Program efficiency

IC152 Feb 2021

- During design, need to choose data structures and algorithms
- Implement in various languages
  - Python, C, Perl, …
- Run on variety of hardware+OS
  - 1 GHz, 2 GHz, 2.8 GHz, ...
  - Linux, Windows, MacOS X, Raspberry Pi, ...
- Range of data sizes
  - Students = 10, 100, 1,300, 10,000, 1,00,000, 2,00,00,000, ...

How to decide which is “best”?

Slides from IC152 2018

- Decide on “basic” operations
  - comparison, arithmetic, copy, etc of scalar variables
  - count each as 1 unit
- Eg 1: Find average of N numbers

```
sum = 0                                    Work
for I in 1 to N do                         1
     sum = sum + A[I]                      2*N
avg = sum / N                              1*N
                                           1
```

Total Work = 2+3N

Simplify problem = Abstraction

- Eg 2: Find average of numbers read so far

- ```
  def GetAvg(A, K):    # avg of A[1..K]
     sum = 0                                 1
     for I in 1 to K do                      2*K
          sum = sum + A[I]                   1*K
     return sum / K                          1
  ```

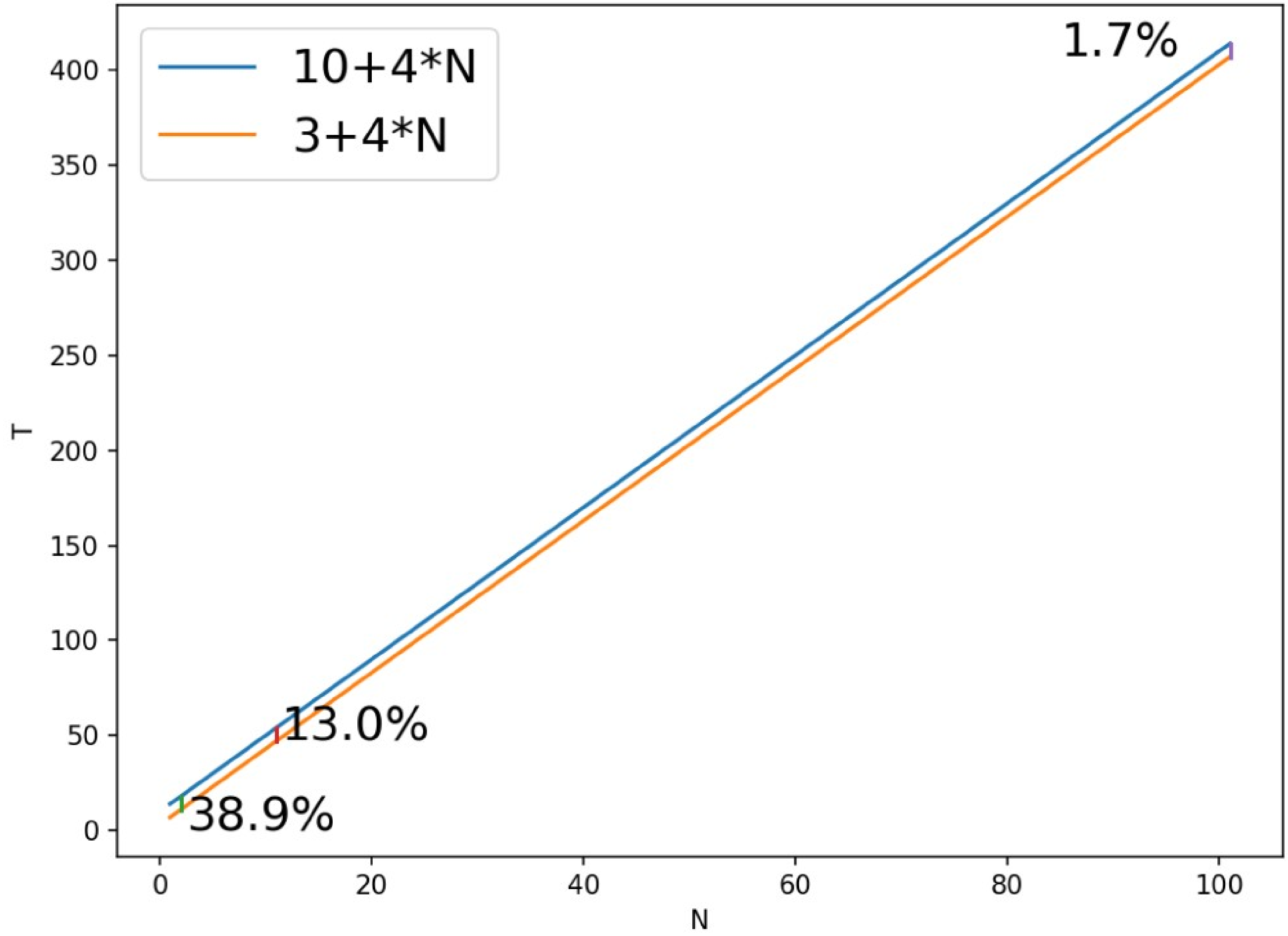
  Total Work = 2+3K

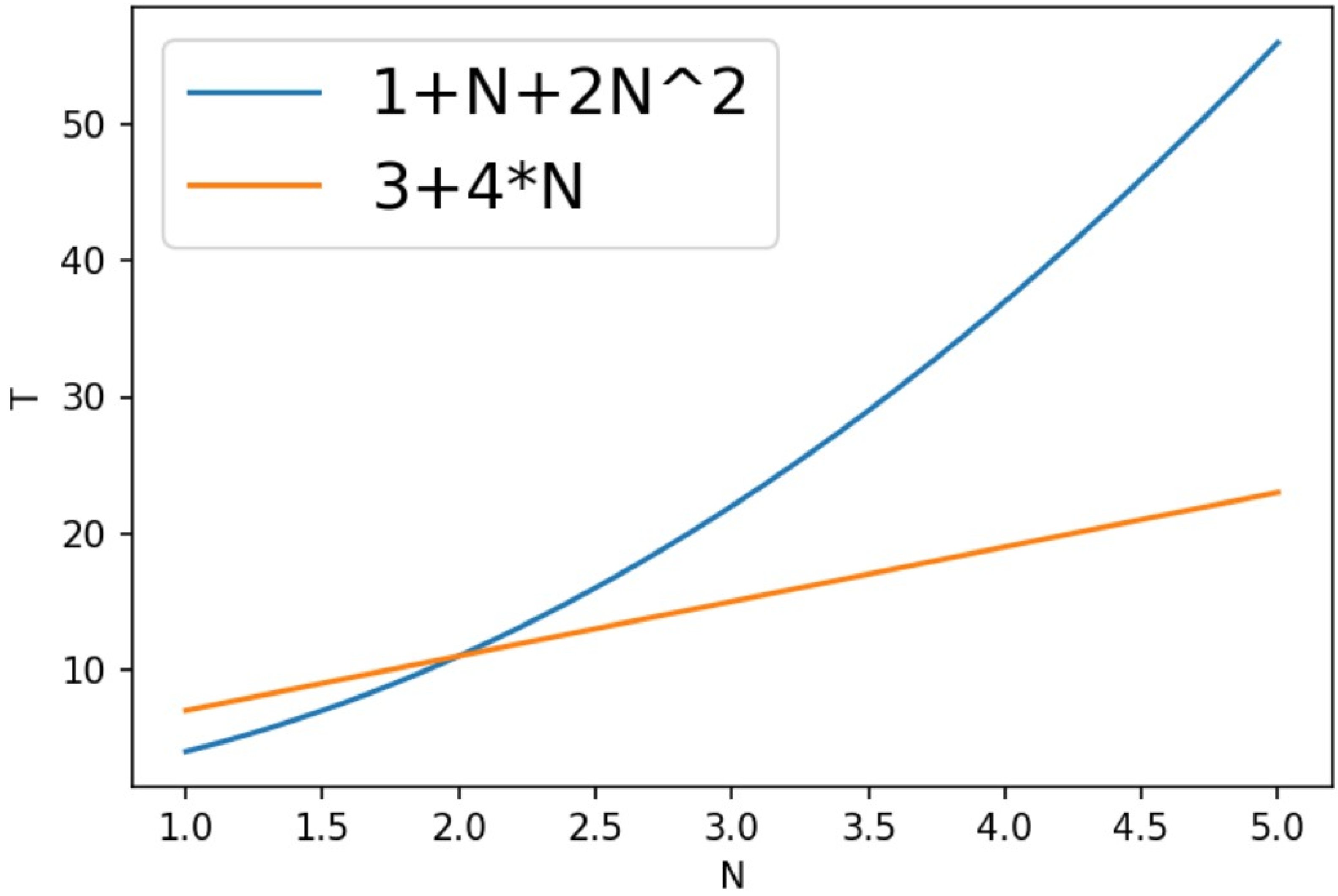
  ```
  for I in 1 to N do
       avg[I] = GetAvg(A, I)
  ```

