

SEM-2 > DSA-ASS-12 > C 12\_1\_bucketsort\_intc > ...

```
1  /* Write a C or C++ program to sort the input array [12, 45, 33, 87, 56, 9, 11, 7, 67] using the Bucket Sort algorithm with 7 buckets. */
2
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  #define BUCK_C 7
7  #define ARR_SIZE 9
8
9  struct node
10 {
11     int data;
12     struct node *next;
13 };
14
15 void insertsorted(struct node **bucket, int value)
16 {
17     struct node *newnode = (struct node *)malloc(sizeof(struct node));
18     newnode->data = value;
19     newnode->next = NULL;
20     if (*bucket == NULL || value < (*bucket)->data)
21     {
22         newnode->next = *bucket;
23         *bucket = newnode;
24         return;
25     }
26     struct node *current = *bucket;
27     while (current->next != NULL && current->next->data < value)
28     {
29         current = current->next;
30     }
31     newnode->next = current->next;
32     current->next = newnode;
33 }
34
```

```
35 void bucketsort(int *arr, int n)
36 {
37     struct node *buckets[BUCK_C] = {NULL};
38     int i;
39     int max = arr[0];
40     for (i = 1; i < n; i++)
41     {
42         if (arr[i] > max)
43         {
44             max = arr[i];
45         }
46     }
47     for (i = 0; i < n; i++)
48     {
49         int index = (arr[i] * BUCK_C) / (max + 1);
50         insertsorted(&buckets[index], arr[i]);
51     }
52     int idx = 0;
53     for (i = 0; i < BUCK_C; i++)
54     {
55         struct node *node = buckets[i];
56         while (node != NULL)
57         {
58             arr[idx++] = node->data;
59             struct node *temp = node;
60             node = node->next;
61             free(temp);
62         }
63     }
64 }
65
66 int main()
67 {
68     int arr[ARR_SIZE] = {12, 45, 33, 87, 56, 9, 11, 7, 67};
69     printf("Original array : ");
70     for (int i = 0; i < ARR_SIZE; i++)
71     {
72         printf("%d ", arr[i]);
73     }
74     bucketsort(arr, ARR_SIZE);
75     printf("\nSorted array : ");
76     for (int i = 0; i < ARR_SIZE; i++)
77     {
78         printf("%d ", arr[i]);
79     }
80     printf("\n");
81     return 0;
82 }
83
```

```
Original array : 12 45 33 87 56 9 11 7 67  
Sorted array  : 7 9 11 12 33 45 56 67 87
```

SEM-2 > DSA-ASS-12 > 12\_2\_bt\_inprepost.c > ...

```
1  /* Write a C/C++ program that ask the user to enter 10 integer arr.
2  Use these arr to construct a binary tree with 10 nodes.
3  After constructing the tree, perform and display the inorder, preorder, and postorder traversals. */
4
5  #include <stdio.h>
6  #include <stdlib.h>
7
8  struct node
9  {
10     int data;
11     struct node *left;
12     struct node *right;
13 };
14
15 struct queue
16 {
17     struct node *treenode;
18     struct queue *next;
19 };
20
21 struct queue
22 {
23     struct queue *front;
24     struct queue *rear;
25 };
26
27 struct queue *createqueue()
28 {
29     struct queue *q = (struct queue *)malloc(sizeof(struct queue));
30     q->front = q->rear = NULL;
31     return q;
32 }
33
```

```

34 void enqueue(struct queue *q, struct node *node)
35 {
36     struct queuenode *temp = (struct queuenode *)malloc(sizeof(struct queuenode));
37     temp->treenode = node;
38     temp->next = NULL;
39     if (q->rear == NULL)
40     {
41         q->front = q->rear = temp;
42     }
43     else
44     {
45         q->rear->next = temp;
46         q->rear = temp;
47     }
48 }
49
50 struct node *dequeue(struct queue *q)
51 {
52     if (q->front == NULL)
53     {
54         return NULL;
55     }
56     struct queuenode *temp = q->front;
57     struct node *node = temp->treenode;
58     q->front = q->front->next;
59     if (q->front == NULL)
60     {
61         q->rear = NULL;
62     }
63     free(temp);
64     return node;
65 }
66
67 int isempty(struct queue *q)
68 {
69     return (q->front == NULL);
70 }
71

```

```

72 struct node *createnode(int value)
73 {
74     struct node *new_node = (struct node *)malloc(sizeof(struct node));
75     new_node->data = value;
76     new_node->left = new_node->right = NULL;
77     return new_node;
78 }
79
80 void insert(struct node **root, int value)
81 {
82     struct node *new_node = createnode(value);
83     if (*root == NULL)
84     {
85         *root = new_node;
86         return;
87     }
88     struct queue *q = createqueue();
89     enqueue(q, *root);
90     while (!isempty(q))
91     {
92         struct node *temp = dequeue(q);
93
94         if (temp->left == NULL)
95         {
96             temp->left = new_node;
97             break;
98         }
99         else
100         {
101             enqueue(q, temp->left);
102         }
103
104         if (temp->right == NULL)
105         {
106             temp->right = new_node;
107             break;
108         }
109         else
110         {
111             enqueue(q, temp->right);
112         }
113     }
114     while (!isempty(q))
115     {
116         dequeue(q);
117     }
118     free(q);
119 }

