Heap sort algorithm

Name: Raneem Montaser Ibrahim

Sec:3 CS department

Id:1000276904

(a): Required Algorithms

1. **Heapify Function**:
   Ensures the max-heap property, where the parent node is greater than or equal to its child nodes.

2. **Build-Heap Function**:
   Transforms an input array into a max-heap.

3. **Heap-Sort Function**:
   Repeatedly extracts the maximum element from the heap and reduces the heap size to achieve a sorted array.

(b): Analysis

1. Heapify Algorithm

- **Purpose**: Adjusts the subtree rooted at an index to satisfy the max-heap property.
- **Time Complexity**:
  - **Worst Case**: $O(\log n)$, as the function may traverse the height of the heap, which is $\log n$ for a binary heap.
  - **Best Case**: $O(1)$, if the element at the root is already greater than its children, no swaps are required.

2. Build-Heap Algorithm

- **Purpose**: Constructs a max-heap by calling the `Heapify` function for all non-leaf nodes.
- **Time Complexity**:
  - The number of calls to `Heapify` decreases as you go deeper into the heap tree.
  - The loop runs for n/2 nodes (all non-leaf nodes), and each call to Heapify takes O(log n).
  - Total complexity: $O(n)$

3. Heap-Sort Algorithm

- **Purpose**: Sorts the array after constructing the heap by extracting the maximum element one by one and maintaining the heap property.
- **Time Complexity**:

    - Each extraction involves a call to `Heapify`, which takes $O(\log n)$.
    - For n elements, the total sorting time is $O(n\log n)$.

Overall Time Complexity

The combined complexity of heap construction and sorting is:

$$O(n) + O(n\log n) = O(n\log n)$$

(c): Implementation

C# implementation of Heap-Sort:

```
using System;

class HeapSort
{
    static void Main(string[] args)
    {
        int[] arr = { 17, 21, 13, 5, 2, 19 };
        Console.WriteLine("Original array: " + string.Join(", ", arr));

        PerformHeapSort(arr);
        Console.WriteLine("Sorted array: " + string.Join(", ", arr));
    }

    static void PerformHeapSort(int[] arr)
    {
        int n = arr.Length;


        for (int i = n / 2 - 1; i >= 0; i--)
        {
            Heapify(arr, n, i);
        }


        for (int i = n - 1; i >= 0; i--)
        {
            Swap(arr, 0, i);
            Heapify(arr, i, 0);
        }
    }

    static void Heapify(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, largest);
            Heapify(arr, n, largest);
        }
    }

    static void Swap(int[] arr, int i, int j)
    {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}
```

Output Example

Input:

Original array: [17, 21, 13, 5, 2, 19]

Output:

Sorted array: [2,5,13,17,19,21]