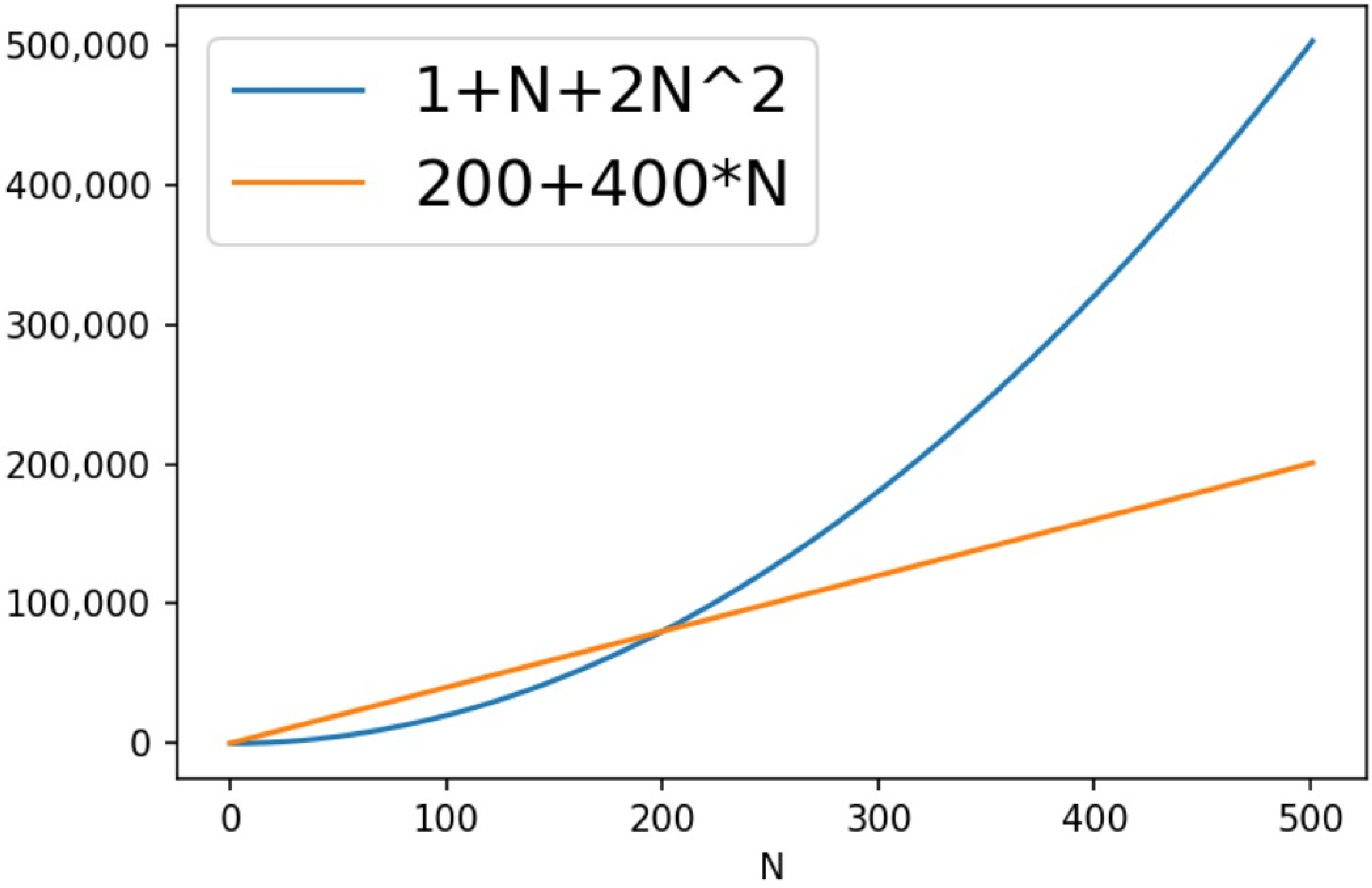
  Total Work = (4+3)+(4+3*2)+(4+3*3)+...+(4+3*N)

