University of Waterloo
CS240 - Spring 2025
Programming question 1

Due Date: Tuesday June 10, 5pm

Please read the following link for guidelines on submission:

`https://student.cs.uwaterloo.ca/~cs240/assignments.phtml#guidelines`

Late Policy: Assignments are due at 5:00pm, with a grace period until 8:00pm.

Building decision trees

You are going to build the decision trees of some sorting algorithms. We will be working with insertion sort and selection sort (for vectors of int's); we give you an implementation of both of them. In all the assignment, you can suppose that the array entries we want to sort are all pairwise distinct.

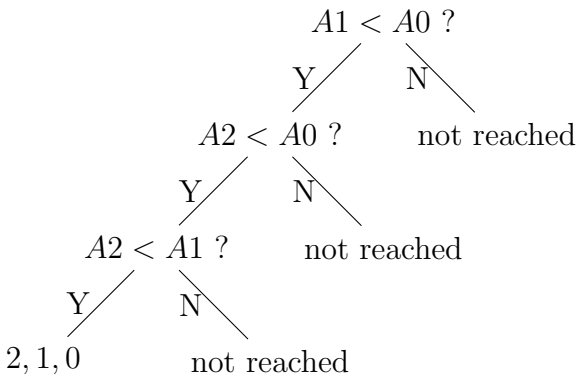
Once we have chosen our algorithm and an input size n, we can construct the corresponding decision tree.

- This is a binary tree, where each internal node has two children.
- Internal nodes are labelled by a comparison between two elements in the input array. For each comparison, we give the indices of the two arguments we compare as they were in the original array (the algorithm may have moved things around, but we make sure we give indices corresponding to the original layout). We indicate this by writing "$Ai < Aj$?". The left child is for "yes" cases, the right child for "no" cases.
- Leafs are labelled with either a permutation of $\{0, \ldots, n-1\}$ or the string "not reached". The first case corresponds to leafs v for which there exists an input A whose path in the decision tree ends at v. Then at v we store the sorting permutation of A (see below). If a leaf is not reachable, we label it with the string "not reached".

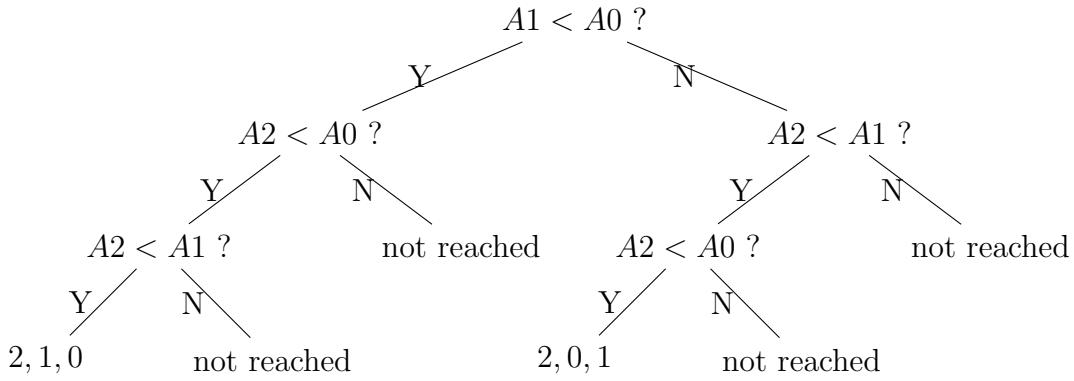
The sorting permutation of an array A (with distinct entries) tells us what is the order of the entries in A: if $A = [0, 2, 3, 1]$, its sorting permutation is $B = [0, 3, 1, 2]$: 0 is at position 0 in A, 1 is at position 3, 2 is at position 1 and 3 is at position 2. When A itself is a permutation, like here, B is the inverse permutation.

The algorithm you will use to construct the decision tree in size n does the following: start from the empty tree, and run your sorting algorithm on all permutations of $\{0, \ldots, n-1\}$. Each permutation gives you a path, which you add to the tree you are building. You will have to modify the sorting functions we give you to make this possible, but of course do not change the sorting algorithms themselves.

For example, for insertion sort with $n = 3$ and input $A = [2, 1, 0]$, you obtain this path:



If you next run the algorithm on $A = [1, 2, 0]$, you add a second path to the tree, which now looks like this:



Your code should read from `cin` a string s and an integer n. The string s can be either "insertion" or "selection". You should generate all permutations of $\{0, \ldots, n-1\}$ and use them to build the decision tree for the corresponding algorithm in size n. The only display should be the whole decision tree.

To display the tree, use *level order* traversal, from left to right. Each node should have an index, which is determined as we did for heaps (root has index 0, left child of node of index i has index $2i+1$, right child has index $2i+2$). For each internal node, print its index, what comparison we do, and the indices of its children. For a leaf node with a sorting permutation, print the index and this permutation (use space as separator). For a non-reached node, print the index and "not reached".

For the previous example, after only building the path for $[2, 1, 0]$, your tree would be displayed as this:

```
node 0 : A1 < A0 ? 1 : 2
node 1 : A2 < A0 ? 3 : 4
node 2 : not reached
```

```
node 3 : A2 < A1 ? 7 : 8
node 4 : not reached
node 7 : 2 1 0
node 8 : not reached
```

After we are done with all 6 permutations of $\{2, 1, 0\}$, given the following input file

```
insertion
3
```

the output of your program should be

```
node 0 : A1 < A0 ? 1 : 2
node 1 : A2 < A0 ? 3 : 4
node 2 : A2 < A1 ? 5 : 6
node 3 : A2 < A1 ? 7 : 8
node 4 : 1 0 2
node 5 : A2 < A0 ? 11 : 12
node 6 : 0 1 2
node 7 : 2 1 0
node 8 : 1 2 0
node 11 : 2 0 1
node 12 : 0 2 1
```

When Coding:
You can include and use `vector`, `tuple`, `stack`, `queue`, `priority_queue`, but not `algorithm`. We will only test you with the format:

```
insertion
X
```

OR

```
selection
X
```

where X is always a positive integer. We will also only test you one command in each test case. Starter code, or execute file (for I/O) will not be provided, unlike CS246(E). Please also carefully read *sort.cpp* before you start to implement.