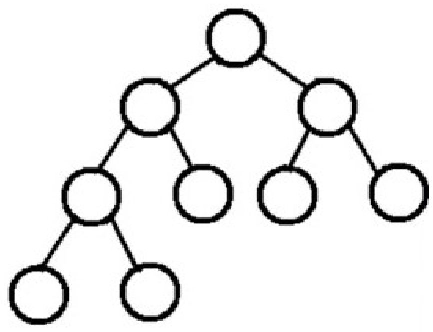
ALGORITHMS TEKDERS SINAVI

Question 1. We are given a complete binary tree where nodes and edges have **positive** weights. Node weights are stored in a *1*-dimensional array ***WN***. Edge weights are stored in a *2*-dimensional array ***WE*** where *0* denotes no edge.

Starting at the root of the tree and moving to either one of the children from the current node, **the goal** is to find the minimum total weight (i.e. sum of node and edge weights) path from the root to any one of the leaves.

Example:



Node weights array: ***WN*** = [3 4 2 6 1 9 8 8 5]

Edge weights array: ***WE*** =

$$\begin{bmatrix} 0 & 1 & 5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Output: Min total weight path includes nodes 1-2-5 with total weight 9.

1. Implement the greedy algorithm (i.e., write a function) of choosing the child with smallest sum of edge and node weights each time.

2. Implement a recursive algorithm (i.e., write a function) to find the minimum total weight. You must determine the input parameters. Also, give the time complexity of a recursive algorithm that implements this formulation? Show your work.

3. Implement a dynamic programming algorithm to solve the problem. You must determine the input parameters. Also, give the time complexity of your dynamic programming solution? Show your work.

4. In your main function:

   a. Show that the greedy algorithm does not solve this problem optimally.

   b. Run each of the recursive and dynamic functions with three different input sizes and compute the actual running times (in milliseconds or seconds) of these three algorithms. You will need to calculate the time passed before and after making a call to each function. Provide a 2x3 table involving the actual running times.

Question 2. You are given a sorted array **A** of *n* distinct integers, drawn from *1* to *m* where *n<m*. That is, **A** is a subset of *[1,2,...,m]*. Implement an *O(logn)* time algorithm (i.e., write a function) to find the smallest non-negative integer that is missing in *A* given as a parameter. For example, when **A** is *[1, 2, 3, 5, 7, 8, 10]*, the function must return *4*.

Question 3. Implement an *O(n)* algorithm that, given a set S of *n* distinct numbers and a positive integer *k≤n*, determines the *k* numbers in *S* that have the closest value to the median of *S*.

Question 4. Given an unsorted array, the array has this property that every element in array is at most k distance from its position in sorted array where *k* is a positive integer smaller than size of array. For example, let us consider *k* is *2*, an element at index *7* in the sorted array, can be at indexes *5*, *6*, *7*, *8*, *9* in the given array. We can sort such arrays more efficiently with the help of Heap data structure. Implement the following algorithm that uses a Heap:

- Create a Min Heap of size *k+1* with first *k+1* elements.
- One by one remove min element from heap, put it in result array, and add a new element to heap from remaining elements.

If you implement your own heap data structure rather than using available libraries, you will gain 10 bonus pts.