Project 2

CPSC335

Fahad Alsowaylim: fss@csu.fullerton.edu

Giuliana Pham: gpham@csu.fullerton.edu

6/12/2020

Longest Running Subsequence

Pseudocode

```
sequence longest_increasing_powerset(const sequence &A)
{
  const size_t n = A.size();                                        (1)
  sequence best;
  std::vector<size_t> stack(n + 1, 0);                              (2)
  size_t k = 0;                                                     (1)
  while (true)                                                      (2^n times)
  {
        if (stack[k] < n)                                           (1)
        {
              stack[k + 1] = stack[k] + 1;                          (3)
              ++k;                                                  (2)
        }
        else
        {
              stack[k - 1]++;                                       (3)
              k--;                                                  (2)
        }

        if (k == 0)                                                 (1)
              break;

        sequence candidate;
        for (size_t i = 1; i <= k; ++i)                             (k times)
              candidate.push_back(A[stack[i] - 1]);                 (2)

        if (is_increasing(candidate) && best.size() < candidate.size())   (3)
              best = candidate;                                     (1)
  }
  return best;
}
```

Mathematical Analysis

s.c. = 1 + 2 + 1 + 2^n((1+max(5,5)) + 1 + (k*2) + (3 + max(1,0)))

= 4 + 2^n((1 + 5) + 1 + 2k + (3 + 1)) = 4 + 2^n(11 + 2k)

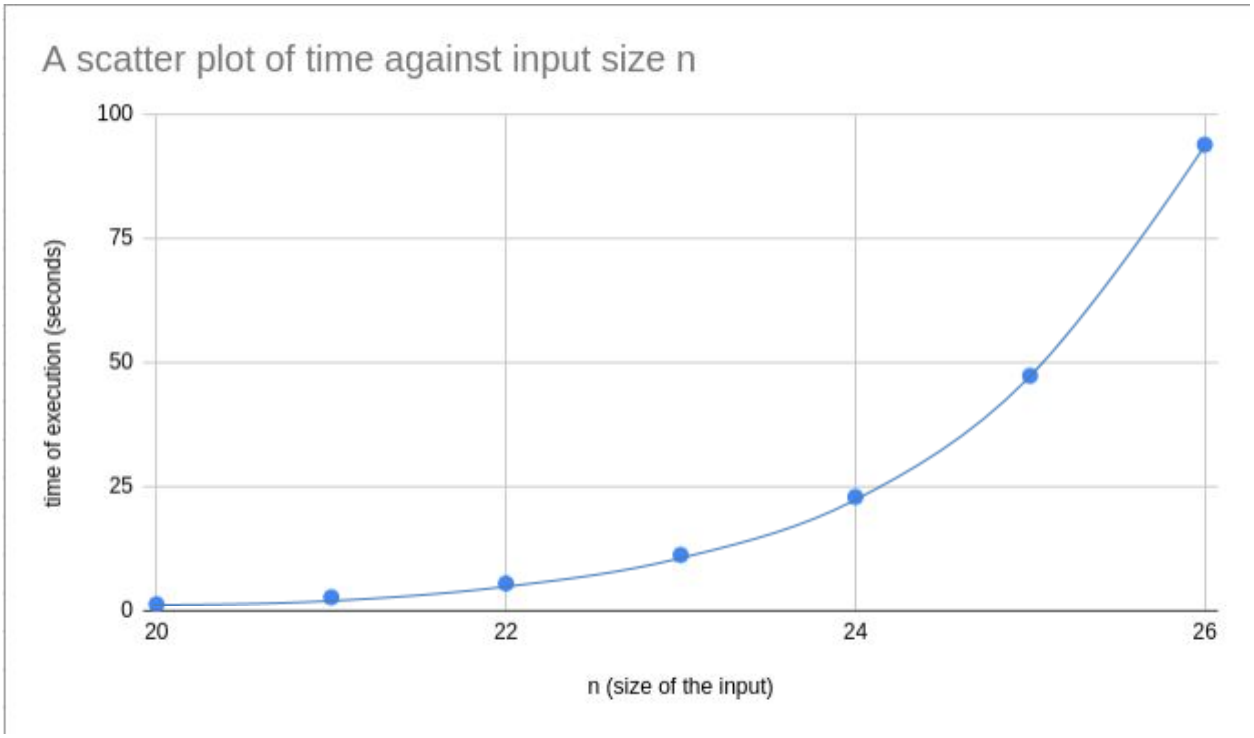
Since k=n in the worst case, O(4 + 2^n(11 + 2k)) = O(4 + 2^n(11 + 2n))

= O(2^n(n)) = O(n*2^n)

So, the Big Oh notation for this particular algorithm is: $O(n * 2^n)$.

Scatterplot

A number of tests were run while giving the algorithm random input of variable length based to see how much time the algorithm took to execute and give credible results. Below is the scatterplot produced as a result of the experimental tests.



Conclusion

The trend of the scatter plot curve is comparable to the shape of the graph of n*2^n, the time complexity calculated in our mathematical analysis. This leads to the conclusion that it is consistent with the empirical analysis. Therefore, the hypothesis that “for large values of n, the mathematically-derived efficiency class of an algorithm accurately predicts the observed running time of an implementation of that algorithm” is affirmed.

Our implemented algorithm is mathematically calculated as exponential time complexity. The scatter plot shows that for a change in input size from n=20 to n=26, the time to execute increases by a factor of 100. An input of 26 is still comparatively small. Therefore, the hypothesis that “algorithms with exponential or factorial running times are extremely slow, probably too slow to be of practical use” is also affirmed by our empirical analysis.