California State University, Fullerton
CPSC 335 Project 2 Report
05/11/2021

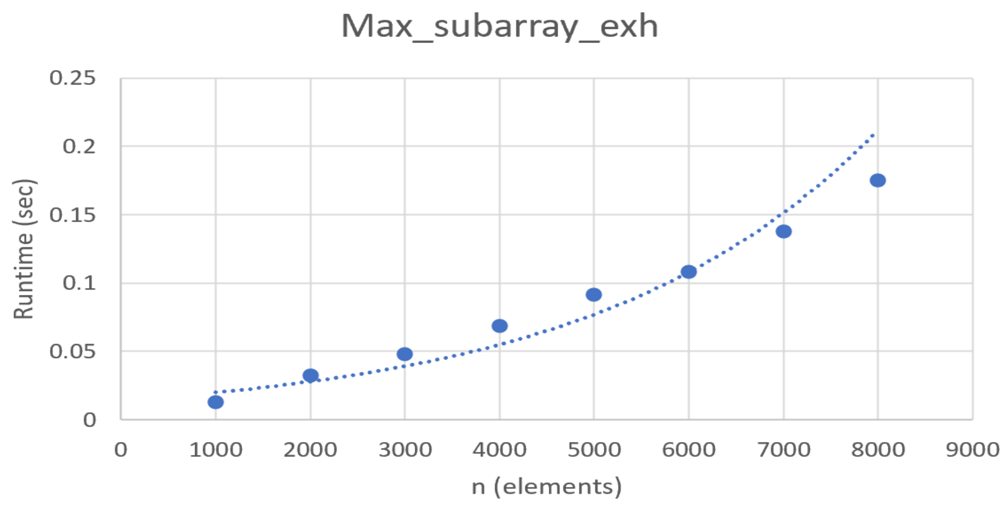
Submitted by:

Janelle Estabillo	estabillojanelle@csu.fullerton.edu
Nour Alqurain	nouralqurain@csu.fullerton.edu
Triet Le	tle877@csu.fullerton.edu

