Note: So far we have seen selection sort, bubble sort, and insertion sort. All three of these take $O\left(n^2\right)$ time. But there are $O(n \lg n)$ time sorting algorithm, e.g., mergesort, which we shall learn later in the course. In these home work problems, if you need sorting, you may assume that you have an $O(n \lg n)$ sorting algorithm at hand.

1. (10 points) A firm wants to determine the highest floor of its n-story headquarters from which a gadget can fall without breaking. The firm has two identical gadgets to experiment with. If one of them gets broken, it cannot be repaired, and the experiment will have to be completed with the remaining gadget. Design as efficient (as few droppings as you need) an algorithm as you can to solve this problem.

2. (10 points) Apply topological sorting using Depth First Search on the following directed graph. Show

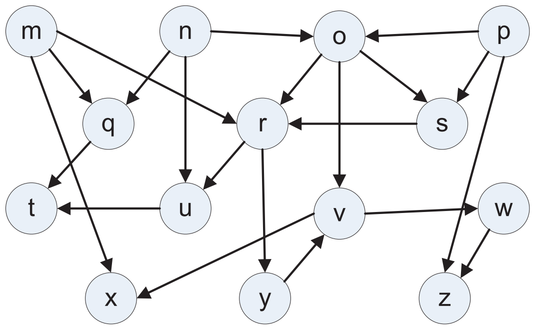

Figure 1: Digraph for Topological Sorting

the full stack with push and pop indices for each element, DFS forest (with various types of edges), and the final topological order of the vertices.

3. (10 points) There are two types of professional wrestlers: "babyfaces" ("good guys") and "heels" ("bad guys"). Between any pair of professional wrestlers, there may or may not be a rivalry. Suppose we have n professional wrestlers and we have a list of r pairs of wrestlers for which there are rivalries. Give an $O(n + r)$-time algorithm that determines whether it is possible to designate some of the wrestlers as babyfaces and the remainder as heels such that each rivalry is between a babyface and a heel. If it is possible to perform such a designation, your algorithm should produce it. Explain why do you think your algorithm's time efficiency is $O(n + r)$? [**Hint:** Think in terms of graphs.]

4. (10 points) One can model a maze by having a vertex for a starting point, a finishing point, dead ends, and all the points in the maze where more than one path can be taken, and then connecting the vertices according to the paths in the maze.

   (a) Construct such a graph for the maze in Figure 2.

   (b) Which traversal–DFS or BFS–would you use if you found yourself in a maze and why?

5. (10 points) Generate all permutations of $\{3, 5, 6, 9\}$ by

   (a) the bottom-up minimal-change algorithm.

   (b) the Johnson-Trotter algorithm (with arrows and the greatest mobile element marked).

   (c) the lexicographic-order algorithm (show i, $i + 1$, and j for each permutation as in the slides).

6. (10 points) Consider the pseudocode of the algorithm for generating permutations in the listing **Algorithm 1**:

   (a) Trace the algorithm by hand for $n = 2, 3$, and 4. [**Hint:** Try to reuse the work you did for $n = 2$ while you are working for $n = 3 \ldots$]

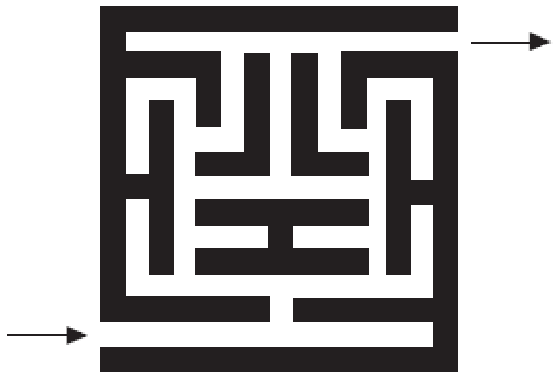

Figure 2: Maze

Algorithm 1 Permute(n)

```
{Input: A positive integer $n$ and a global array $A[1..n]$}
{Output: All permutations of elements of $A$}
if $n = 1$ then
  print $A$
else
  for $i \leftarrow 1$ to $n$ do
    $Permute(n-1)$
    if $n$ is odd then
      swap $A[1]$ and $A[n]$
    else
      swap $A[i]$ and $A[n]$
    end if
  end for
end if
```

(b) Prove the correctness of *Permute*(n). [**Hint:** Prove that if Permute($n-1$) works correctly, Permute(n) must also work correctly.]

(c) What is the time efficiency of *Permute*(n)? [**Hint:** The pseudocode gives you the recurrence relation (number of swaps), solve it. You may need the formula $e \approx \sum_{i=0}^{n} \frac{1}{i!}$ for large n.]

7. (10 points) Consider the following algorithm for searching in a sorted array $A[0..n-1]$. If $n = 1$, simply compare the search key K with the single element of the array; otherwise, search recursively by comparing K with $A\left[\lfloor \frac{n}{3} \rfloor\right]$, and if K is larger, compare it with $A\left[\lfloor \frac{2n}{3} \rfloor\right]$ to determine in which third of the array to continue search.

   (a) Set up a recurrence for the number of key comparisons in the worst-case. You may assume that $n = 3^k$.

   (b) Solve the recurrence for $n = 3^k$.

   (c) Compare the algorithm's efficiency with that of binary search.

8. (10 points) You need to search for a given number in an $n \times n$ matrix in which every row and every column is sorted in increasing order. Can you design a $O(n)$ algorithm for this problem? Express the algorithm in pseudocode. [**Hint:** Start comparing your key with the last element of the first row.]

9. (10 points) Suppose you are given an array A of n sorted numbers that has been *circularly shifted* k positions to the right. For example, $\langle 35, 42, 5, 15, 27, 29 \rangle$ is a sorted array that has been circularly shifted $k = 2$ positions, while $\langle 27, 29, 35, 42, 5, 15 \rangle$ has been shifted $k = 4$ positions.

(a) Suppose you know what k is. Give an as-efficient-as-you-can (AEAYC) algorithm to find the largest number in A. Express the algorithm in pseudocode.

(b) Suppose you *do not* know what k is. Give an AEAYC algorithm to find the largest number in A. Express the algorithm in pseudocode.

10. (10 points) Given a set S of n integers and an integer t, give an $O\left(n^{k-1} \lg n\right)$ algorithm to test whether k of the integers in S add up to t. Express the algorithm in pseudocode. [**Hint:** For example, when $k = 3$, the complexity must be $O\left(n^2 \lg n\right)$. In this case it is straightforward to devise an algorithm with $O\left(n^3\right)$ complexity. Think how you can turn $O(n^2 \cdot n)$ into $O(n^2 \cdot \lg n)$. Then generalize the idea for any $k > 1$.]