# Homework 03 — *COSC 311: Algorithms*, Fall 2022

**Due:** Friday, 10/21/2022 at 11:59 pm

**Exercise 1.** Suppose you are given an unsorted array $a$ of size $n-1$ where $n = 2^B$ that contains all of the numbers 0 through $n-1$ (in an arbitrary order) except for a single missing value $m$. The numbers are each represented in binary with $B$ bits, so that $a[i]$ stores the $i$th number, and $a[i][j]$ is the $j$th bit of $a[i]$. Use the divide and conquer strategy to devise an algorithm that finds the missing value $m$ using $O(n)$ bit comparison and swap operations. Use the Master Theorem to justify the running time of your procedure.

*Hint.* Note the similarity with the setup of RadixSort. While RadixSort uses $O(Bn)$ bit comparisons and swap operations, your algorithm must use only $O(n)$ such operations.

**Exercise 2.** In class, we saw Dijkstra's algorithm for the single-source shortest path problem:

```
Dijkstra(V, E, u):
1. initialize d[u] = 0 and [v] = infinity for all v != u
2. maintain set S of finalized nodes, initially empty
3. while S != V do:
     find node v in V - S with minimal d[v]
     add v to S
     for each neighbor x of v
       update d[x] <- min(d[x], d[v] + w(v, x))
     endfor
   endwhile
4. return d
```

For simplicity, we assumed that $V = \{1, 2, 3, \ldots, n\}$ so that $d$ is an array where $d[x]$ stores the (weighted) distance from $u$ to $x$. This array, however, does not give the actual path from $u$ to $x$, but just the *length* of the shortest such path. Given a shortest path from $u$ to $x$, $P = ue_1v_1e_2v_2 \ldots v_{k-1}e_kx$, we say that $v_{k-1}$ is $x$'s **parent**. That is, $x$'s parent is the next vertex from $x$ along the shortest path from $x$ to $u$. Observe that $x$'s parent $v$ is $x$'s neighbor satisfying $d[x] = d[v] + w(v, x)$.

1. Write a modified version of `Dijkstra` called `DijkstraPath` that returns an array $p$ such that for each vertex $x$, $p[x]$ stores $x$'s parent. Your algorithm should only differ from `Dijkstra` in a few lines of (pseudo)code.

2. Write a method `GetPath(p, u, x)` that given the array $p$ returned by `DijkstraPath`, `GetPath(p, u, x)` returns the shortest path from $u$ to $x$. That is, `GetPath(p, u, x)` should return an array `path` of length $k+1$ where `path[1] = u`, `path[k + 1] = x`, and $k$ is the number of hops on the shortest (weighted) path from $u$ to $x$ in $G$. The running time of `GetPath(p, u, x)` should be $O(k)$.