Time Complexity

```
std::string sequence_to_string(const sequence& seq) {
  std::stringstream ss;                                   O(1)
  ss << "[";                                              O(1)
  bool first = true;                                      O(1)
  for (auto& x : seq) {                                   O(n)
    if (!first) {                                         O(2)
      ss << ", ";                                         O(1)
    }
    ss << x;                                              O(1)
    first = false;                                        O(1)
  }
  ss << "]";                                              O(1)
  return ss.str();                                        O(1)
}
            Time complexity:                              5n+5
```

```
bool is_nonincreasing(const sequence& A) {
  for(std::vector<int>::size_type j = 0; j < A.size()-1; j+=1)        O(n)
  {                                                    O()
              if(A[j]<A[j+1])                                         O(1)
              {
                        return false;                                 O(1)
              }
  }
              return true;                                            O(1)
  }

Time complexity:                                                      2n+1
```

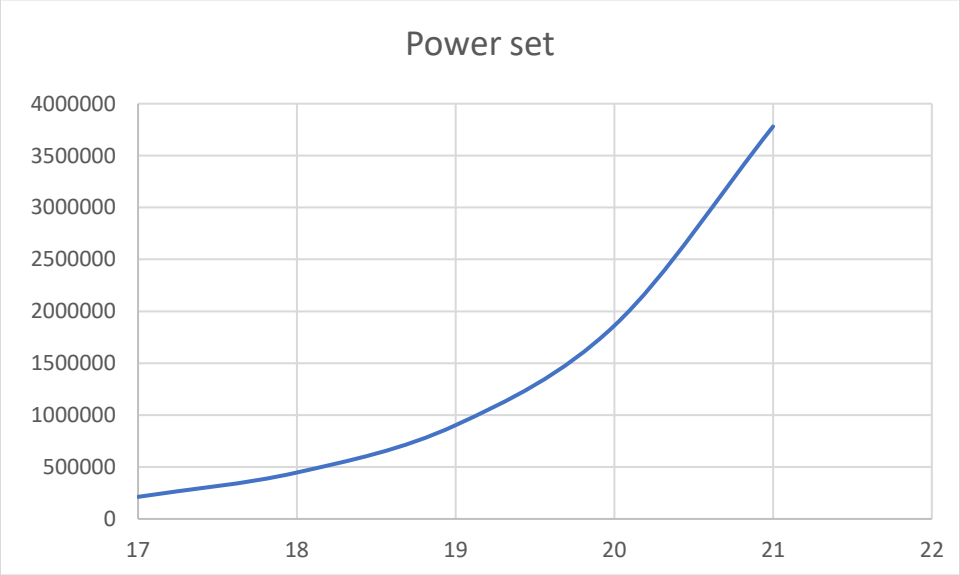
```
sequence longest_nonincreasing_end_to_beginning(const sequence& A) {
  const size_t n = A.size();                                          O(1)
  std::vector<size_t> H(n, 0);                                        O(1)
  for (signed int i = n-2;  i>= 0; i--) {                             O(n)
    for (size_t j = i+1; j < n ; j++) {                               O(n!)
          if(A[i]>=A[j])                                              O(1)
          {
                if(H[i]<=H[j])                                        O(1)
                {
                      H[i]=H[j]+1;                                    O(2)
                }
          }
    }
  }
  auto max = *std::max_element(H.begin(), H.end()) + 1;               O(1)

  std::vector<int> R(max);                                            O(1)
  size_t index = max-1, j = 0;                                        O(2)
  for (size_t i = 0; i < n; ++i) {                                    O(n)
    if (H[i] == index) {                                              O(1)
          R[j]=A[i];                                                  O(1)
          j++;                                                        O(1)
          index--;                                                    O(1)
    }
  }
  return sequence(R.begin(), R.begin() + max);                        O(1)
}
Time complexity:                                                      4*n^2 +5+4n
Time complexity:                                                      n^2
```

