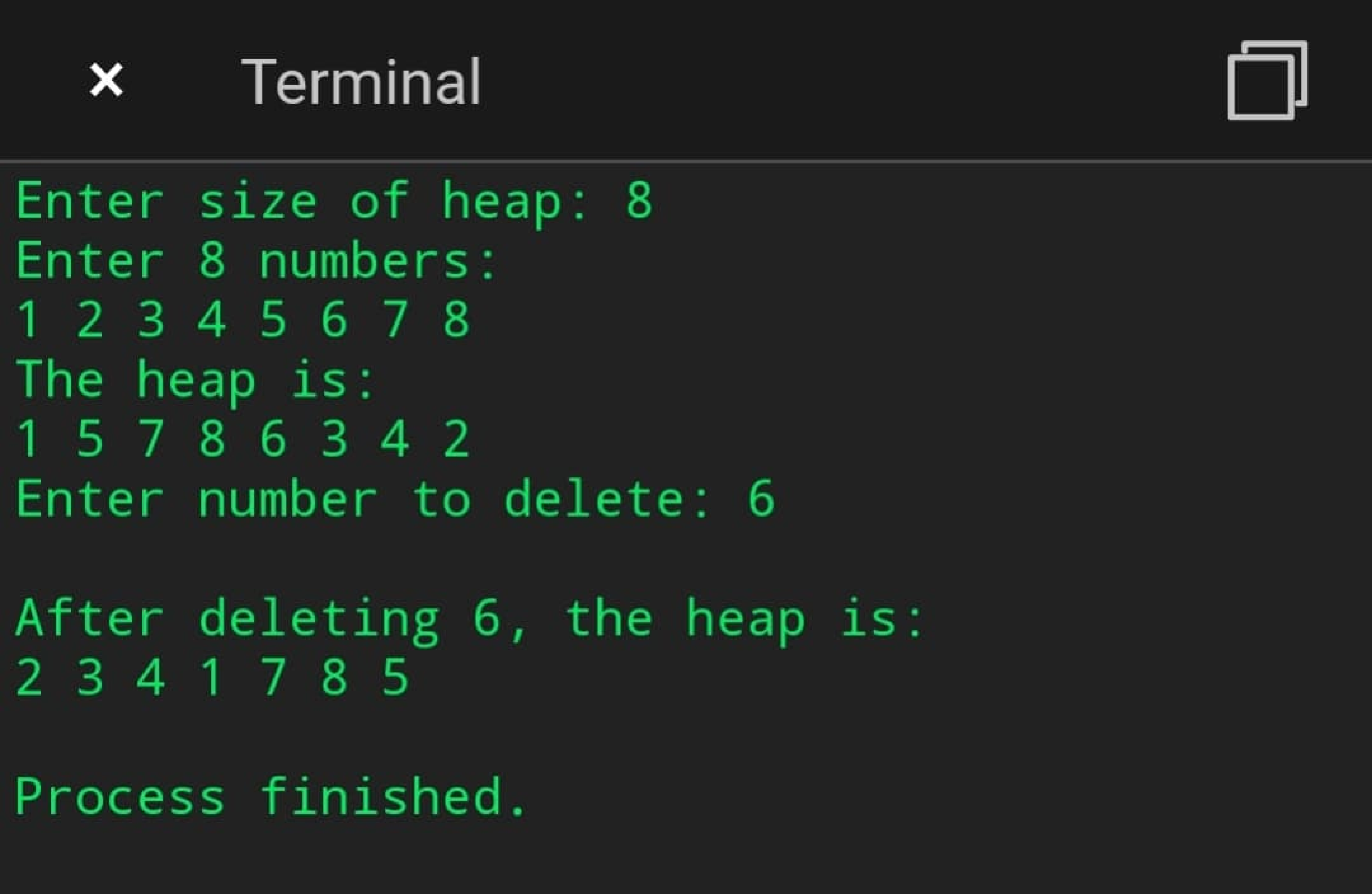
Delete_Dec_BinomialHeap.cpp

Saved

```cpp
#include <bits/stdc++.h>

#include <iostream>
using namespace std;

struct Node {
    int val, degree;
    Node *parent, *child, *sibling;
};
Node *root = NULL;

void binomialLink(Node *h1, Node *h2) {
    h1->parent = h2;
    h1->sibling = h2->child;
    h2->child = h1;
    h2->degree = h2->degree + 1;
}

Node *createNode(int n) {
    Node *new_node = new Node;
    new_node->val = n;
    new_node->parent = NULL;
    new_node->sibling = NULL;
    new_node->child = NULL;
    new_node->degree = 0;
    return new_node;
}

Node *mergeBHeaps(Node *h1, Node *h2) {
    if (h1 == NULL) return h2;
    if (h2 == NULL) return h1;

    Node *res = NULL;

    if (h1->degree <= h2->degree)
        res = h1;

    else if (h1->degree > h2->degree)
        res = h2;

    while (h1 != NULL && h2 != NULL) {
        if (h1->degree < h2->degree)
            h1 = h1->sibling;

        else if (h1->degree == h2->degree) {
            Node *sib = h1->sibling;
            h1->sibling = h2;
            h1 = sib;
        } else {
            Node *sib = h2->sibling;
            h2->sibling = h1;
            h2 = sib;
        }
    }
    return res;
}

Node *unionBHeaps(Node *h1, Node *h2) {
    if (h1 == NULL && h2 == NULL) return NULL;

    Node *res = mergeBHeaps(h1, h2);

    Node *prev = NULL, *curr = res, *next = curr->sibling;
    while (next != NULL) {
        if ((curr->degree != next->degree) || ((next->sibling != NULL) && (next->sibling)->degree == curr->degre
            prev = curr;
            curr = next;
        } else {
            if (curr->val <= next->val) {
                curr->sibling = next->sibling;
                binomialLink(next, curr);
            } else {
                if (prev == NULL)
                    res = next;
                else
                    prev->sibling = next;
                binomialLink(curr, next);
                curr = next;
            }
        }
        next = curr->sibling;
    }
    return res;
}

void binomialHeapInsert(int x) {
    root = unionBHeaps(root, createNode(x));
}

void display(Node *h) {
    while (h) {
        cout << h->val << " ";
        display(h->child);
        h = h->sibling;
    }
}

void revertList(Node *h) {
    if (h->sibling != NULL) {
