```
[] G & Share
main.c
                                                                              Run
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <limits.h>
 4 #define INITIAL CAPACITY 16
 5 - typedef struct {
        int *arr;
 6
 7
        int size;
 8
        int capacity;
 9
   } MinHeap;
   MinHeap* createHeap(int capacity);
10
    void insert(MinHeap *heap, int value);
11
   int extractMin(MinHeap *heap);
12
   void heapifyUp(MinHeap *heap, int index);
13
14
   void heapifyDown(MinHeap *heap, int index);
    void printHeap(MinHeap *heap);
15
   void freeHeap(MinHeap *heap);
16
17 - int main() {
18
        MinHeap *heap = createHeap(INITIAL CAPACITY);
19
        insert(heap, 3);
        insert(heap, 2);
20
21
        insert(heap, 15);
22
        insert(heap, 5);
23
        insert(heap, 4);
24
        insert(heap, 45);
25
        printf("Min-Heap: ");
26
        printHeap(heap);
```

```
[] & & Share
main.c
                                                                               Run
                                                                                         Output
26
        printHeap(heap);
27
        printf("Extracted min: %d\n", extractMin(heap));
28
        printf("Min-Heap after extraction: ");
29
        printHeap(heap)
30
        freeHeap(heap);
31
        return 0;
32
33 - MinHeap* createHeap(int capacity) {
        MinHeap *heap = (MinHeap *)malloc(sizeof(MinHeap));
34
35
        heap->capacity = capacity;
36
        heap->size = 0;
37
        heap->arr = (int *)malloc(capacity * sizeof(int));
38
        return heap;
39 }
40 - void insert(MinHeap *heap, int value) {
        if (heap->size == heap->capacity) {
41 -
42
            heap->capacity *= 2;
            heap->arr = (int *)realloc(heap->arr, heap->capacity * sizeof(int));
43
44
        }
45
        heap->arr[heap->size] = value;
46
        heapifyUp(heap, heap->size);
47
        heap->size++;
48
49 - void heapifyUp(MinHeap *heap, int index) {
        while (index > 0) {
50 *
            int parentIndex = (index - 1) / 2;
51
```

```
[] G & Share
                                                                               Run
                                                                                          Output
main.c
51
            int parentIndex = (index - 1) / 2;
52 *
            if (heap->arr[index] >= heap->arr[parentIndex]) {
53
                break;}
54
            int temp = heap->arr[index];
55
            heap->arr[index] = heap->arr[parentIndex];
56
            heap->arr[parentIndex] = temp;
            index = parentIndex;
57
58
        }
59
    }
60 - int extractMin(MinHeap *heap) {
        if (heap->size <= 0) {
61 *
62
            return INT_MAX;
63
        if (heap->size == 1) {
64 *
65
             return heap->arr[--heap->size];
66
        }
        int root = heap->arr[0];
67
68
        heap->arr[0] = heap->arr[--heap->size];
        heapifyDown(heap, 0);
69
70
        return root;
71
   }
72 * void heapifyDown(MinHeap *heap, int index) {
73
        int smallest = index;
        int left = 2 * index + 1;
74
75
        int right = 2 * index + 2;
76
         if (loft < hoop scize 00 hoop sarr[loft] < hoop sarr[cmalloct])
```

```
[] G & Share
main.c
                                                                               Run
                                                                                         Output
75
         int right = 2 * index + 2;
76
77 -
         if (left < heap->size && heap->arr[left] < heap->arr[smallest]) {
78
             smallest = left;
79
         }
         if (right < heap->size && heap->arr[right] < heap->arr[smallest]) {
80 -
 81
             smallest = right;
 82
         }
83
         if (smallest != index)
             int temp = heap->arr[index];
 84
85
             heap->arr[index] = heap->arr[smallest];
             heap->arr[smallest] = temp;
86
87
             heapifyDown(heap, smallest);
88
         }
89 }
90 - void printHeap(MinHeap *heap) {
         for (int i = 0; i < heap->size; i++) {
91 -
 92
             printf("%d ", heap->arr[i]);
 93
         printf("\n");
94
95 }
96 - void freeHeap(MinHeap *heap) {
         free(heap->arr);
97
98
         free(heap);
99 }
100
```

```
[] Share
                                                                            Run
main.c
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 int compare(const void *a, const void *b) {
       return (*(int*)a - *(int*)b);
 5 }
6 void printArray(int arr[], int size) {
7
       int i;
       for (i = 0; i < size; i++)
 8
 9
          printf("%d ", arr[i]);
       printf("\n");
