Freya Varez

**Problem Statement**

The purpose of the tests below was to compare four different algorithms with respect to their time-complexities. These four algorithms take a sequence of integers and find the maximum sum of a subsequence within this integer set. The four algorithms are characterized below:

**Algorithm 1:** “brute force” method, finds all possible subsequences by setting 2 points in the sequence, then finding all subsequences between them, keeping track of the ‘max’ subsequence. Theoretical time complexity .

**Algorithm 2:** This algorithm reduces the running time by noticing that the inner for-loop is unnecessary. Thus, the time-complexity is reduced to .

**Algorithm 3:** This recursive algorithm uses a “divide-and-conquer” method, splitting the original sequence into 2 subsequences recursively, then finding the largest sum of a subsequence (ignoring the location of the respective subsequence). Theoretical time complexity .

**Algorithm 4:** This algorithm takes into account the fact that any max subsequence must be ≥ 0. Thus, it makes one pass, *only* recording subsequence sums greater than 0 (this algorithm, again ignores the location of these subsequences; recording only the integer sum). This algorithm is the most efficient of the four algorithms, with .

**Experimental Setup**

The experiment was conducted on a Unix system (through the EECS servers). This experiment was conducted with 11 input sizes (8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096 and 8192). Each input size has 10 unique samples for a total of 110 input samples.

**Experimental Results**

Graph

These plots follow theoretical expectations (as mentioned in the problem statement), the algorithms 1-4 were progressively more efficient (as the graph above whose).