CS202 – Data Structures

LECTURE-02

Analysis Tools

Experimental and Asymptotic Analysis

Dr. Maryam Abdul Ghafoor
Assistant Professor
Department of Computer Science, SBASSE

Register with “Poll Everywhere”

- Please register yourself using your LUMS email address: https://pollev.com/blesseddawn004/register

- To participate in activities, please open the following link on your browser and keep it open during class:
  - https://pollev.com/blesseddawn004/

Agenda

- Data Structures VS Algorithms
- Experimental Analysis
- Asymptotic Analysis

Data Structures versus Algorithm

- Data Structure focuses on organization of data
  - Containers that hold actual data
  - Characterization: linear vs non-linear, static vs. dynamic, homogeneous vs. heterogeneous

- Algorithm focuses on design and analysis of algorithms
  - step-by-step procedure to manipulate data
  - Operations

Data Structures versus Algorithm

Variable

Placeholder for one data item

Memory address

Data type

….

Operations

Store a data item

Search for a data item

…

Array

Placeholder for multiple data items

Memory address of first item

Data type

Contiguous memory allocation

….

Operations

add a data item

Search for a data item

…

Data Structures versus Algorithm

- Data Structures: To learn how different data structures organize and store data efficiently, and how they impact the performance of algorithms

- Algorithms: To learn how to design efficient algorithms to solve various problems and to analyze their computational complexity

Algorithm Efficiency

Poll

What factors influence the efficiency or performance of an algorithm?

Algorithm Efficiency

- What factors affect the efficiency of an algorithm?

- Some resources may matter more than others based on context
  - Speed of machine
  - Amount of memory
  - Network bandwidth (e.g., video streaming services)
  - Energy consumption(e.g., IoT devices)

Which Performs Better? (n = 1 Billion)

Activity

Computer A
Speed: 1Trillion/sec

Number of instructions: $6n^2$

Time?

Computer B
Speed: 10 Million/sec

Number of instructions: 50n(log n)

Time?

Which Performs Better? (n = 1 Billion)

Computer A

Speed: 1Trillion (10^{12})/sec

Number of instructions: $6n^2$

Time?

6000000 ~ 69 days

Computer B

Speed: 10 Million (10^7)/sec

Number of instructions: $50n(\log n)$

Time?

45000 ~ 12.5 hrs

Which Algorithm is Efficient?

