## Abstract

This project deals with displaying various combinations given an upper bound and the number of numbers in a combination. The second part of this project deals with displaying the various permutations until a given number. The project implements permutations in two methods and compares their running times.

## Problem Statement

Implement the algorithms discussed in class:

(a) Combination(A, n, k)

(b) Permute(A, n). For Permute(), implement Take 2 and Heap's algorithms, and compare their running times for n = 8..14.

## Methodology

All three algorithms were discussed in class.

## Development Platform

Mac OS 10.11 El Captain, 128GB SSD, 8GB RAM, Intel Core i5 2.7GHz Processor, Java 1.7 and IntelliJ IDE

## Test Results

|  |  |  |
| --- | --- | --- |
| Value of “n” | Running Time for PermuteTake2 | Running Time for Heaps Algorithm |
| 8 | 0.84385 seconds | 0.178964 seconds |
| 9 | 5.138151 seconds | 1.378129 seconds |
| 10 | 55.249562 seconds | 11.788681 seconds |
| 11 | 689.388794 seconds | 149.784464 seconds |
| 12 |  |  |
| 13 |  |  |
| 14 |  |  |

For values of n = 12,13,14 the running time was too large to be analytically calculated. We can assume that it would grow exponentially.

## Discussion of Results

Based on the above running time results, it is clear both the algorithms are time consuming. But among the two, Heap’s Algorithm does significantly better for every use case.

## Conclusion

This project has helped me understand that calculating various permutations and combinations is still a taxing effort on the computer’s resources. Also, if you are computing permutations, it is better to use Heap’s algorithm.

## References

1. Heap’s Algorithm - <https://en.wikipedia.org/wiki/Heap%27s_algorithm>