heapsort in c++

Here is an implementation of **Heapsort** in C++:

cpp Copy code

```cpp
#include <iostream>
using namespace std;

// Function to heapify a subtree rooted at index i
void heapify(int arr[], int n, int i) {
    int largest = i; // Initialize largest as root
    int left = 2 * i + 1; // left child index
    int right = 2 * i + 2; // right child index

    // Check if left child exists and is greater than root
    if (left < n && arr[left] > arr[largest]) {
        largest = left;
    }

    // Check if right child exists and is greater than root or left child
    if (right < n && arr[right] > arr[largest]) {
        largest = right;
    }

    // If largest is not root, swap and heapify the affected subtree
    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}

// Main function to perform heapsort
void heapSort(int arr[], int n) {
    // Build a max-heap (rearrange the array into a heap)
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, n, i);
    }

    // One by one extract elements from the heap
    for (int i = n - 1; i >= 0; i--) {
        // Move current root to end
        swap(arr[0], arr[i]);

        // Call heapify on the reduced heap
        heapify(arr, i, 0);
    }
}

// Function to print the array
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

// Driver program to test above functions
int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Unsorted array: ";
    printArray(arr, n);

    heapSort(arr, n);
```

```
    cout << "Sorted array: ";
    printArray(arr, n);

    return 0;
}
```

Explanation:

1. **Heapify**: This function is used to ensure the subtree rooted at a given index `i` maintains the heap property (max-heap property in this case). It does this by comparing the root with its left and right children and swapping with the largest child if necessary, and recursively calling `heapify` to ensure the subtree remains a heap.
2. **HeapSort**:
   - **Building the heap**: We start by building a max heap from the given array. This is done by calling `heapify` on each non-leaf node (starting from the last non-leaf node and going up to the root).
   - **Sorting**: After the heap is built, the largest element is at the root. We swap the root with the last element of the heap and reduce the heap size by one. Then we call `heapify` on the root again to restore the heap property for the remaining heap.
3. **printArray**: This function is used to print the elements of the array.

Example:

For the input array `{12, 11, 13, 5, 6, 7}`, the output will be:

c Copy code

```
Unsorted array: 12 11 13 5 6 7
Sorted array: 5 6 7 11 12 13
```

Time Complexity:

- **Building the heap**: O(n)
- **Heapify operations**: O(log n) for each element, and since we perform heapify n times, the overall complexity is O(n log n).
- Thus, the overall time complexity of Heapsort is **O(n log n)**.