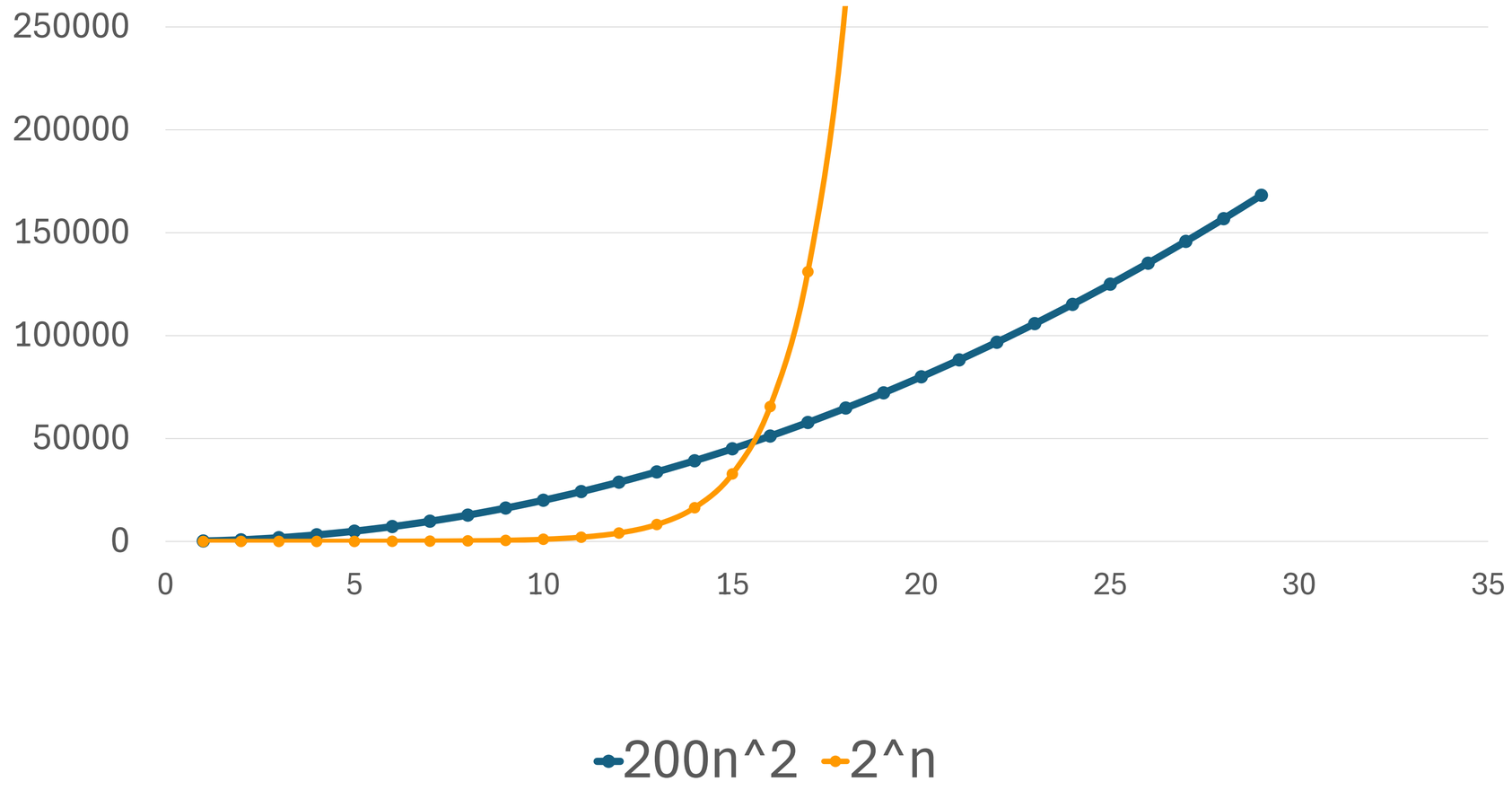
Activity

Approach A: $200n^2$

Approach B: 2^n

What is the smallest value of n such that an algorithm whose running time is $200n^2$ runs faster than an algorithm whose running time is 2^n on the same machine?

What is the smallest value of n such that an algorithm whose running time is 200n^2 runs faster than an algorithm whose running time is 2^n on the same machine?



n	200n^2	2^n
1	200	2
2	800	4
3	1800	8
4	3200	16
5	5000	32
6	7200	64
7	9800	128
8	12800	256
9	16200	512
10	20000	1024
11	24200	2048
12	28800	4096
13	33800	8192
14	39200	16384
15	45000	32768
16	51200	65536
17	57800	131072
18	64800	262144
19	72200	524288
20	80000	1048576
21	88200	2097152
22	96800	4194304
23	105800	8388608
24	115200	16777216
25	125000	33554432
26	135200	67108864
27	145800	134217728
28	156800	268435456
29	168200	536870912
30	180000	1073741824

Algorithm Efficiency

- What factors affect the efficiency of an algorithm?

- Some resources may matter more than others based on context
  - Speed of machine
  - Amount of memory
  - Network bandwidth (e.g., video streaming services)
  - Energy consumption(e.g., IoT devices)
  - Input size

Measuring Performance

- Experimental measurement involves timing actual runs.
  - Real, concrete value

- Theoretical measurement involves analyzing usage abstractly, without running the code.

