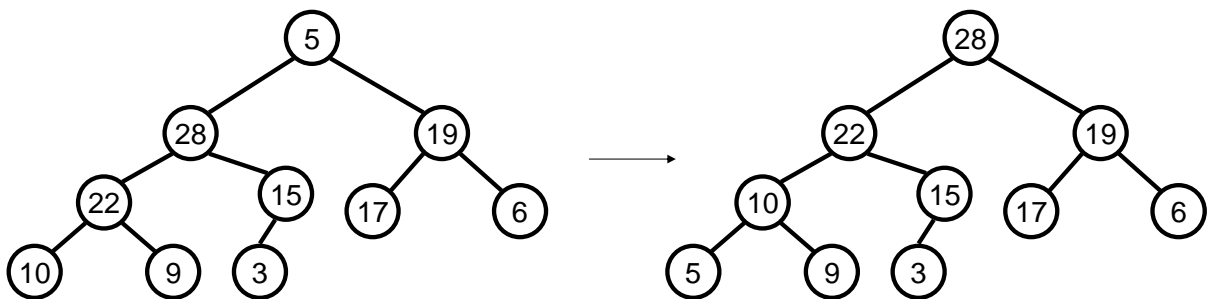
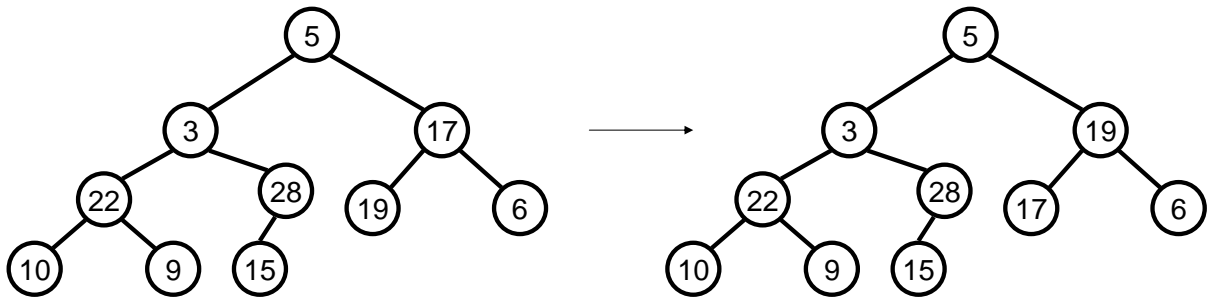
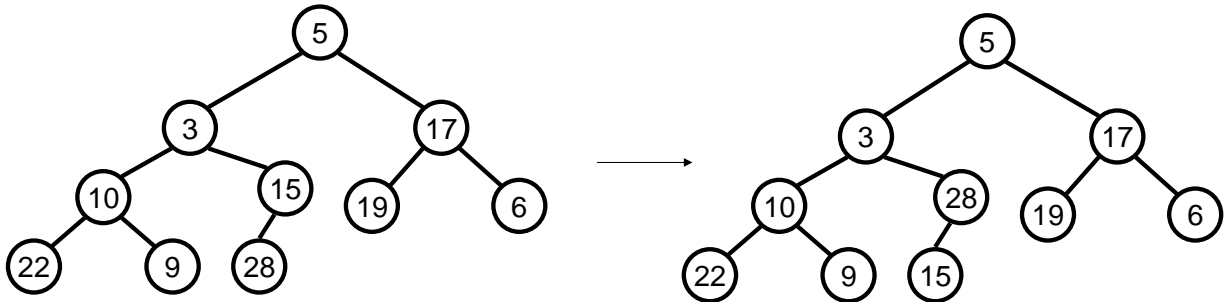


CS 3343 (Spring 2008) Assignment 6

Solution

1. (40 points) Heap and heap sort.

- a. (15 points) Study the pseudocode of BuildHeap and Heapify on Slide #11 and #23 in Lecture 13. Use Slides #24 – 36 as a model, illustrate the operation of BuildHeap on array $A = [5\ 3\ 17\ 10\ 15\ 19\ 6\ 22\ 9\ 28]$. To save space and time, you only need to show the content of the tree after each call to heapify(). (That is, slides 24, 25, 27, 29, 32, 36). The following partially filled trees are provided for your convenience. Make sure your final result is indeed a heap.



- b. (5 points) Exercise # 6.3-2 in textbook page 135: Why do we want the loop index i in algorithm BuildHeap to decrease from $\lfloor \text{length}(A)/2 \rfloor$ to 1 rather than increase from 1 to $\lfloor \text{length}(A)/2 \rfloor$?

