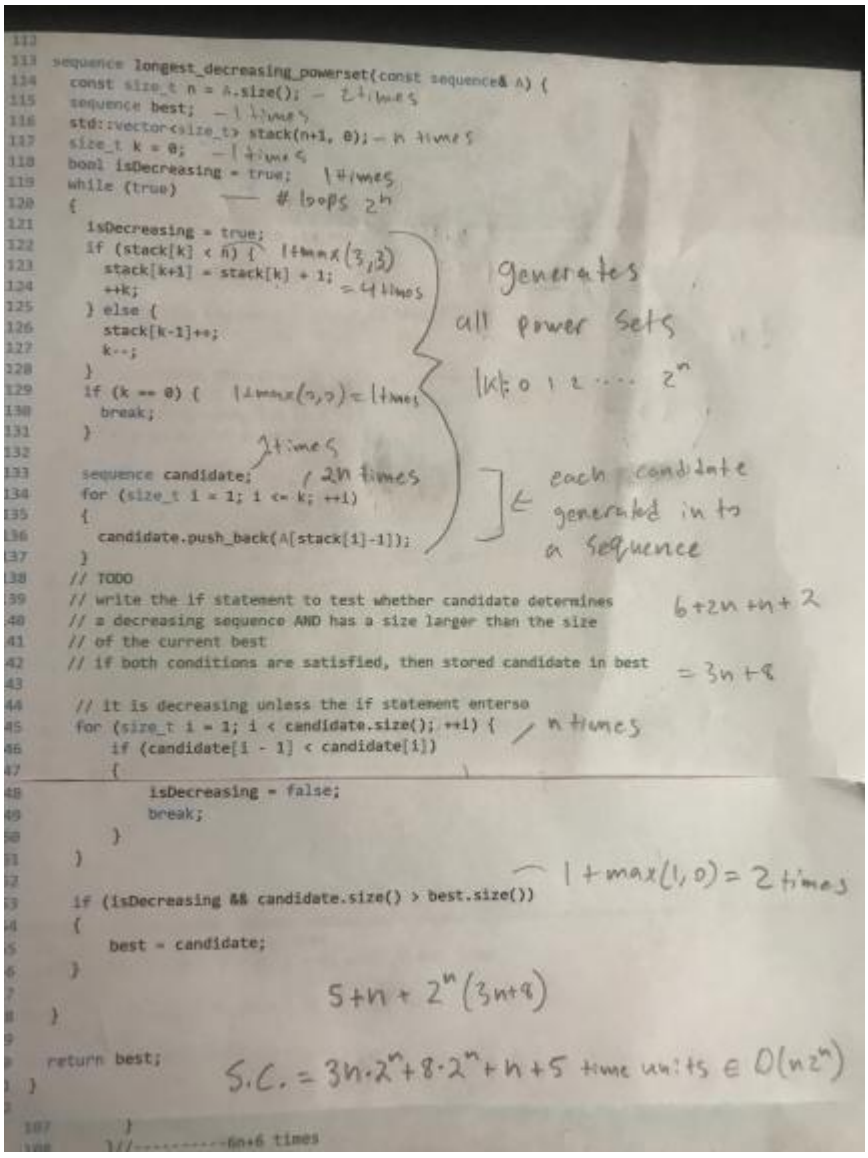


```
sequence longest_decreasing_powerset(const sequence& A) {
  const size_t n = A.size();  — 2 times
  sequence best;  — 1 time
  std::vector<size_t> stack(n+1, 0);  — n times
  size_t k = 0;  — 1 time
  bool isDecreasing = true;  1 time
  while (true)  — # loops 2^n
  {
    isDecreasing = true;
    if (stack[k] < n) {  1+max(3,3)
      stack[k+1] = stack[k] + 1;  = 4 times
      ++k;
    } else {
      stack[k-1]++;
      k--;
    }
    if (k == 0) {  1+max(2,2) = 1+1 time
      break;
    }

    sequence candidate;  1 time  / 2n times
    for (size_t i = 1; i <= k; ++i)
    {
      candidate.push_back(A[stack[i]-1]);
    }
    // TODO
    // write the if statement to test whether candidate determines
    // a decreasing sequence AND has a size larger than the size
    // of the current best
    // if both conditions are satisfied, then stored candidate in best

    // it is decreasing unless the if statement enters
    for (size_t i = 1; i < candidate.size(); ++i) {  / n times
        if (candidate[i - 1] < candidate[i])
        {
            isDecreasing = false;
            break;
        }
    }

    if (isDecreasing && candidate.size() > best.size())  — 1+max(1,0) = 2 times
    {
        best = candidate;
    }

  }

  return best;
}
```

