Introducing Algorithm Performance
The "Maximum Subsequence Sum Problem"

KV for CSE 330, Jan 2020

Based on Weiss, DSAAC++, Chapter 2

These notes do not strictly follow the order of presentation in the textbook …

Three algorithms are given and implemented. Here are runs for Algorithm1, Algorithm2, and Algorithm4. We will focus on Alg1 and Alg4 …

```
Vector size: 10


vector: 21 -45 12 -45 -6 29 42 25 -9 -23

max_sub_seq_alg1: 96(220 additions)
max_sub_seq_alg2: 96(55 additions)
max_sub_seq_alg4:  96(10 additions)

Another run? [y/n] y

y    1

Vector size: 25


vector: 21 -45 12 -45 -6 29 42 25 -9 -23 1 35 4 -30 12 1 -39 13 -17 24 47 -18 -2
8 23 27 32 9 25 38 22 -15 -47 -43 4 39

max_sub_seq_alg1: 245(7770 additions)
max_sub_seq_alg2: 245(630 additions)
max_sub_seq_alg4:  245(35 additions)

Another run? [y/n] y

y    1

Vector size: 100


vector: 21 -45 12 -45 -6 29 42 25 -9 -23 1 35 4 -30 12 1 -39 13 -17 24 47 -18 -2
8 23 27 32 9 25 38 22 -15 -47 -43 4 39 50 22 45 24 42 -36 1 -11 20 -34 47 47 6 3
0 8 49 31 15 42 -43 50 5 -34 7 15 6 -39 1 49 -7 43 4 38 24 41 20 -9 -45 -16 16 2
6 49 42 7 -13 -20 45 41 27 29 6 -35 33 5 -32 36 13 7 40 40 22 39 -32 -16 2 15 33
 37 35 26 -33 5 24 3 27 29 -27 44 36 44 -15 -23 18 19 -23 -23 34 30 1 34 33 -44
32 -50 7 15 2 36 -7 14

max_sub_seq_alg1: 1456(419220 additions)
max_sub_seq_alg2: 1456(9180 additions)
max_sub_seq_alg4:  1456(135 additions)

Another run? [y/n] n

n    0
Press any key to continue . . .
```

	#Adds	#Adds	#Adds
Vector Size	Alg1	Alg2	Alg3
10	220	55	10
25	7770	630	35
100	419220	9180	135

```
// adopted from Weiss, DSAAC++, p. 61, Algorithm1
int max_subseq_sum_alg1(const vector<int>& vec, int& ops)
{
     int maxSum = 0;

     for (int i = 0; i < vec.size(); i++)
          for(int j = i; j < vec.size(); j++)
          {
               int thisSum = 0;

               for (int k = i; k <= j; k++)
               {
                    thisSum += vec[k];
               }

               if (thisSum > maxSum)
                    maxSum = thisSum;
          }
     return maxSum;
}



// adopted from Weiss, DSAAC++, p. 62, Algorithm2
int max_subseq_sum_alg2(const vector<int>& vec, int& ops)
{
     // see book
}
```

We are skipping Algorithm3 …

```
// adopted from Weiss, DSAAC++, p. 66, Algorithm4
int max_subseq_sum_alg4(const vector<int>& vec, int& ops)
{
     int maxSum = 0;
     int thisSum = 0;
     ops = 0;

     for (int i = 0; i < vec.size(); i++)
     {
          thisSum += vec[i];
          ops += 1;

          if (thisSum > maxSum)
               maxSum = thisSum;
          else if (thisSum < 0)
               thisSum = 0;
     }
     return maxSum;  // WHY DOES THIS WORK????
}
```

All three algorithms solve the problem, i.e.., produce the correct solution.

The algorithms solve the problem at different cost:

Cost as:

- **# of operations (e.g., additions)**
- **Run time (proportional to # ops)**
- **memory space**

Algorithm performance is measured not in terms of exact numbers, but in terms of the order of magnitude of the performance trend when problems get biggger.

Size of the problem: **vector of size 10 – small problem**

vector of size 100 – larger problem

vector of size 100K – very large problem

etc.