        revertList(h->sibling);
        (h->sibling)->sibling = h;
    } else
        root = h;
}
```

Delete_Dec_BinomialHeap.cpp
Saved

```cpp

Node *extractMinBHeap(Node *h) {
    if (h == NULL) return NULL;

    Node *min_node_prev = NULL;
    Node *min_node = h;

    int min = h->val;
    Node *curr = h;
    while (curr->sibling != NULL) {
        if ((curr->sibling)->val < min) {
            min = (curr->sibling)->val;
            min_node_prev = curr;
            min_node = curr->sibling;
        }
        curr = curr->sibling;
    }
    if (min_node_prev == NULL && min_node->sibling == NULL)
        h = NULL;

    else if (min_node_prev == NULL)
        h = min_node->sibling;

    else
        min_node_prev->sibling = min_node->sibling;

    if (min_node->child != NULL) {
        revertList(min_node->child);
        (min_node->child)->sibling = NULL;
    }

    return unionBHeaps(h, root);
}

Node *findNode(Node *h, int val) {
    if (h == NULL) return NULL;

    if (h->val == val) return h;

    Node *res = findNode(h->child, val);
    if (res != NULL) return res;

    return findNode(h->sibling, val);
}

void decreaseKeyBHeap(Node *H, int old_val, int new_val) {
    Node *node = findNode(H, old_val);
    if (node == NULL) return;

    node->val = new_val;
    Node *parent = node->parent;

    while (parent != NULL && node->val < parent->val) {
        swap(node->val, parent->val);
        node = parent;
        parent = parent->parent;
    }
}

Node *binomialHeapDelete(Node *h, int val) {
    if (h == NULL) return NULL;
    decreaseKeyBHeap(h, val, INT_MIN);
    return extractMinBHeap(h);
}

int main() {
    int k, m, n;
    cout << "Enter size of heap: ";
    cin >> n;
    cout << "Enter " << n << " numbers: " << endl;
    for (int i = 0; i < n; ++i) {
        cin >> k;
        binomialHeapInsert(k);
    }

    cout << "The heap is:\n";
    display(root);
    cout << "\nEnter number to delete: ";
    cin >> m;
    root = binomialHeapDelete(root, m);

    cout << "\nAfter deleting " << m << ", the heap is:\n";

    display(root);
    cout << endl;
    return 0;
}
```



```
Enter size of heap: 8
Enter 8 numbers:
1 2 3 4 5 6 7 8
The heap is:
1 5 7 8 6 3 4 2
Enter number to delete: 6

After deleting 6, the heap is:
2 3 4 1 7 8 5


Process finished.
```