Fahad Alsowaylim Fss@csu.fullerton.edu

Giuliana Pham gpham@csu.fullerton.edu

Project 4 Written Report

Pseudocode

```
sequence longest_increasing_end_to_beginning(const sequence& A) {

        const size_t n = A.size();                                  //1

        std::vector<size_t> H(n, 0);                                //1

        for (signed int i = n-2;  i>= 0; i--) {                     //n-1
                for (size_t j = i+1; j < n ; j++) {                 //(n-1)-(i+1)+1
                        if (A[i] < A[j]) {                          //1
                                if (H[i] <= H[j])                   //1
                                        H[i] = H[j] + 1;            //2
                        }
                }
        }

        auto max = *std::max_element(H.begin(), H.end()) + 1;       //2

        std::vector<int> R(max, 0);                                 //1

       size_t index = max-1, j = 0;                                 //3
       for (size_t i = 0; i < n; ++i) {                             //n
                if (H[i] == index) {                                //1
                        R[j++] = A[i];                              //2
                        Index--;                                    //1
                }
       }
  return sequence(R.begin(), R.begin() + max);                      //2
}
```

Mathematical Analysis

S.C. = 1 + 1 + series_{i=0 to n-2}(series_{i=i+1 to n-1}(1 + max (1 + max(2,0)), 0)) + 2 + 1 + 3
+ n(1 + max(3,0)) + 2
= 2 + series_{i=0 to n-2}(series_{j=i+1 to n-1}(4)) + 6 + n(4) + 2
= 10 + 4n + series_{i=0 to n-2}(series_{j=1 to n-1}(4)-series_{j=1 to i}(4))
= 10 + 4n + series_{i=0 to n-2}(4(n-1)-4i)
= 10 + 4n + series_{i=0 to n-2}(4(n-1)) - series_{i=0 to n-2}(4i)
= 10 + 4n + 4(n-1)(n-2+1) - 4(n-2)(n-1)/2
= 10 + 4n + 4(n-1)^2 - 2(n-2)(n-1)
= 10 + 4n + 4(n^2-2n+1) - 2(n^2-3n+2)
= 10 + 4n + 2n^2 - 2n
= 2n^2 + 2n + 10

O(2n^2 + 2n + 10) = O(n^2)

Scatter Plot Analysis

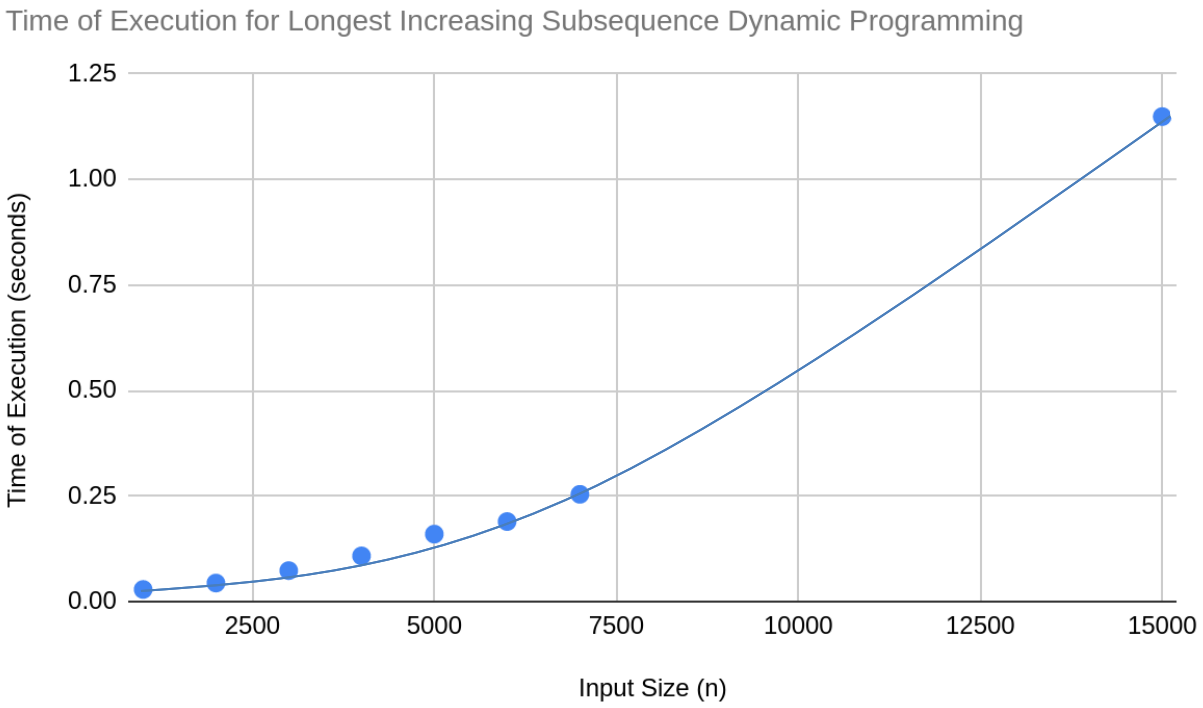
- *Is there a noticeable difference in the running speed of the algorithm as compared to the powerset algorithm in Project 4? Which is faster, and by how much? Does this surprise you?*

Yes, there is a huge difference in this scatter plot and the scatter plot from project 4. From both plots it can be seen that the scatter plot of this project is much better than the one in project 2. If we look over the scatter plot of project 2, then it was taking about 25 seconds for an input of size 24. But in this plot (project 4) it is taking 0.25 seconds for an input of size 6500, which is a huge gap.

From the above analysis it is clear that the one in project 4 is much faster than in project 2. It is exponential time faster than the one in project 2 as it is increasing in polynomial time while the one in project 2 was increasing with exponential time. Yes, it surprised me. I was thinking that the one in project 4 will work faster than the one in project 2, but the extent it is faster is very impressive.

- *Are the fit lines on your scatter plots consistent with this efficiency class? Justify your answer.*

Yes the lines in the scatter plot are consistent with their respective efficiency classes as the efficiency class of the algorithm in project 2 was exponential and it is growing in exponential time. While the efficiency class of the algorithm in project 4 is polynomial, it is growing in polynomial time.



Conclusion

The trend of the scatter plot curve is comparable to the shape of the graph of our mathematically derived efficiency class. This affirms 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.

Project 2 had a time complexity of O(n*2^n), which is exponential time. Project 4 has O(n^2) quadratic time, which is 2 classes of orders of growth smaller than exponential time. An algorithm running at O(100n^2) can process 10^2 problem size in 1 second, compared to O(2^n) processing a 19 problem size in 1 second. Therefore, the dynamic programming implementation is much faster than exhaustive optimization implementation. This affirms the hypothesis that polynomial time dynamic programming algorithms are more efficient than exponential time exhaustive optimization algorithms that solve the same problem.