For the order of magnitude performances of Algorithms1,2,4, we write:

Algorithm1: cost of run time is $O(n^3)$, n being the size of problem ("cubic")

With increasingly larger problems (e.g., ones involving an increasingly large data set), the run time of the algorithm follows a cubic trend relative to problem size n.

Algorithm2: cost is $O(n^2)$ ("quadratic")

With increasingly larger problems (size = n), the run time of the algorithm grows following a quadratic trend.

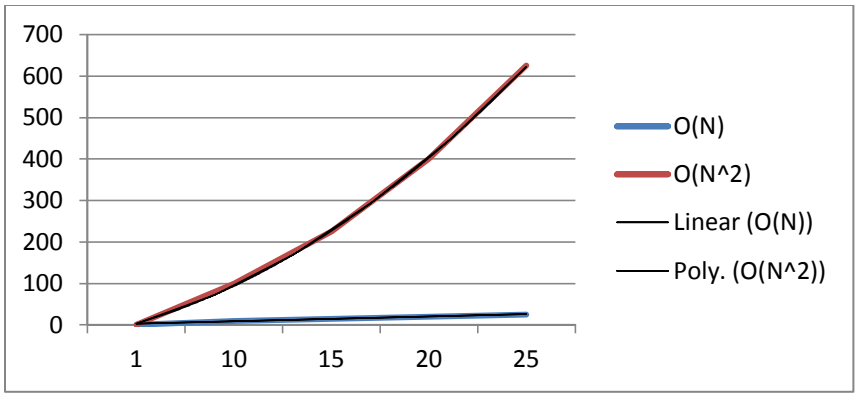
Algorithm4: cost is O(n) ("linear")

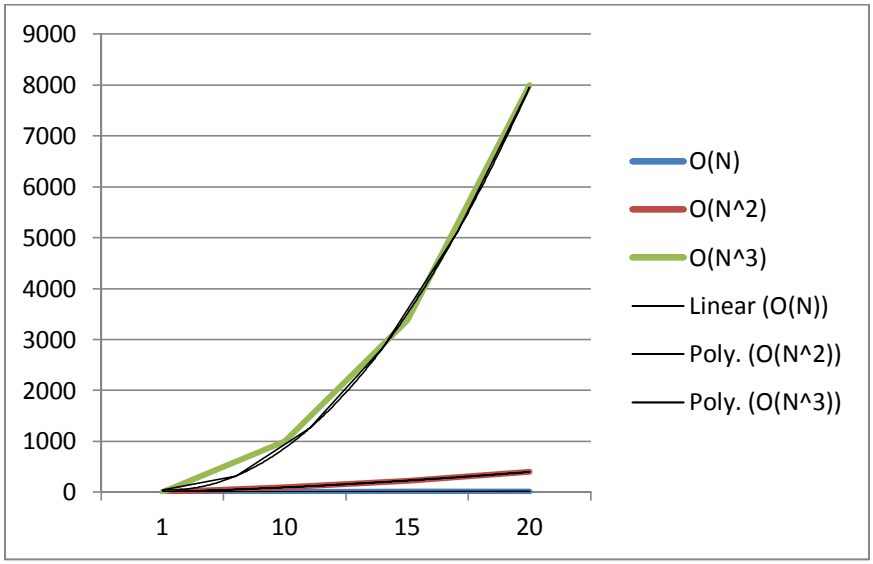
With increasingly larger problems (size = n), the run time of the algorithm grows following a linear trend.

SMALLER is BETTER ("more efficient"), and "linear" is a very desirable trend.

Is there better than LINEAR? – YES
Is there worse than CUBIC? – YES

Graphically





Formally:

(a preview of CSE 431 Algorithm Analysis)

Let the performance of your program follow some function T(N), where n is the size of the problem.

Then $T(N) = O(N^2)$ means that there are positive constants c and k such that $0 \leq T(N) \leq cN^2$ for all $n \geq k$.

General:

$T(N) = O(g(N))$ means that there are positive constants c and k such that $0 \leq T(N) \leq cg(N)$ for all $n \geq k$.