# CS21003 - Tutorial 2

## Solution Sketch

1. You are given keys 10, 22, 31, 4, 15, 28, 17, 88, 59. You need to insert these keys into a hash table of length $m = 11$ using open addressing with the auxiliary hash function $h(k, 0)$ = k rem m. Illustrate the result of inserting these keys using linear probing, using quadratic probing with $a = 1$ and $b = 3$, and using double hashing with $h_1(k)$ = k rem m and $h_2(k)$ = 1 + (k rem (m-1)).

   $\Rightarrow$ Try it yourself.

2. You are given two arrays $A$ and $B$ of integers of sizes $m$ and $n$. Your task is to check whether $A$ and $B$ are equal as sets. The arrays $A$ and $B$ need not be sorted, and may contain repeated occurrences of the same values. When we treat them as sets, all repetitions should be discarded (only one occurrence counts). For example, if $A$ = (5, 1, 2, 5, 1, 8, 1, 3) and $B$ = (2, 1, 8, 2, 5, 3), the answer is *Yes*, since both the arrays are equal to $\{1, 2, 3, 5, 8\}$ as sets. Design an algorithm to solve this problem in expected $O(m + n)$ time.

   $\Rightarrow$ The basic idea is to check first if $A \subseteq B$ and then if $B \subseteq A$. To check if $A \subseteq B$, we first hash all the elements of $B$ into a hashtable of size $n$. For each of the element of $A$, we now search in the hashtable of $B$. If any of the element is not found, the answer is no. Otherwise, we establish that $A \subseteq B$. A similar procedure is repeated to check if $B \subseteq A$.

3. You are given an array $A$ of $n$ integers. Your task is to find the number of sub-arrays such that $\sum_{k=i}^{j} a[k] = 0$ in expected $O(n)$.

   $\Rightarrow$ Consider the prefix sum array $p$ such that $p[i] = \sum_{j=1}^{i} a[i]$. Note that for $\sum_{k=i}^{j} a[k] = 0$, $p[i-1] = p[j]$. We use this idea to solve th problem as follows:

   We maintain a hash table which stores $(x, count)$ where $x$ is the key and count is the number of $i$ such that $p[i] = x$. We initialize a variable answer to the number of $i$ such that $p[i] = 0$ and iterate over array $a$ from 1 to n. At index $i$, we search for $p[i]$ in the hash table. If don't find $p[i]$, we just insert $(p[i], 1)$ in the hash table. If we find $p[i]$, let's it's corresponding value in the hash table be $c$, we add $c$ to the answer and update the value of $p[i]$ in the hash table to $c + 1$. Finally, answer contains the required number of sub-arrays.

4. You are given a rooted tree $T$. The width of $T$ is the maximum number of nodes at a level in the tree. For example, consider a tree of height three on ten nodes $a, b, c, d, e, f, g, h, i, j$, where $a$ is the root having three children $b, c, d$, node $b$ has two children $e, f$, node $d$ has three children $g, h, i$, and $h$ has one child $j$. In this tree, the numbers of nodes at levels $0, 1, 2, 3$ are respectively $1, 3, 5, 1$. The width of this tree is therefore 5. Design an algorithm to compute the width of $T$ in $O(n)$ time, where $n$ is the number of nodes in T.

   $\Rightarrow$ Solve for the simpler case when it is a binary tree, you may use $O(h)$ space to maintain the number of nodes at each level in the binary tree. Use preorder traversal.

5. Assume that a set $S$ of $n$ numbers is stored in some form of balanced binary search tree, i.e., the height of the tree is $O(logn)$. In addition to the key value and the pointers to children, assume that every node contains the number of nodes in its subtree. Design $O(logn)$ algorithms for performing the following operation:

Given a positive integer $k$, $1 \leq k \leq n$, compute the $k^{th}$ smallest element of $S$.

$\Rightarrow$ The idea is to use the elements in the left subtree recursively. Let $l$ be the number of elements in the left subtree (can be queried in $O(1)$ time). if $k = l + 1$, root is the answer, if $k < l$, search should proceed in the left subtree, if $k > l + 1$, search should proceed in the right subtree for finding $k - l - 1$th smallest element.