Agenda

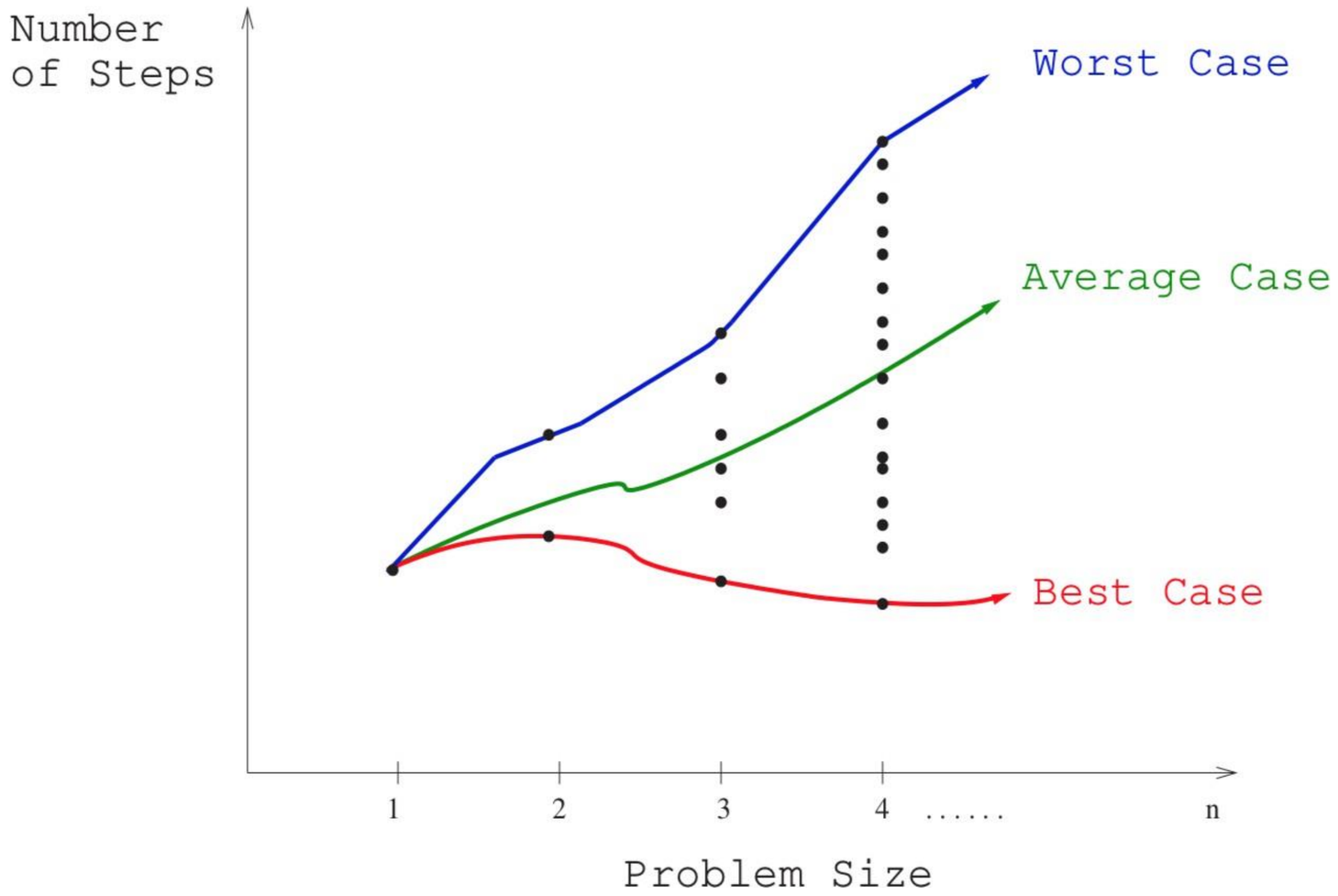
- Data Structures vs Algorithms
- Experimental Analysis
- Asymptotic Analysis

Running Time

- Best Case: The input that uses the least resources (fastest runtime)
- Worst Case: The input that uses the most resources (slowest runtime)
- Average Case: Typical performance across random inputs (closer to perceived performance)

  When designing algorithms, you aim to improve the worst-case complexity.

Best, Worst and Average Cases



Best, Worst and Average Cases

Activity

- Given the array [8,2,4,6,10], how many steps will linear search take to find a number in the best, worst, and average case?

Experimental Analysis – Comparing Algorithms

- Implement the two algorithms
- Use a timing utility to measure time
- Ensure that all factors external to the algorithm are kept constant (e.g., number of background apps, programming language used)
- For each input size and input type, run each algorithm multiple times
- Calculate the average running time and the standard deviation (or the confidence interval) to find if the difference is statistically significant

Experimental Analysis

- Timing utilities
  - “time” utility in Linux/Mac

```
bash-3.2$ time open political.csv

real    0m0.420s
```

  - %time in python

```
1 %time sum(range(1000000))
```

```
CPU times: user 19.4 ms, sys: 112 µs, total: 19.6 ms
```

