Karim Sammouri
CSE 374 – A
Dr. Philippe Giabbanelli
6 May 2021
Assignment #5

Section III: Explaining and Illustrating the Use of a Heap

1) In your own words, briefly explain how heap sort works.

Ans: My response assumes we want to sort a given array in ascending order. Heap sort starts by transforming the given array into a max heap. Since we have a max heap, the max will be the first node of the tree. We'll swap the max with the last element of the array. The last element is now sorted. We'll have a pointer that tracks the last element of the *unsorted* part of the array and decrements after swapping the max with the last element, creating a partition that separates the unsorted and sorted parts of the array. Since just the first node of the tree will be out of place, we can use heapify to turn the unsorted part of the array into a max heap. Consequently, the max will again be the first node of the tree, and we can swap it with the last element of the unsorted part of the array. We repeat this process until we have one element in the unsorted part of the array by which time we know the array is fully sorted.

In the context of the array [12, 6, 10, 5, 1, 9], we'll first attempt to turn the array into a max heap. Since the given array is a valid max heap, we can continue using it in our example, excluding the empty first spot for the sake of simplicity. Since it's a max heap, the max (12) will be at the front. We'll swap the max with the last element (9), resulting in the array [9, 6, 10, 5, 1, **12**]. This creates a partition and divides the array between an unsorted part (9 to 1) and a sorted part (12). We'll turn the unsorted part into a max heap using heapify so we can get the max in the front again, resulting in [10, 6, 9, 5, 1, **12**]. We'll swap the max (10) with the last element of the unsorted part of the array (1), resulting in [1, 6, 9, 5, **10, 12**]. Last two elements are now sorted. We repeat this process to get [9, 6, 1, 5, **10, 12**] after using heapify to put the max of the unsorted part (9) at the front, [5, 6, 1, **9, 10, 12**] after swapping the max (9) with the last element of the unsorted part (5), [6, 5, 1, **9, 10, 12**] after using heapify to put the max of the unsorted part (6) at the front, [1, 5, **6, 9, 10, 12**] after swapping the max (6) with the last element of the unsorted part (1), [5, 1, **6, 9, 10, 12**] after using heapify to put the max (5) of the unsorted part at the front, [**1**, **5, 6, 9, 10, 12**] after swapping the max (5) with the last element of the unsorted part of the array (1). And since we have one element in the unsorted part of the array (1), we realize the whole array has been sorted and we stop.

In the context of the array [1, 14, 7, 8, 3], we'll again first attempt to turn the array into a max heap, resulting in [14, 8, 7, 1, 3]. Since it's a max heap, the max (14) will be at the front. We'll swap the max with the last element (3), resulting in the array [3, 8, 7, 5, 1, **14**]. This creates a partition and divides the array between an unsorted part (3 to 8) and a sorted part (14). We'll turn the unsorted part into a max heap using heapify so we can get the max in the front again, resulting in [8, 5, 7, 3, 1, **14**]. We'll swap the max (8) with the last element of the unsorted part of the array (1), resulting in [1, 5, 7, 3, **8, 14**]. Last two elements are now sorted. We repeat this process to get [7, 5, 1, 3, **8, 14**] after using heapify to put the max of the unsorted part (7) at the front, [3, 5, 1, **7, 8, 14**] after swapping the max (7) with the last element of the unsorted part (3), [5, 3, 1, **7, 8, 14**] after using heapify to put the max of the unsorted part (5) at the front, [1, 3, **5, 7, 8, 14**] after swapping the max (5) with the last element of the unsorted part (1), [3, 1, **5, 7, 8, 14**] after using heapify to put the max (3) of the unsorted part at the front, [**1, 3, 5, 7, 8, 14**] after swapping the max (3) with the last element of the unsorted part of the array (1). And since we have one element in the unsorted part of the array (1), we realize the whole array has been sorted and we stop.

2) **Explain whether we can turn a min-heap into a Binary Search Tree (BST) in Θ(logn).**

Ans: We cannot turn a min-heap into a Binary Search Tree in theta(logn). It takes theta(logn) to traverse just a path in a min-heap. To traverse every element of the min-heap, we'll have to traverse the whole array that backs it which would take theta(n).