Submitted to:
Kevin Wortman

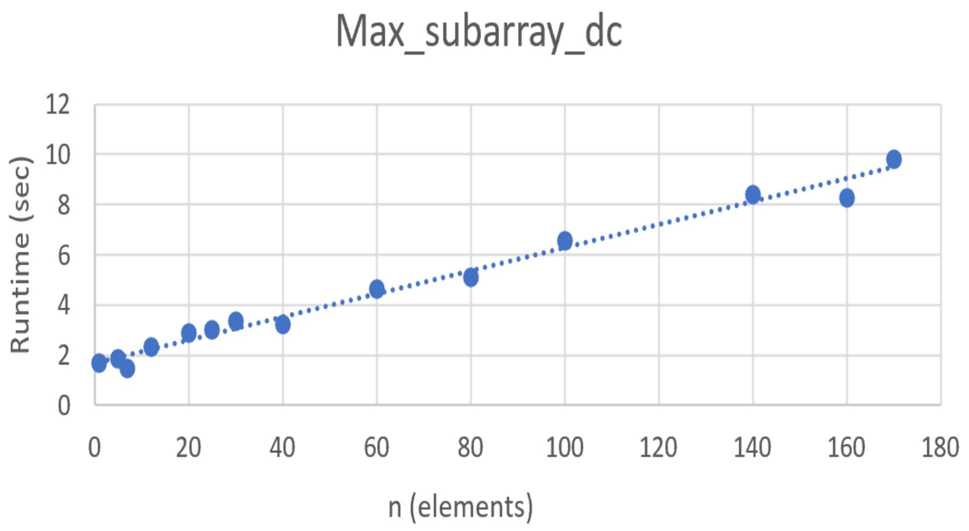
I. Scatter Plots

Maximum Subarray Problem



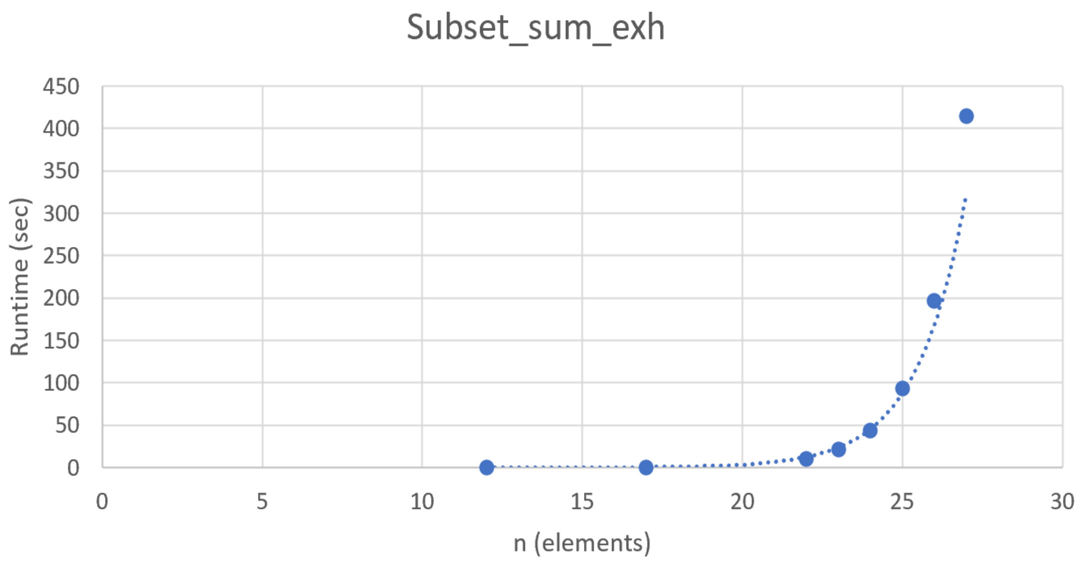
Our empirically-observed time efficiency data is consistent with our mathematically-derived big-O efficiency class.

Decrease-by-Half Algorithm



Our empirically-observed time efficiency data is consistent with our mathematically-derived big-O efficiency class.

The Subset Sum Problem



Our empirically-observed time efficiency data is consistent with our mathematically-derived big-O efficiency class.

II. Additional Questions

1. **Is there a noticeable difference in the performance of the three algorithms? Which is fastest, and by how much? Does this surprise you?**
   - Yes, there is a noticeable difference in the three algorithms. By looking at our empirical analyses and comparing them to the theory, we could see that for the subset_sum problem is the slowest algorithm . It is much slower than the rest of the algorithms while the maximum subarray problem using the decrease-by-half algorithm is the fastest.
   - This doesn't surprise us at all since we know from our hypothesis and from what we learn in the class that algorithms with exponential running times are extremely slow, probably too slow to be of practical use. There, the subset_sum problem should be the slowest algorithm as we expected.
   - On the other hand, the Maximum Subarray Problem using the decrease-by-half algorithm is the fastest out of the three algorithms having a time complexity of O(nlogn) and outperforming the two other algorithms by 0.236268s faster when n=10000 compared with it's exhaustive search counterpart.

2. **Are your empirical analyses consistent with the predicted big-O efficiency class for each algorithm? Justify your answer.**
   - As mentioned in the project brief, in theory, the O(n · 2^n) subset_sum algorithm will be far slower than the other algorithms' O(n^3) and O(nlogn) time complexities. This can be proven by looking at our empirical analyses, therefore

showing a consistent result from the predicted big-O efficiency class for each algorithm.

3. **Is this evidence consistent or inconsistent with hypothesis 1? Justify your answer.**
   - ➔ Hypothesis 1 states that, "Exhaustive search algorithms are feasible to implement, and produce correct outputs." The evidence we got from our analysis is proven to be consistent with our Hypothesis which can be seen on the time complexity and scatter plots of Maximum Subarray Problem and the Subset Sum Problem, both using exhaustive search. Both algorithms produced the desired output correctly and are feasible despite the algorithm being slower than its counterpart.

4. **Is this evidence consistent or inconsistent with hypothesis 2? Justify your answer.**
   - ➔ Hypothesis 2 states that, "Recursion is not always slow, because recursion-based algorithms can be faster than loop-based algorithms." Evidence proving this hypothesis is consistent with hypothesis 2 because we can clearly see that the runtime of the maximum_subaary algorithm using recursion is faster than the maximum_subaary algorithm using the loops. Thus, we can reach the conclusion that recursion-based algorithms are not always slower than loop-based algorithms.

5. **Is this evidence consistent or inconsistent with hypothesis 3? Justify your answer.**
   - ➔ Hypothesis 3 states that, "Algorithms with exponential or factorial running times are extremely slow, probably too slow to be of practical use." The evidence we got from the three algorithms proved the hypothesis to be consistent. The implementation of the subset_sum_exh problem, which has an exponential time complexity, showed an extremely slower running time than the implementation of the decrease-by-half algorithm and Exhaustive Search of the maximum subarray problem.We can see from the graph that the runtime of subset_sum_exh problem grows dramatically when the input size grows by just 1 or 2 elements. Thus, in real life, it is going to take forever for the exponential algorithm to solve a problem with a big input size.