  $= 5.5N + 1.5N^2$
  ```

- $T1(N) = 3 + 4N$
- $T2(N) = 10 + 4N$
  - Small N, T2 > T1, but for large N, both ~ same
- $T3(N) = 1 + N + 2N^2$
- $T4(N) = 10 + 20N + 2N^2$
  - Small N: T4 > R3, large N: both ~ same
- $T2(N) = 10 + 4N$
- $T3(N) = 1 + N + 2N^2$
  - Small N: T2 > T3, but for large N, T3 > T2
- If $T2a(N) = 200 + 400N$
  - Very large N, T3 >T2a


10+4*N
3+4*N
1.7%
13.0%
38.9%
400
350
300
250
200
150
100
50
0
T
0
20
40
60
80
100
N


1+N+2N^2
3+4*N
T
N
10
20
30
40
50
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


1+N+2N^2
200+400*N
500,000
400,000
300,000
200,000
100,000
0
0
100
200
300
400
500
N

- In general, if Tx = a + bN

$$Ty = c + dN + eN^2$$

a, b, c, d, e independent of N

  - Some N0 such that Ty > tx for N ≥ N0
- Complexity of performance:
  - Order the terms in the expression by power of N
  - Set all constants to 1
  - Consider only the highest power
  - This is the asymptotic complexity = O() "Big-O"
  - If Tx is O(N) and Ty is $O(N^2)$
    Tx is faster than Ty

- Comparing designs is very difficult because of differences in implementation, execution platform, input data

==> Asymptotic complexity O()

- $O(N) < O(N^2) < O(N^3) < O(2^N)$
  - Applied to execution time & memory space
- Useful for initial design
- During implementation, testing and production use
  - measurement of actual execution time, memory usage, etc.
  - tuning to improve performance

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

From Klienberg, Algorithm Analysis, 2nd ed.

Linear Time: O(n)

Linear time. Running time is proportional to input size.

Computing the maximum. Compute maximum of n numbers $a_1, \ldots, a_n$.

```
max ← a_1
for i = 2 to n {
   if (a_i > max)
      max ← a_i
}
```

Closest pair of points. Given a list of n points in the plane (x_1, y_1), ..., (x_n, y_n), find the pair that is closest.

$O(n^2)$ solution. Try all pairs of points.

```
min ← (x1 - x2)^2 + (y1 - y2)^2
for i = 1 to n {
   for j = i+1 to n {
      d ← (xi - xj)^2 + (yi - yj)^2      ← don't need to take square roots
      if (d < min)
         min ← d
   }
}
```

Set disjointness. Given n sets $S_1, \ldots, S_n$ each of which is a subset of 1, 2, ..., n, is there some pair of these which are disjoint?

$O(n^3)$ solution. For each pairs of sets, determine if they are disjoint.

```
foreach set Si {
   foreach other set Sj {
      foreach element p of Si {
         determine whether p also belongs to Sj
      }
      if (no element of Si belongs to Sj)
         report that Si and Sj are disjoint
   }
}
```