Permutation Generation with Heap's Algorithm and the Steinhaus-Johnson-Trotter Algorithm

Ryan Bernstein
Ruben Niculcea
Levi Schoen

Contents

1 Heap's Algorithm

1.1 Time Complexity

The significant portion of our implementation of Heap's Algorithm is as follows:

```
 3 def heaps(set, time=False):
 4   def innerHeaps(n, set):
 5     if n == 1:
 6       print set
 7     else:
 8       for i in range(n):
 9         innerHeaps(n-1, set)
10         if n % 2 == 1: #odd number
11           j = 0
12         else:
13           j = i
14         set[j], set[n-1] = set[n-1], set[j] #swap
```

To find the total running time of our algorithm, we'll look at the number of times the following operations are executed:

- Swaps, an $O(1)$ operation
- Comparisons, an $O(1)$ operation
- Assignments, an $O(1)$ operation, and
- Processes (`prints` in our implementation), an $O(n)$ operation

From the code above, we can count how many times each of these operations is performed, *not* including work done within the recursive calls. To better facilitate our analysis, we will also include the number of recursive calls.

1.1.1 Swaps

The swap operation appears only once in our implementation, on line 14. This line is never executed when $n = 1$; if $n > 1$, the loop containing it executes n times.

1.1.2 Comparisons

A comparison appears on line 5 regardless of the current value of n. Another comparison appears within the loop on line 10, which executes n times for all $n > 1$. The second comparison also includes a division for modular arithmetic; we will also consider this an $O(n)$ operation.

1.1.3 Assignments

Not counting the assignments that appear within the swaps, a single assignment occurs on either line 11 or line 13. This is dependent on the result of the comparison on line 10.

1.1.4 Processes

We only process a permutation in the base case of our recursion, when $n = 1$. Thus we execute a process once when $n = 1$ and 0 times in any other case.

Our total number of operations (not counting recursion) can be represented by the following table:

n	1	2	3	4	5
Swaps	0	2	3	4	5
Comparisons	1	3	4	5	6
Assignments	0	2	3	4	5
Processes	1	0	0	0	0
Recursive calls	0	2	3	4	5

1.2 Space Complexity