```

```
121 void inorder(struct node *root)
122 {
123     if (root == NULL)
124     {
125         return;
126     }
127     inorder(root->left);
128     printf("%d ", root->data);
129     inorder(root->right);
130 }
131
132 void preorder(struct node *root)
133 {
134     if (root == NULL)
135     {
136         return;
137     }
138     printf("%d ", root->data);
139     preorder(root->left);
140     preorder(root->right);
141 }
142
143 void postorder(struct node *root)
144 {
145     if (root == NULL)
146     {
147         return;
148     }
149     postorder(root->left);
150     postorder(root->right);
151     printf("%d ", root->data);
152 }
153
```

```

153
154 int main()
155 {
156     struct node *root = NULL;
157     int arr[10];
158     printf("Enter 10 integers : ");
159     for (int i = 0; i < 10; i++)
160     {
161         scanf("%d", &arr[i]);
162     }
163     for (int i = 0; i < 10; i++)
164     {
165         insert(&root, arr[i]);
166     }
167     printf("\nInorder traversal : ");
168     inorder(root);
169     printf("\nPreorder traversal : ");
170     preorder(root);
171     printf("\nPostorder traversal : ");
172     postorder(root);
173     printf("\n");
174     return 0;
175 }
176

```

#### TERMINAL

Enter 10 integers : 16 13 1 23 6 2 5 10 11 12

Inorder traversal : 10 23 11 13 12 6 16 2 1 5

Preorder traversal : 16 13 23 10 11 6 12 1 2 5

Postorder traversal : 10 11 23 12 6 13 2 5 1 16

PS C:\Users\shuvr\OneDrive\Documents\CODING\College C codes>



SEM-2 > DSA-ASS-12 > 12\_3\_inpre\_to\_btc > createnode(int)

```
1  /* Given the inorder and preorder traversals of a binary tree, write a C/C++ program to construct the binary tree.
2  Inorder: 1, 8, 19, 13, 25, 9, 5, 10, 4, 3
3  Preorder: 25, 8, 1, 13, 19, 5, 9, 4, 10, 3 */
4
5  #include <stdio.h>
6  #include <stdlib.h>
7
8  struct node
9  {
10     int data;
11     struct node *left;
12     struct node *right;
13 };
14
15 int find(int *arr, int start, int end, int value)
16 {
17     for (int i = start; i <= end; i++)
18     {
19         if (arr[i] == value)
20         {
21             return i;
22         }
23     }
24     return -1;
25 }
26
27 struct node *createnode(int value)
28 {
29     struct node *newnode = (struct node *)malloc(sizeof(struct node));
30     newnode->data = value;
31     newnode->left = NULL;
32     newnode->right = NULL;
33     return newnode;
34 }
```

```

36 struct node *buildtree(int *preorder, int *inorder, int start, int end, int *index)
37 {
38     if (start > end)
39     {
40         return NULL;
41     }
42     int current = preorder[*index];
43     (*index)++;
44     struct node *newnode = createnode(current);
45     if (start == end)
46     {
47         return newnode;
48     }
49     int pos = find(inorder, start, end, current);
50     newnode->left = buildtree(preorder, inorder, start, pos - 1, index);
51     newnode->right = buildtree(preorder, inorder, pos + 1, end, index);
52     return newnode;
53 }
54
55 void print_inorder(struct node *root)
56 {
57     if (root == NULL)
58     {
59         return;
60     }
61     print_inorder(root->left);
62     printf("%d ", root->data);
63     print_inorder(root->right);
64 }
65
66 void print_preorder(struct node *root)
67 {
68     if (root == NULL)
69     {
70         return;
71     }
72     printf("%d ", root->data);
73     print_preorder(root->left);
74     print_preorder(root->right);
75 }

```

```

76
77 void print_postorder(struct node *root)
78 {
79     if (root == NULL)
80     {
81         return;
82     }
83     print_postorder(root->left);
84     print_postorder(root->right);
85     printf("%d ", root->data);
86 }
87
88 int main()
89 {
90     int inorder[] = {1, 8, 19, 13, 25, 9, 5, 10, 4, 3};
91     int preorder[] = {25, 8, 1, 13, 19, 5, 9, 4, 10, 3};
92     int size = sizeof(inorder) / sizeof(inorder[0]);
93     int index = 0;
94     struct node *root = buildtree(preorder, inorder, 0, size - 1, &index);
95     printf("Inorder traversal : ");
96     print_inorder(root);
97     printf("\n");
98     printf("Preorder traversal : ");
99     print_preorder(root);
100    printf("\n");
101    printf("Postorder traversal : ");
102    print_postorder(root);
103    printf("\n");
104    return 0;
105 }
106

```

#### TERMINAL

```

Inorder traversal : 1 8 19 13 25 9 5 10 4 3
Preorder traversal : 25 8 1 13 19 5 9 4 10 3
Postorder traversal : 1 19 13 8 9 10 3 4 5 25

```

PS C:\Users\shuvr\OneDrive\Documents\CODING\College C codes>