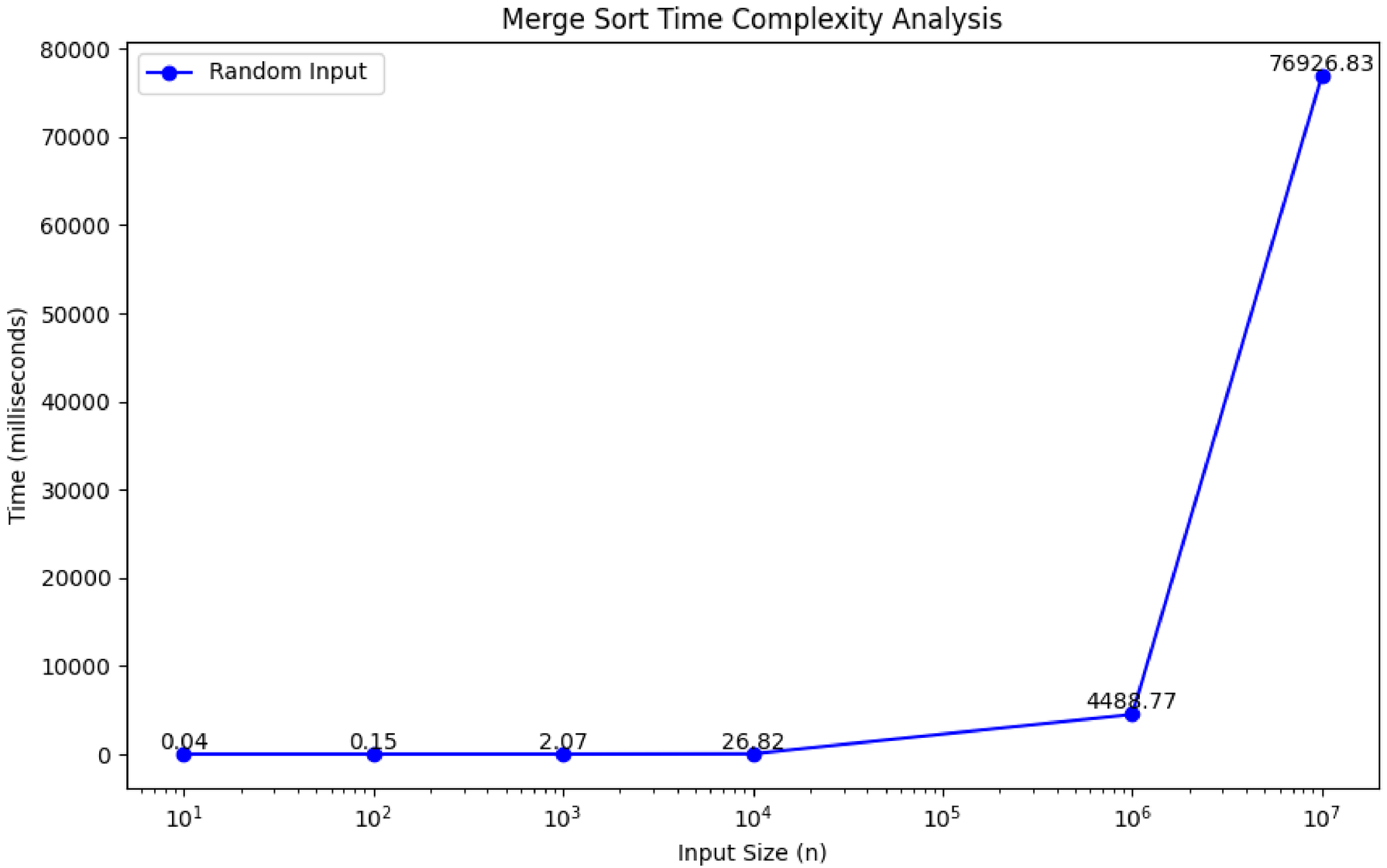
  - chrono in C++

```
std::chrono::high_resolution_clock::now();
```