10
   }
11
12 - int main() {
       int arr[] = \{64, 25, 12, 22, 11\};
13
       int n = sizeof(arr)/sizeof(arr[0]);
14
15
       qsort(arr, n, sizeof(int), compare);
       printf("Sorted array: \n");
16
17
       printArray(arr, n);
18
       return 0;
19
   }
20
```

```
[] & & Share
                                                                               Run
main.c
   #include <stdio.h>
2 #include <stdlib.h>
3 * int compareDescending(const void *a, const void *b) {
        return (*(int*)b - *(int*)a);
4
5
    }
6- void printArray(int arr[], int size) {
7
        int i;
        for (i = 0; i < size; i++)
9
            printf("%d ", arr[i]);
10
        printf("\n");
11
   }
12 * int main() {
13
        int arr[] = \{64, 25, 12, 22, 11\};
14
        int n = sizeof(arr)/sizeof(arr[0]);
15
        qsort(arr, n, sizeof(int), compareDescending);
16
        printf("Sorted array in descending order: \n");
17
        printArray(arr, n);
18
       return 0;
19
   }
20
```

```
[] G & Share
main.c
  #include<stdlib.h>
3 - struct Node {
       int data:
        struct Node *left;
        struct Node *right;
6
7 } Node;
8 Node* createNode(int data);
9 Node* insert(Node* root, int data);
10 Node* search(Node* root, int data);
11 void inorderTraversal(Node* root);
12 void preorderTraversal(Node* root);
13 void postorderTraversal(Node* root);
14 void freeTree(Node* root)
15 * Node* createNode(int data) {
       Node* newNode = (Node*)malloc(sizeof(Node));
16
17
        newNode->data = data;
18
        newNode->left = NULL;
        newNode->right = NULL;
19
20
        return newNode;
21 }
22 * Node* insert(Node* root, int data) {
23 -
        if (root == NULL) {
24
            return createNode(data);
25
26 -
        if (data < root->data) {
27
            root->left = insert(root->left, data);
```

```
[] G & Share
                                                                               Run
                                                                                         Output
main.c
28 -
        } else {
            root->right = insert(root->right, data);
29
30
        }
31
        return root;
32 }
33 * Node* search(Node* root, int data) {
        if (root == NULL || root->data == data) {
34 *
35
            return root;
36
        }
37 -
        if (data < root->data) {
38
            return search(root->left, data);
39 -
        } else {
40
            return search(root->right, data);
41
        }
42 }
43 * void inorderTraversal(Node* root) {
44 -
        if (root != NULL) {
45
            inorderTraversal(root->left);
            printf("%d ", root->data);
46
            inorderTraversal(root->right);
47
48
        }
49 }
50 * void preorderTraversal(Node* root) {
51 -
        if (root != NULL) {
            printf("%d ", root->data);
52
         nroardorTravarcal/roat laft).
```

```
[] ७ 

Share
main.c
                                                                                Run
54
            preorderTraversal(root->right);
55
        }
56 }
57 * void postorderTraversal(Node* root) {
58 -
        if (root != NULL) {
59
            postorderTraversal(root->left);
60
            postorderTraversal(root->right);
            printf("%d ", root->data);
61
62
        }
63 }
64 * void freeTree(Node* root) {
65 -
        if (root != NULL) {
            freeTree(root->left);
66
67
            freeTree(root->right);
            free(root);
68
69
        }
70 }
71 - int main() {
        Node* root = NULL;
72
73
        root = insert(root, 50);
74
        insert(root, 30);
75
        insert(root, 20);
        insert(root, 40);
76
77
        insert(root, 70);
78
        insert(root, 60);
79
        insert(root, 80);
```

```
[] G & Share
main.c
                                                                                Run
73
        root = insert(root, 50);
74
        insert(root, 30);
75
        insert(root, 20);
76
        insert(root, 40);
        insert(root, 70);
77
78
        insert(root, 60);
79
        insert(root, 80);
80
        Node* result = search(root, 40);
81 -
        if (result != NULL) {
            printf("Node with value 40 found.\n");
82
83 +
        } else {
84
            printf("Node with value 40 not found.\n");
85
        }
        printf("Inorder Traversal: ");
86
        inorderTraversal(root);
87
88
        printf("\n");
        printf("Preorder Traversal: ");
89
        preorderTraversal(root);
90
91
        printf("\n");
        printf("Postorder Traversal: ");
92
93
        postorderTraversal(root);
        printf("\n");
94
95
        freeTree(root);
96
        return 0;
```

97