CS60020: Foundations of Algorithm Design and Machine Learning
Assignment 3
Submission Deadline: 24th Feb 2018

1. You are given a list of airline tickets represented by pairs of departure and arrival airports [from, to]. Let the number of tickets be n with the following two assumptions
    a. The list of tickets is not cyclic, and
    b. There is one ticket from every city except final destination.

   The starting city is not known, and your task is to design an algorithm to find a valid itinerary, covering all cities, from the given set of tickets in **O(n)** time.

   Example:

   Suppose you are given four tickets [Shanghai, Delhi], [Lagos, Karachi], [Istanbul, Shanghai], [Karachi, Istanbul]. A valid itinerary from the given tickets is Lagos → Karachi → Istanbul → Shanghai → Delhi.

2. You are given a set of n ropes, each having different lengths. Your objective is to connect these ropes into one with minimum cost. The cost of connecting two ropes is equal to the sum of their lengths. Propose an algorithm which can perform the task in **O(n log n)** time.

   Example:

   Suppose you are given 4 ropes of lengths 4, 2, 1, and 5. You can connect the ropes in following ways:
    a. First connect ropes of lengths 1 and 2. Now you have three ropes of lengths 4, 5 and 3.
    b. Now connect ropes of lengths 3 and 4. Now you have two ropes of lengths 5 and 7.
    c. Finally connect the remaining two ropes.

   Total cost for connecting all ropes is 3 + 7 + 12 = 22. This is the optimized cost for connecting ropes. Other ways of connecting ropes would always have same or more cost. For example, if we connect 4 and 5 first (we get three strings of 2, 1 and 9), then connect 9 and 1 (we get two strings of 10 and 2). Finally we connect 10 and 2. Total cost in this way is 9 + 10 + 12 = 31.

3. You are given an array A[1..n] with a sliding window of size k which is moving from the very left of the array to the very right. Assume that you can only see the k numbers in the window at a particular time. Each time the sliding window moves rightwards by one position. Your task is to find the maximum for each and every

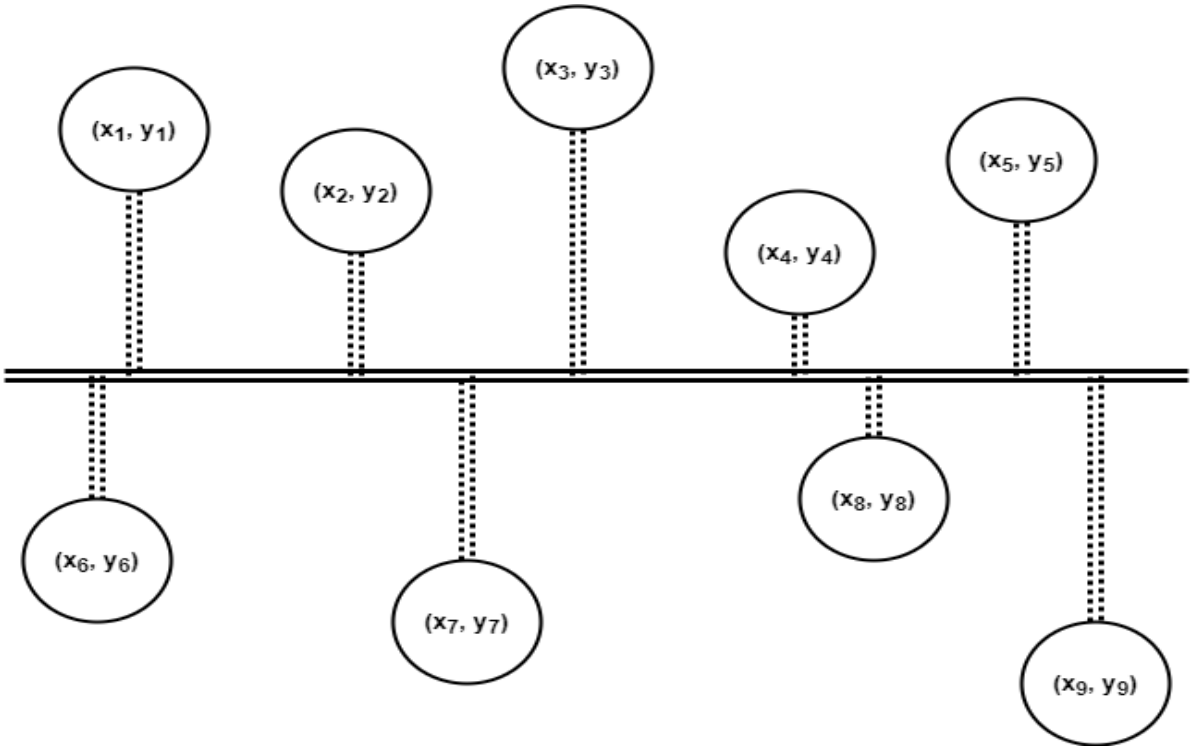
contiguous subarray of size k in **O(n log k)** time. [**Hint:** Use any self-balancing Binary Search Tree]