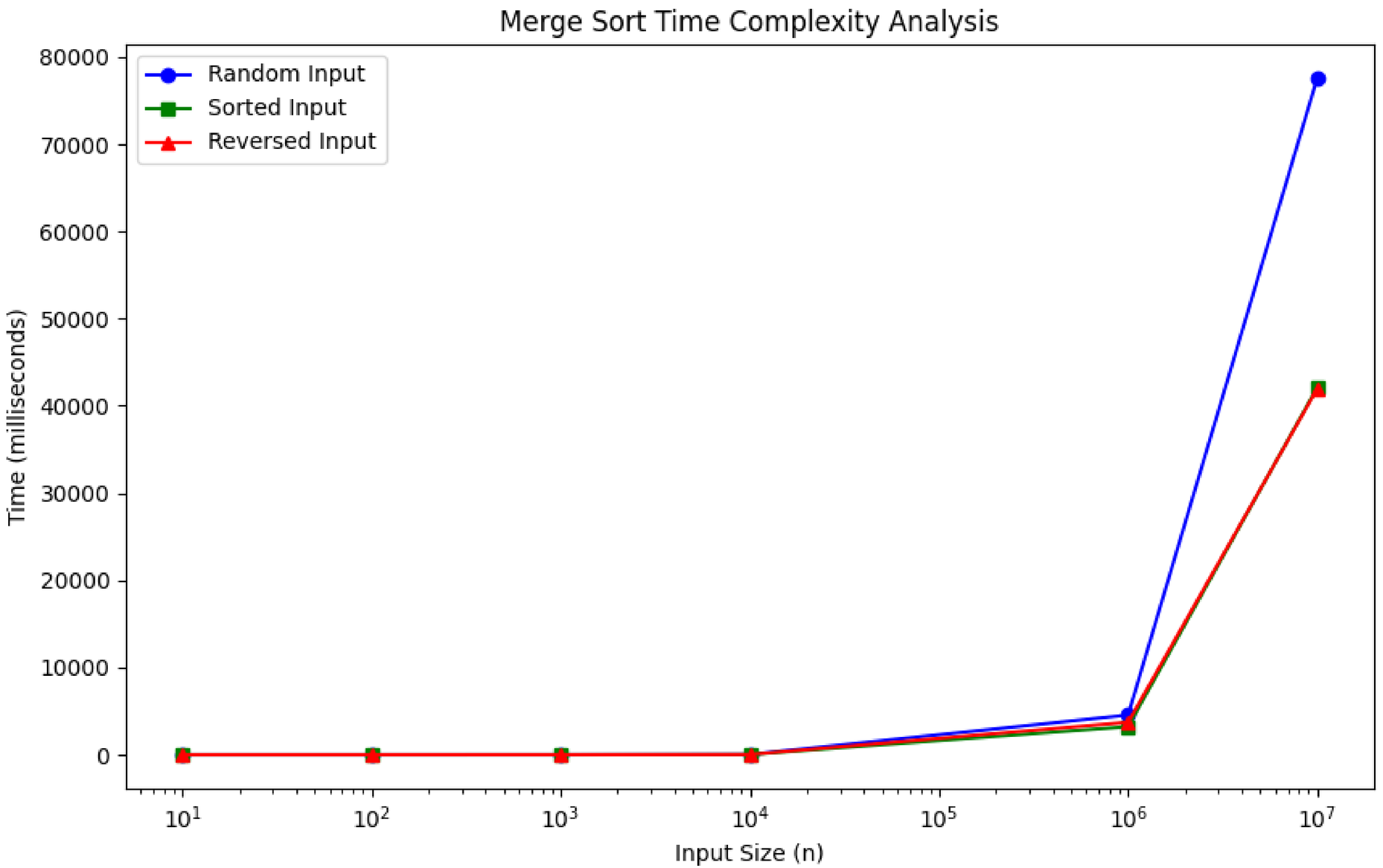
Analysis of a Sorting Algorithm

Scenario	Input Type	Input size	Results
Impact of input size	Random	10, 100, 1000, 10000, 1000000, 10000000	
Impact of input Type	Random, Sorted, Reversed	10, 100, 1000, 10000, 1000000, 10000000	
Fixed input (multiple runs)	Random	100000	

