, &i, root, &child);
    if (flag)
        root = createNode(i, child);
}
void insertNode(int key, int pos, struct BTreeNode *node, struct BTreeNode *child) {
    int j = node->count;
    while (j > pos) {
        node->keys[j + 1] = node->keys[j];
        node->children[j + 1] = node->children[j];
        j--;
    }
    node->keys[j + 1] = key;
    node->children[j + 1] = child;
    node->count++;
}

void splitNode(int key, int *pval, int pos, struct BTreeNode *node, struct BTreeNode *child,
struct BTreeNode **newNode) {
    int median, j;
    if (pos > MIN)
        median = MIN + 1;
```

```
    else
      median = MIN;
    *newNode = (struct BTreeNode *)malloc(sizeof(struct BTreeNode));
    j = median + 1;
    while (j <= MAX) {
      (*newNode)->keys[j - median] = node->keys[j];
      (*newNode)->children[j - median] = node->children[j];
      j++;
    }
    node->count = median;
    (*newNode)->count = MAX - median;

    if (pos <= MIN)
      insertNode(key, pos, node, child);
    else
      insertNode(key, pos - median, *newNode, child);

    *pval = node->keys[node->count];
    (*newNode)->children[0] = node->children[node->count];
    node->count--;
}
int setValue(int key, int *pval, struct BTreeNode *node, struct BTreeNode **child) {
    int pos;
    if (!node) {
      *pval = key;
      *child = NULL;
      return 1;
    }

    if (key < node->keys[1]) {
      pos = 0;
    } else {
      for (pos = node->count; (key < node->keys[pos] && pos > 1); pos--);
      if (key == node->keys[pos]) {
        printf("Duplicates not allowed\n");
        return 0;
      }
    }
    if (setValue(key, pval, node->children[pos], child)) {
      if (node->count < MAX) {
        insertNode(*pval, pos, node, *child);
        return 0;
      } else {
        splitNode(*pval, pval, pos, node, *child, child);
        return 1;
```

```
    }
  }
  return 0;
}
void search(int key) {
  int pos, flag = 0;
  struct BTreeNode *node = root;

  while (node) {
    for (pos = 0; pos < node->count && key > node->keys[pos + 1]; pos++);
    if (pos <= node->count && key == node->keys[pos + 1]) {
      printf("Element %d found in the B-Tree\n", key);
      flag = 1;
      break;
    }
    node = node->children[pos];
  }
  if (!flag)
    printf("Element %d not found in the B-Tree\n", key);
}
void deletion(int key) {
  struct BTreeNode *node = root;
  struct BTreeNode *temp;
  int pos, flag = 0;

  while (node) {
    for (pos = 0; pos < node->count && key > node->keys[pos + 1]; pos++);
    if (pos <= node->count && key == node->keys[pos + 1]) {
      flag = 1;
      break;
    }
    node = node->children[pos];
  }

  if (flag) {
    printf("Element %d deleted from the B-Tree\n", key);
    removeVal(root, pos);
  } else {
    printf("Element %d not found in the B-Tree\n", key);
  }

  if (root->count == 0) {
    temp = root;
    root = root->children[0];
    free(temp);
```

```
    }
}

void removeVal(struct BTreeNode *node, int pos) {
    int i;
    struct BTreeNode *temp;

    if (node->children[pos - 1]) {
        temp = node->children[pos];
        node->keys[pos] = predecessor(node);
        removeVal(node->children[pos], node->count);
    } else {
        for (i = pos + 1; i <= node->count; i++) {
            node->keys[i - 1] = node->keys[i];
            node->children[i - 1] = node->children[i];
        }
        node->count--;
    }
}

int predecessor(struct BTreeNode *node) {
    while (node->children[node->count])
        node = node->children[node->count];
    return node->keys[node->count];
}

int successor(struct BTreeNode *node) {
    while (node->children[0])
        node = node->children[0];
    return node->keys[1];
}
void traversal(struct BTreeNode *node) {
    int i;
    if (node) {
        for (i = 0; i < node->count; i++) {
            traversal(node->children[i]);
            printf("%d ", node->keys[i + 1]);
        }
        traversal(node->children[i]);
    }
}
```

3. Construct Min and Max Heap using arrays, delete any element and display the content of the Heap

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

#define MAX_SIZE 100

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

// Min Heap functions
void minHeapify(int arr[], int n, int i) {
   int smallest = i;
   int left = 2 * i + 1;
   int right = 2 * i + 2;

   if (left < n && arr[left] < arr[smallest])
      smallest = left;

   if (right < n && arr[right] < arr[smallest])
      smallest = right;

   if (smallest != i) {
      swap(&arr[i], &arr[smallest]);
      minHeapify(arr, n, smallest);
   }
}

void buildMinHeap(int arr[], int n)
 {
   int i;
   for (i= n / 2 - 1;i>=0;i--)
      minHeapify(arr, n, i);
}

void deleteMinHeapElement(int arr[], int *n, int value) {
   int i;
   for (i = 0; i < *n; i++) {
      if (arr[i] == value)
         break;
   }
```

```
    if (i == *n) {
        printf("Element not found in Min Heap\n");
        return;
    }

    arr[i] = arr[*n - 1];
    (*n)--;
    buildMinHeap(arr, *n);
}

// Max Heap functions
void maxHeapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest])
        largest = left;

    if (right < n && arr[right] > arr[largest])
        largest = right;

    if (largest != i) {
        swap(&arr[i], &arr[largest]);
        maxHeapify(arr, n, largest);
    }
}

void buildMaxHeap(int arr[], int n)
 { int i;
    for (i = n / 2 - 1; i >= 0; i--)
        maxHeapify(arr, n, i);
}

void deleteMaxHeapElement(int arr[], int *n, int value) {
    int i;
    for (i = 0; i < *n; i++) {
        if (arr[i] == value)
            break;
    }

    if (i == *n) {
        printf("Element not found in Max Heap\n");
        return;
```

```
    }

    arr[i] = arr[*n - 1];
    (*n)--;
    buildMaxHeap(arr, *n);
}

void displayHeap(int arr[], int n)
 { int i;
    for (i= 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main() {
    int minHeap[MAX_SIZE], maxHeap[MAX_SIZE];
    int n, i, value;

    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter elements: ");
    for (i = 0; i < n; i++) {
        scanf("%d", &minHeap[i]);
        maxHeap[i] = minHeap[i];
    }

    buildMinHeap(minHeap, n);
    buildMaxHeap(maxHeap, n);

    printf("Min Heap: ");
    displayHeap(minHeap, n);

    printf("Max Heap: ");
    displayHeap(maxHeap, n);

    printf("Enter element to delete: ");
    scanf("%d", &value);

    deleteMinHeapElement(minHeap, &n, value);
    deleteMaxHeapElement(maxHeap, &n, value);

    printf("Min Heap after deletion: ");
    displayHeap(minHeap, n);
```

```
    printf("Max Heap after deletion: ");
    displayHeap(maxHeap, n);

    return 0;
}
```