Example:
Input:
A[] = {2, 3, 4, 2, 5, 6, 3, 4, 7}
k = 4
Output:
4 5 6 6 6 7

Input :
A[] = {7, 4, 9, 6, 8, 3, 14, 11, 89, 12}
k = 3
Output :
9 9 9 8 14 14 89 89

4. Suppose you use Radix Sort to sort n integers in the range ($n^{k/2}$, n^k], for some k>0, which is independent of n. Derive the **time complexity** of the process?

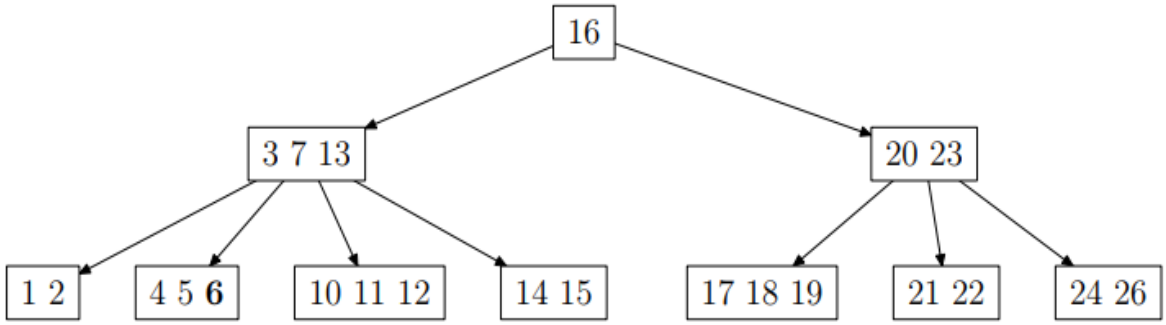
5. Suppose you are the CEO of an oil company and planning to build a large pipeline running from east to west through an oil field of n wells. Your objective is to connect each well directly to the main pipeline with smaller spur pipelines along the shortest route (either north or south), as shown in the following figure.



Let (x_1, y_1), (x_2, y_2), (x_3, y_3), …, (x_n, y_n) be the coordinates of the n wells. Design a **linear time** algorithm to determine the optimal location of the main pipeline, which would minimize the total length of the spurs.

6. a) Create a **Red-Black Tree** by inserting following sequence of numbers 8, 18, 5, 15, 17, 25, 40, 80. (No need to use any color. If a node with color *red* contains an element x, write it as **x:r**, write **x:b** for the *black* nodes.)

   b) Consider an initial **B-Tree** in the following figure.



   Delete the following elements one by one from the tree and show each step.
   6, 13, 7, 4, and 2.

7. Suppose you have an array of integers, where different integers may have different numbers of digits with a restriction that the total number of digits over *all* integers in the array is n. Your task is to design an algorithm to sort the array in **O(n)** time.

8. We know that the worst case time complexity of a *QuickSort* implementation is $O(n^2)$. The worst case occurs when the chosen pivot element is always an extreme (smallest or largest) element, which happens when the input array is sorted or reverse sorted, and either first or last element is picked as the pivot. *Randomized QuickSort* works well for a sorted array as well, but there is still a possibility that the randomly picked element is always an extreme. Can you design a modified QuickSort algorithm ($QuickSort_{mod}$) which theoretically reduces the worst case to **O(n log n)**? [**Hint:** Think about order statistics.]