Effect of Input Size



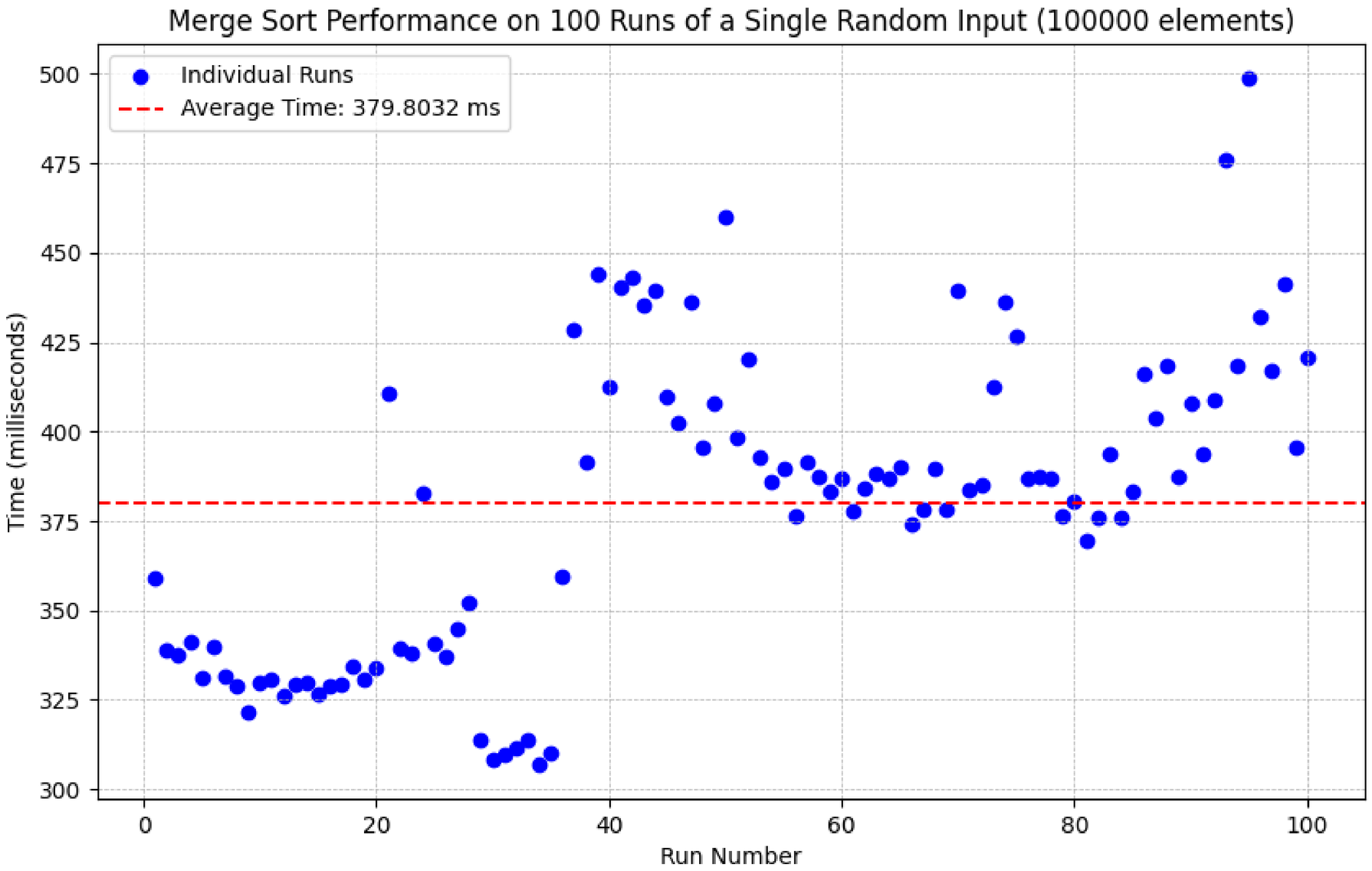
Effect of Input Type



Analysis of a Sorting Algorithm

Scenario	Input Type	Input size	Results
Impact of input size	Random	10, 100, 1000, 10000, 1000000, 10000000	Running time varies with input size
Impact of input Type	Random, Sorted, Reversed	10, 100, 1000, 10000, 1000000, 10000000	Running time varies with input type
Fixed input (multiple runs)	Random	100000	

Performance on Multiple Runs



Analysis of a Sorting Algorithm

Scenario	Input Type	Input size	Results
Impact of input size	Random	10, 100, 1000, 10000, 1000000, 10000000	Running time varies with input size
Impact of input Type	Random, Sorted, Reversed	10, 100, 1000, 10000, 1000000, 10000000	Running time varies with input type
Fixed input (multiple runs)	Random	100000	Running time varies across runs even with the same input

Limitations of Experimental Analysis

- Scalability issues
  - Testing on very large inputs may be impractical
- Resource intensive
  - Requires hardware, cloud, or energy costs
- Implementation overhead
  - Algorithms must be coded first, which is time-consuming (especially for complex ones)
- Uncontrollable factors
  - System load, compiler optimizations, and hardware differences affect results
- Limited generalization
  - Results may not reflect performance on all machines or future technologies

Can we come with a "general" way?

