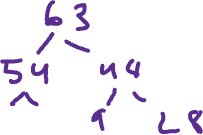
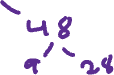
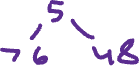
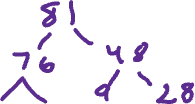
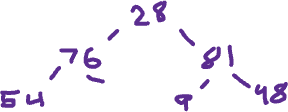
Max-Heap

Simulate the heapsort algorithm manually by sorting the following 9 element array:

Original maxheap heapSort

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [0] | 5 |  | 92 |  | 5 |
| [1] | 22 |  | 76 |  | 9 |
| [2] | 9 |  | 81 |  | 22 |
| [3] | 76 |  | 54 |  | 28 |
| [4] | 63 |  | 63 |  | 48 |
| [5] | 81 |  | 9 |  | 54 |
| [6] | 48 |  | 48 |  | 63 |
| [7] | 92 |  | 22 |  | 76 |
| [8] | 54 |  | 5 |  | 81 |
| [9] | 28 |  | 28 |  | 92 |



Code Portion:

import java.util.Arrays;

//Consider an array of elements that represents either a Min or Max heap.

//If you were to replace the value at index i with a new value, chances are, it's no longer a heap

public class Activity71 {

public static void main(String[] args) {

int[] minHeap = {100, 90, 80, 55, 60, 73, 3};

System.out.println("Original "+Arrays.toString(minHeap));

//alter the original heap

minHeap[generate(0, 6)] = generate(1, 100);

System.out.println("this is new heap: " + Arrays.toString(minHeap));

//reheap to minHeap, with new value

heapSort(minHeap);

//final max

System.out.println( "new heap: " + Arrays.toString(minHeap));

}

public static void heapSort(int[] heap) {

if(heap.length == 0) {

return;

}

for(int i = (heap.length / 2) - 1; i >= 0; i--) {

reheap(heap, i, heap.length);

}

for(int i = heap.length - 1; i >= 0; i--) {

//swap element

int temp = heap[0];

heap[0] = heap[i];

heap[i] = temp;

reheap(heap, 0, i);

}

}

public static void reheap(int[] heap, int checkCurrentNode, int TotalIndex) {

int root = checkCurrentNode;

int leftChild = (2 \* checkCurrentNode) + 1;

int rightChild = 2 \* (checkCurrentNode + 1); // leftChild + 1;

if(leftChild < TotalIndex && heap[leftChild] > heap[root]) {

root = leftChild;

}else if (rightChild < TotalIndex && heap[rightChild] > heap[root]) {

root = rightChild;

}

if(root != checkCurrentNode) {

//swap element

int temp = heap[checkCurrentNode];

heap[checkCurrentNode] = heap[root];

heap[root] = temp;

//continue recursivly through each node

reheap(heap, checkCurrentNode, root);

}

}

public static int generate(int min, int max) {

return (int) (Math.random() \* (max + min)) - min;

}

}

Write-up:

On paper reheap from deletion or adding is very clear in my head. Writing it out on code is another ball game. But the article and video across the web were very useful. Recursion is the name of the game for sorting heaps. Constant comparing through the array is solved through loops and recursion. The main problem was visually imagining what exactly what is call when the method is called again. I found that the method is checking the values with the next node.