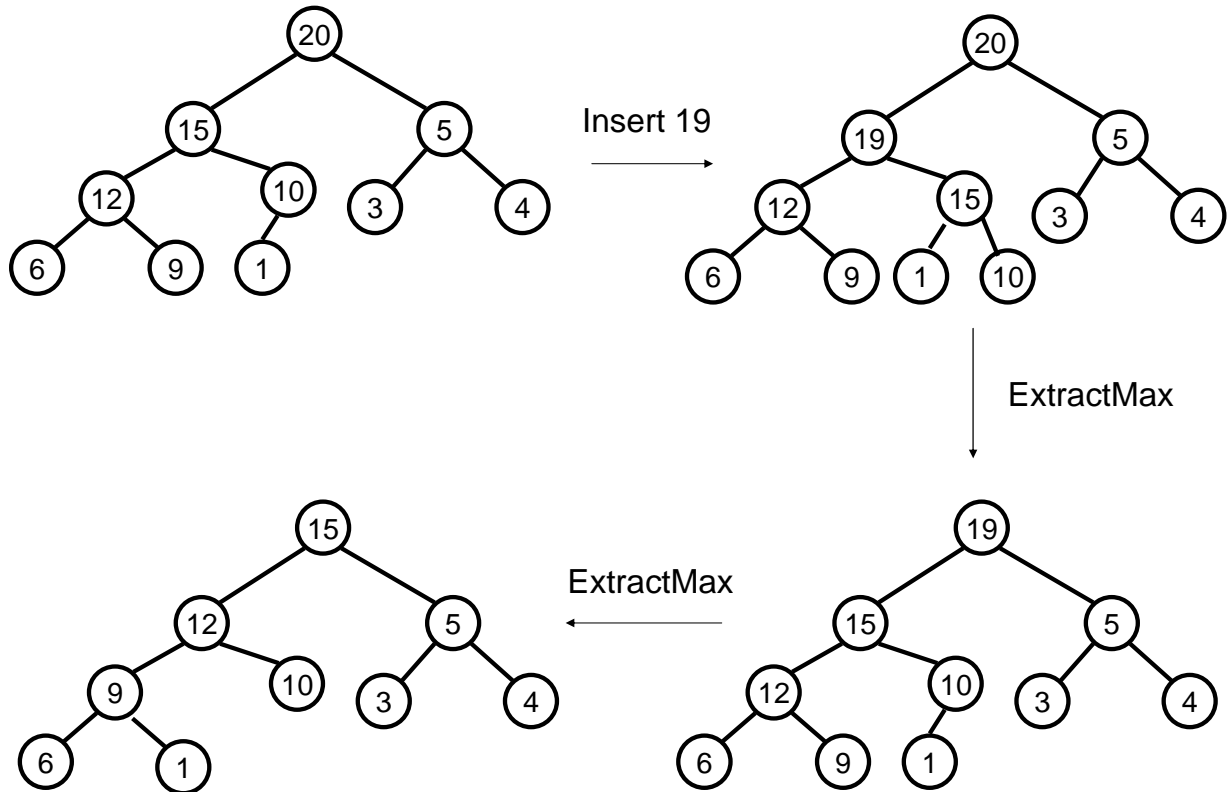
To call $\text{Heapify}(A, i)$, we need to make the assumption that the subtrees rooted at the left and right children of i are both heaps. This assumption is invalid if we start building the heap from $A[1]$.

- c. (10 points) Exercise # 6.4-3 in textbook page 136: What is the running time of heapsort on an array A of length n that is already sorted in increasing order? What about decreasing order? (Hint: first think about the cost for building heap in these two cases, then think about the cost for the actual sorting part. You can use examples $[0\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9]$ or $[9\ 8\ 7\ 6\ 5\ 4\ 3\ 2\ 1\ 0]$ to help you think.)

In either case, buildHeap will take $\Theta(n)$ time: to build a heap using an array that is sorted in increasing order, we have to sift every elements down (worst-case seneraio), which is in $\Theta(n)$; to build a heap using an array that is sorted in decreasing order also takes $\Theta(n)$ time because we have to call Heapify $n/2$ times, even though each call to Heapify will return immediately and takes constant time.

After the heap is constructed, each call to heapify takes $\Theta(\log n)$ time. Therefore the total cost for heap sort is $\Theta(n \log n)$, regardless of the initial order of the array.

- d. (10 points) Starting from the heap below, show the content of the new heap after each heap operation.



2. (30 points) Sorting algorithms.

You are given a list of short words. Each word has length exactly five, and uses only the following four letters: a, c, g, and t. For example, aaact, acgtc, gctac, etc. (In fact, those are called “DNA words” that control the genes in our cells.) You want to sort the words alphabetically. However, you do not have a library function to sort strings, and you are too busy to write one by yourself. Fortunately the library developers have implemented a collection of popular algorithms for sorting integer arrays, including merge sort, quicksort, counting sort, heap sort, insertion sort, selection sort, etc.

- a. How do you plan to use a function that sorts integers to sort those DNA words? (Hint: there are only $4^5 = 1024$ unique words.)

You can simply convert the strings to integers using a one-to-one mapping. For example, you may let $a = 0$; $c = 1$; $g = 2$; and $t = 3$, and treat each string as a base-4 integer: $aaaaa$ is 0, $aaaac$ is 1, $ttttt$ is 1023, etc. The conversion takes $\Theta(5n)$ time. If efficiency is not a concern, you can then use any sorting algorithm that takes integers to sort them.

- b. Which algorithm would you choose? More specifically, are there cases that you would prefer one algorithm but in other cases you would prefer another algorithm? (Note: I am not considering special cases that the words are almost sorted or things like that. Any combination of the letters is possible. Basically those quadratic algorithms are out of the question.)

If the number of words is much larger than 1024, you can use counting sort, since the time complexity is linear to the number of words. Otherwise quick sort or merge sort would be better.

Also as a side note, if the word length are much larger than 5, you cannot use counting sort directly, because the number of unique words is exponential to the word length. However you can use radix sort.

- c. If instead of sorting the words by alphabetic order, you decide that the four letters should have the order $a < g < t < c$, can you still use those algorithms to sort the words? Why or why not?

You just need to change the mapping, for example, $a = 0$, $g = 1$, $t = 2$, $c = 3$.