- A general way of analyzing the running time of algorithms that:
  - Takes into account all possible inputs
  - Is independent of hardware and software configuration
  - Does not require implementing the algorithms

Agenda

- Abstract Data Types VS Data Structures
- Experimental Analysis
- Asymptotic Analysis

Asymptotic Analysis

- Asymptotic: Approaching an extreme value like infinity
  - When input size becomes extremely large, we use asymptotic analysis to evaluate efficiency

- Asymptotic behavior: How the function/data behaves under extreme conditions

RAM Model of Computation

- Algorithm design is considered machine-independent by relying on a hypothetical model called the Random Access Machine (RAM).
  - Each primitive operation takes 1 step
    - `e.g., +, =, /, *, (), comparison, a[], return, etc.`
  - Accessing memory takes exactly 1 step.

- The running time of an algorithm is measured by counting the number of steps it performs

Practice: Counting Primitive Operations

Algorithms	Assignments	Additions	Comparisons	Total

sum = 0;
for(i=0; i < n; i++)
  sum = sum + n;				

Is the RAM Model useful?

- Is it too simple to be useful?
  - Multiplication ≠ Addition (1-step assumption fails)
  - Compiler optimizations (e.g., loop unrolling) break assumptions
  - Memory access times vary

- Yet… RAM works well in practice
  - Balances realism and simplicity

The Number of Primitive Operations?

Poll

	Statements	Steps
1	`float Avg(vector<int>& grades, int size){ //size=n`	
2	`  float sum = 0;`	
3	`  int count = 0;`	
4	`  while ( count < size ) {`	
5	`    sum += grades[count];`	
6	`    count += 1;`	
7	`  }`	
8	`  if ( size > 0 )`	
9	`    return sum / size;`	
10	`  else`	
11	`    return 0.0f;`	
12	`}`	
Total		

The Number of Primitive Operations?

	Statements	Steps
1	`float Avg(vector<int>& grades, int size){ //size=n`	
2	`  float sum = 0;`	1
3	`  int count = 0;`	1
4	`  while ( count < size ) {`	n + 1
5	`    sum += grades[count];`	3n
6	`    count += 1;`	2n
7	`  }`	
8	`  if ( size > 0 )`	1
9	`    return sum / size;`	2
10	`  else`	
11	`    return 0.0f;`	1
12	`}`	
Total		6n + 6 6n + 5

The Number of Primitive Operations?

	Statements	Steps
1	`float Avg(vector<int>& grades, int size){ //size=n`	
2	`  float sum = 0;`	1
3	`  int count = 0;`	1
4	`  while ( count < size ) {`	n + 1
5	`    sum += grades[count];`	3n
6	`    count += 1;`	2n
7	`  }`	
8	`  if ( size > 0 )`	1
9	`    return sum / size;`	2
10	`  else`	
11	`    return 0.0f;`	1
12	`}`	
Total		6n + 6 6n + 5

What is the count of primitive operations if size = 0?

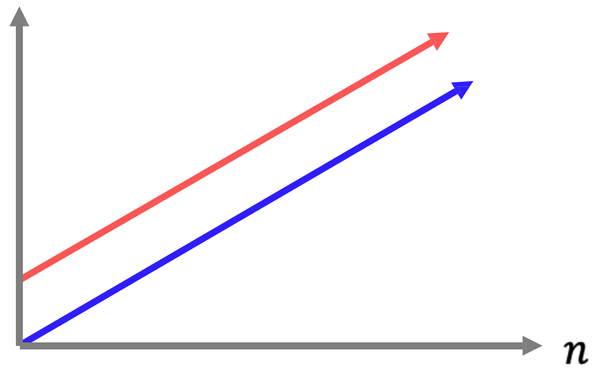
Expressing Running Time

- To analyze time or space complexity mathematically, we express it as a function:
  - T(n): time needed when input size is n
  - S(n): space needed similarly

Expressing Running Time

- Do we really care about the constant 6 if T(n)=6n+6?
  - Well, for large input sizes, the constant terms will be dominated by higher order terms so they would not matter much
  - Ex: T1(n)=n+7 vs T2(n)=n+4

T_1 and T_2 are both linear and for large input sizes, constants do not matter much.