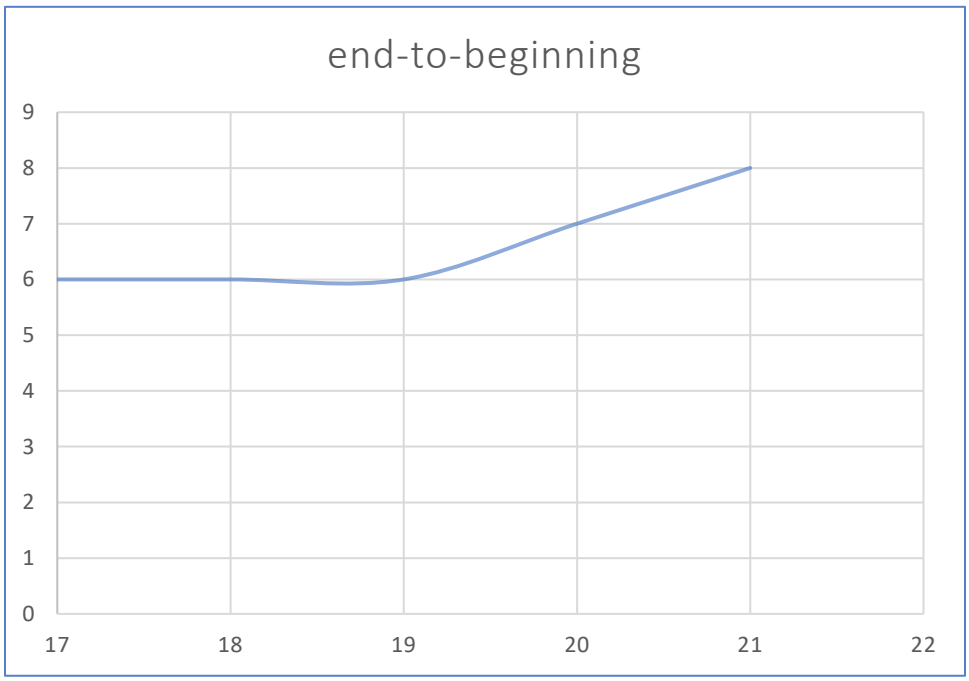
```
sequence longest_nonincreasing_powerset(const sequence& A) {          O()
  const size_t n = A.size();                                            O(1)
  sequence best;                                                        O(1)
        signed int parity=0;                                            O(1)
  std::vector<size_t> stack(n+1, 0);                                    O(1)
  size_t k = 0;                                                         O(1)
  while (true) {                                                        O(2^n)
    if (stack[k] < n) {                                                 O(1)
      stack[k+1] = stack[k] + 1;                                        O(2)
```

```
    ++k;                                                                    O(1)
   } else {
    stack[k-1]++;                                                           O(1)
    k--;                                                                    O(1)
   }
   if (k == 0) {                                                            O(1)
    break;                                                                  O(1)
   }
   sequence candidate;                                                      O(1)
   for (size_t i = 1; i <= k; ++i) {                                        O(n)
    candidate.push_back(A[stack[i]-1]);                                     O(1)
   }
          if(candidate.size()>best.size())                                  O(3)
          {
                  parity=0;                                                 O(1)
                  for (size_t i = 0; i < candidate.size()-1; ++i) {         O(n)
                          if(candidate[i]<candidate[i+1])                    O(1)
                          parity=1;                                         O(1)
                  }
                  if(parity==0){                                            O(1)
                          best=candidate;                                   O(1)
                          }
          }
  }
  return best;                                                               O(1)
}
Time complexity:                                                            5+2^n (6+n+3*2k+2 )
Time complexity:                                                            2^n*n
```

Graph

Count	end to beginning	Power set(ms)
17	6	212701
18	6	447057
19	6	903001
20	7	1.86E+06
21	8	3.78E+06





For count of 50 ,100 and 500 it might take more than minutes

a. Provide pseudocode for your two algorithms.

: End to Beginning

1. Take sequence
2. From end of sequence, start calculating the depth of the number (number of values less than that number (for (signed int i = n-2; i>= 0; i--))
3. Compare the number which having greater index. for (size_t j = i+1; j < n ; j++)
4. Put count in new array

5.once loop completed calculate substring using the new array formed by count in non-increasing order

Power set:

1. Create subset using given array
2. The new subset has been stored in the stack.
3. Once stack is full compare the given stack with the best sub sequence. If the subsequence is greater put the new subset in the best.
4. return the best subsequence.

b. What is the efficiency class of each of your algorithms, according to your own mathematical analysis? (You are not required to include all your math work, just state the classes you derived and proved.)

: The end to beginning is having the best efficiency than powerset algorithm.

End to Beginning: N^2

Power Set : (2^N)*N

c. Is there a noticeable difference in the running speed of the algorithms? Which is faster, and by how much? Does this surprise you?

Yes, End to Beginning is the faster one. as the time complexity increases gradually. But in Power set time complexity increases exponentially.

d. Are the fit lines on your scatter plots consistent with these efficiency classes? Justify your answer.

: Yes, the power set increases time complexity exponentially and for End to beginning the time complexity increases with respect to input

e. Is this evidence consistent or inconsistent with the hypothesis stated on the first page? Justify your answer.

The graph output is consistent for End to beginning as it increases with respect to the input.