generates all power sets |k|: 0 1 2 .... 2^n

each candidate generated into a sequence

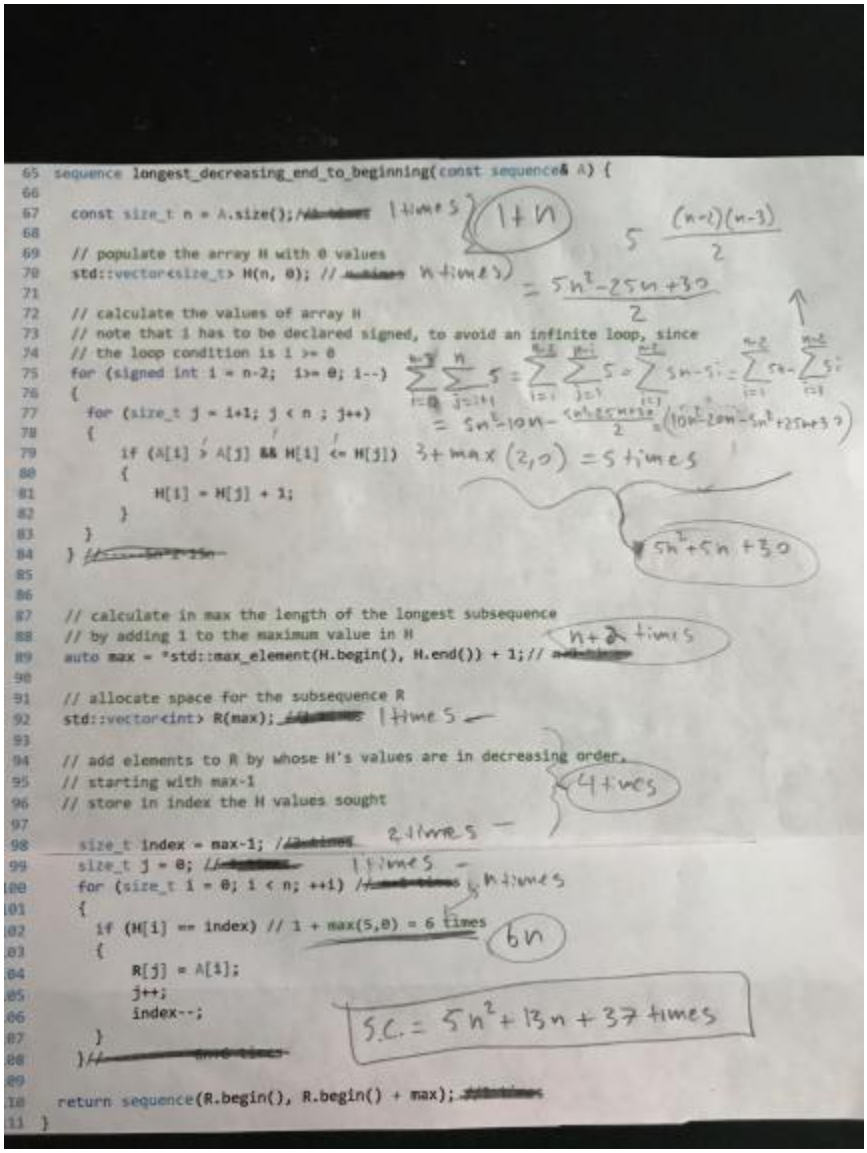
$6+2n+n+2$

$= 3n+8$

$5+n+2^n(3n+8)$

S.C. $= 3n \cdot 2^n + 8 \cdot 2^n + n + 5$ time units $\in O(n2^n)$

```
      }
    }//----------6n+6 times
```



```
65  sequence longest_decreasing_end_to_beginning(const sequence& A) {
66
67    const size_t n = A.size();// 1 times
68
69    // populate the array H with 0 values
70    std::vector<size_t> H(n, 0); // n times
71
72    // calculate the values of array H
73    // note that i has to be declared signed, to avoid an infinite loop, since
74    // the loop condition is i >= 0
75    for (signed int i = n-2;  i>= 0; i--)
76    {
77      for (size_t j = i+1; j < n ; j++)
78      {
79          if (A[i] > A[j] && H[i] <= H[j])
80          {
81              H[i] = H[j] + 1;
82          }
83      }
84    }
85
86
87    // calculate in max the length of the longest subsequence
88    // by adding 1 to the maximum value in H
89    auto max = *std::max_element(H.begin(), H.end()) + 1;// n+2 times
90
91    // allocate space for the subsequence R
92    std::vector<int> R(max); // 1 times
93
94    // add elements to R by whose H's values are in decreasing order,
95    // starting with max-1
96    // store in index the H values sought
97
98    size_t index = max-1; // 2 times
99    size_t j = 0; // 1 times
100   for (size_t i = 0; i < n; ++i) // n times
101   {
102     if (H[i] == index) // 1 + max(5,0) = 6 times
103     {
104         R[j] = A[i];
105         j++;
106         index--;
107     }
108   }
109
110   return sequence(R.begin(), R.begin() + max);
111 }
```

