Heap Sort Technique

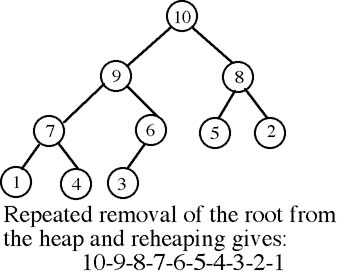
Algorithm

**Description:**

All Priority Queue Sorting is based on arranging the data structures into a priority queue using the keys. Because of its structure a Heap is automatically a priority queue. Repeatedly dequeuing structures from the heap and moving them into an array will give a sorted array.

**Data holder:** Heap of data structures and an empty array of data structures.  
  
**Technique:**

1. Build a heap from the given data.
2. Copy the data in the root node of the heap (it will be the one with the largest key or greatest priority) and place it at the end of the sorted data (assuming the sorted array is filled from right to left).



1. Reheap the heap.
2. Repeat until the heap is empty.

**Analysis:** At first, it would seem that because of the shifting and re-building of the heap that this sort would take as long, or longer, than Selection Sort. Surprisingly this sort can run in O(n log n).

Heap Sort Code Example

Heap Sort

**Heap Sort**

**Sort Type: Comparison-Based Sorting / Priority Queue Sorting**

**Algorithm**

**Description:**

All Priority Queue Sorting is based on arranging the data structures into a priority queue using the keys. Because of its structure a Heap is automatically a priority queue. Repeatedly dequeuing structures from the heap and moving them into an array will give a sorted array.

**Data holder:** Heap of data structures and an empty array of data structures.   
  
**Technique:**

1. Build a heap from the given data.
2. Copy the data in the root node of the heap (it will be the one with the largest key or greatest priority) and place it at the end of the sorted data (assuming the sorted array is filled from right to left).
3. Reheap the heap.
4. Repeat until the heap is empty.

**Analysis:** At first, it would seem that because of the shifting and re-building of the heap that this sort would take as long, or longer, than Selection Sort. Surprisingly this sort can run in O(n log n).

//implementation of Heap Sort

#include <iostream>

using namespace std;

// To heapify a subtree rooted with node i which is

// an index in arr[]. n is size of heap

void heapify(int arr[], int n, int i)

{

    int largest = i; // Initialize largest as root

    int l = 2\*i + 1; // left = 2\*i + 1

    int r = 2\*i + 2; // right = 2\*i + 2

    // If left child is larger than root

    if (l < n && arr[l] > arr[largest])

        largest = l;

    // If right child is larger than largest so far

    if (r < n && arr[r] > arr[largest])

        largest = r;

    // If largest is not root

    if (largest != i)

    {

        swap(arr[i], arr[largest]);

        // Recursively heapify the affected sub-tree

        heapify(arr, n, largest);

    }

}

// main function to do heap sort

void heapSort(int arr[], int n)

{

    // Build heap (rearrange array)

    for (int i = n / 2 - 1; i >= 0; i--)

        heapify(arr, n, i);

    // One by one extract an element from heap

    for (int i=n-1; i>=0; i--)

    {

        // Move current root to end

        swap(arr[0], arr[i]);

        // call max heapify on the reduced heap

        heapify(arr, i, 0);

    }

}

/\* A utility function to print array of size n \*/

void printArray(int arr[], int n)

{

    for (int i=0; i<n; ++i)

        cout << arr[i] << " ";

    cout << "\n";

}

// Driver program

int main()

{

    int arr[] = {12, 11, 13, 5, 6, 7};

    int n = sizeof(arr)/sizeof(arr[0]);

    heapSort(arr, n);

    cout << "Sorted array is \n";

    printArray(arr, n);

}