CS21003 ALGORITHMS-1
(Tutorial 6: Solutions)
Date: Oct 31 2020

1 Ugly Numbers

We will keep a min-heap and initially push 1 as the first ugly number.
Repeat this for n times :
Extract the minimum(m) from the min-heap .
Push $2 * m$, $3 * m$, $5 * m$ into the min-heap.
The n^{th} minimum extracted will be the n^{th} ugly number.

2 Median for stream of data

Maintain two heaps in the following way:
A max-heap to store the smaller half of the input numbers (contains $(n + 1)/2$ elements at any time).
A min-heap to store the larger half of the input numbers (contains $n/2$ elements at any time).
If the no.of elements are odd then the median will be the maximum value in the max-heap.
If no. of elements is even then median will be the average of maximum in max-heap and minimum in min-heap.

3 Ternary Heaps

Similar to Binary heaps

4 Linear-time Sorting

One way you can handle such cases is by doing a manual mapping.
Say you have array elements as : -9, 1, 2, 7, 3, 6, -1
Most negative element is -9.
So take index for -9 as 0(ie. $-9 + 9$)
Take index for 1 as $1 + 9 = 10$
index for 2 as $2 + 9 = 11$ and so on.
So by doing such index manipulation you can handle the negative numbers.

Date Sorting: Stable sorting using day, month and year, respectively. Apply counting sort for each.

For the circle question: Divide it into n concentric circles / rings from the center (equal area), apply bucket sort.

5 Priority Change

Let us add the facility to a priority queue that the priority of an item may change after insertion. Provide an algorithm *changePriority* that, given the index of an element in the supporting array and a new priority value, assigns the new priority value to the element, and reorganizes the array so that heap ordering is restored. Your algorithm should run in O(log n) time for a heap of n elements .

6 Maximum Sum Combination

1. Sort both arrays array A and array B.
2. Create a max heap to store the sum combinations along with the indices of elements from both arrays A and

B which make up the sum. Heap is ordered by the sum.
3. Initialize the heap with the maximum possible sum combination i.e ($A[N–1] + B[N–1]$ where N is the size of array) and with the indices of elements from both arrays (N – 1, N – 1). The tuple inside max heap will be ($A[N - 1] + B[N–1]$, N – 1, N – 1). Heap is ordered by first value i.e sum of both elements.
4. Pop the heap to get the current largest sum and along with the indices of the element that make up the sum. Let the tuple be (sum, i, j).

    4.1. Next insert ($A[i–1] + B[j]$, i–1, j) and ($A[i] + B[j–1]$, i, j – 1) into the max heap but make sure that the pair of indices i.e (i–1, j) and (i, j–1) are not already present in the max heap.

    4.2 Go back to 4 until K times.