1 + n

5 (n-1)(n-3)/2 = (5n² - 25n + 30)/2

$\sum_{i=0}^{n-2} \sum_{j=i+1}^{n} 5 = \sum_{i=1}^{n-2} \sum_{j=1}^{n-i} 5 = \sum_{i=1}^{n-2} 5n - 5i = \sum_{i=1}^{n-2} 5n - \sum_{i=1}^{n-2} 5i$

$= 5n^2 - 10n - \frac{(n-1)(5n+10)}{2}$ (10n² - 20n - 5n² + 25n - 30)

3 + max(2,0) = 5 times

5n² + 5n + 30

n + 2 times

1 time

4 times

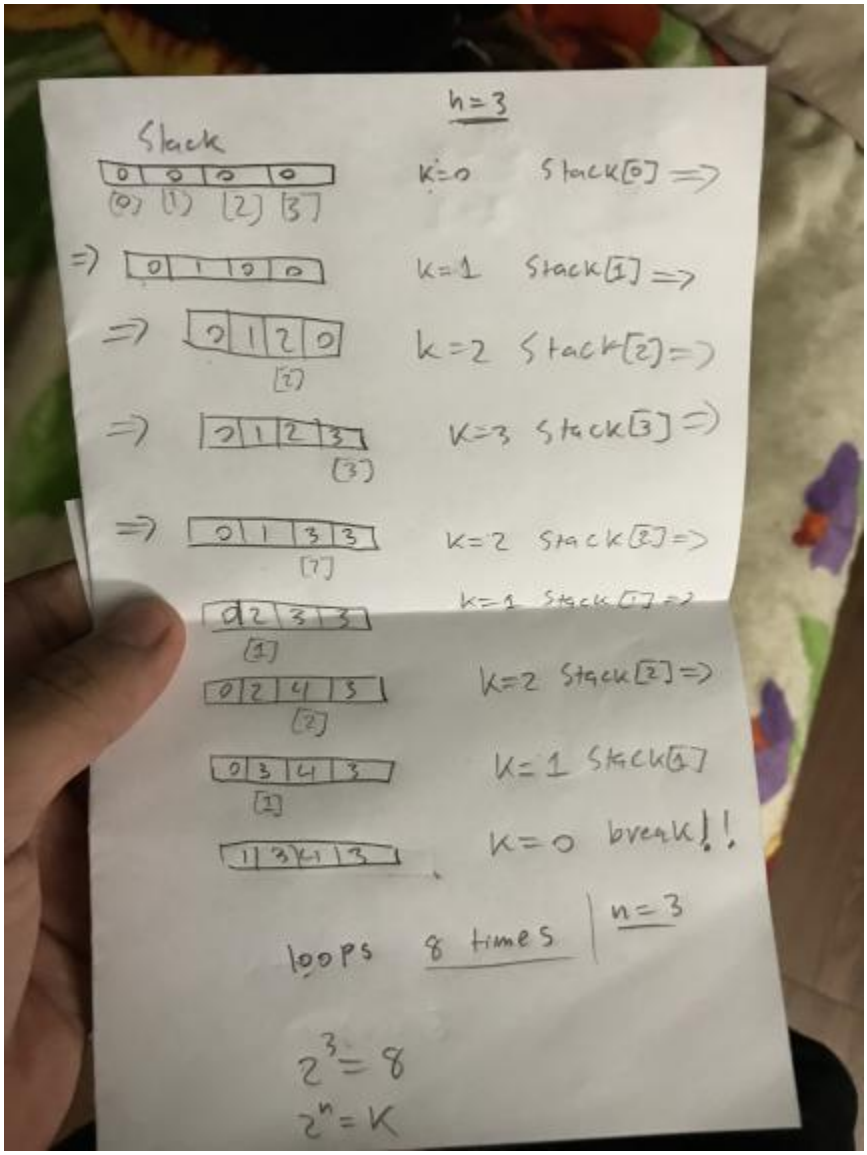
2 times

1 time

n times

6n

S.C. = 5n² + 13n + 37 times


n=3
Stack
k=0 stack[0] =>
k=1 stack[1] =>
k=2 stack[2] =>
k=3 stack[3] =>
k=2 stack[2] =>
k=2 stack[2] =>
k=1 stack[1]
k=0 break!!
loops 8 times
n=3
$2^3 = 8$
$2^n = K$