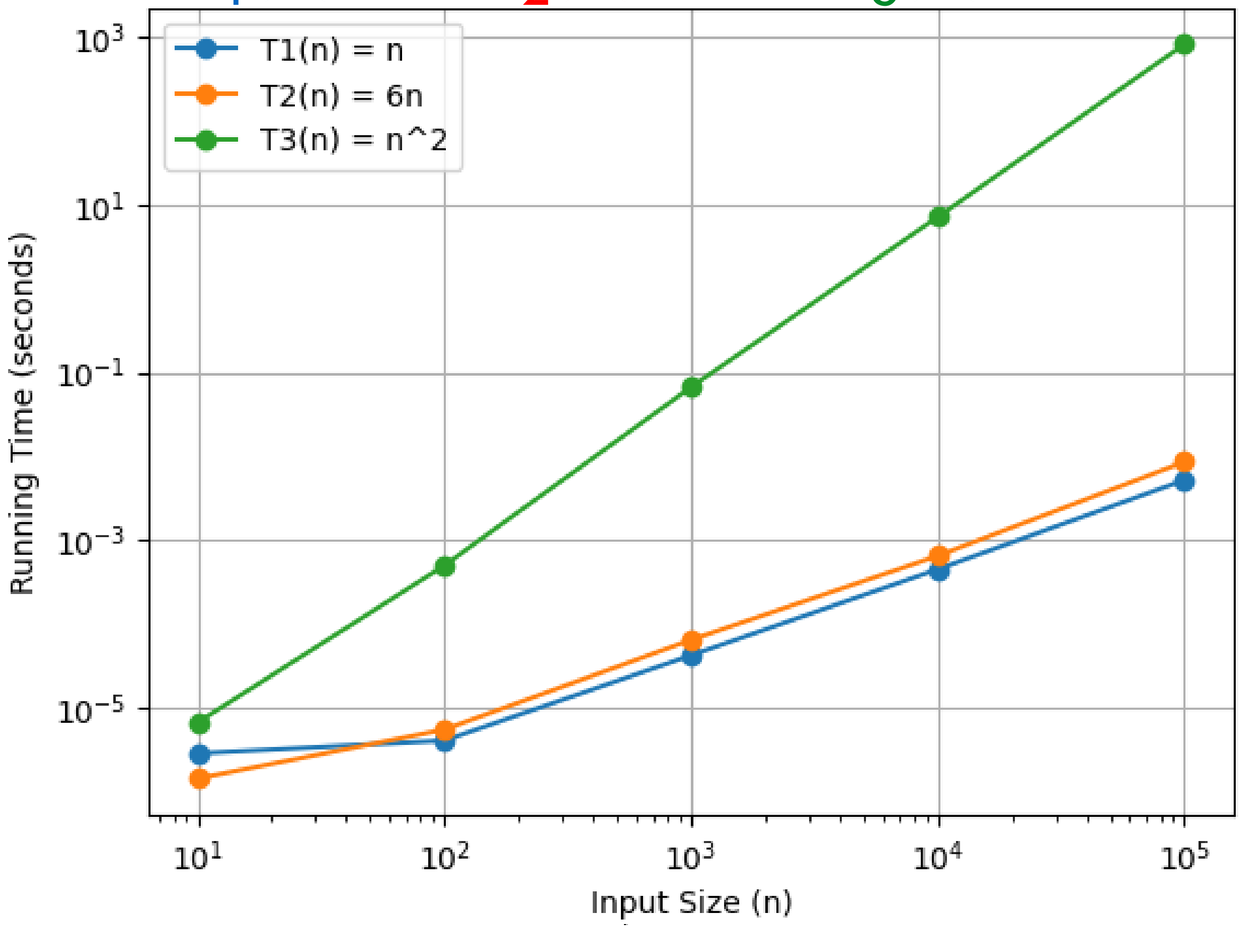


N	N + 7	N + 4	Ratio (T_1/T_2)
10	17	14	1.214
1000	1007	1004	1.00298
1,000,000	1000007	1000004	1.000002999988

Expressing Running Time

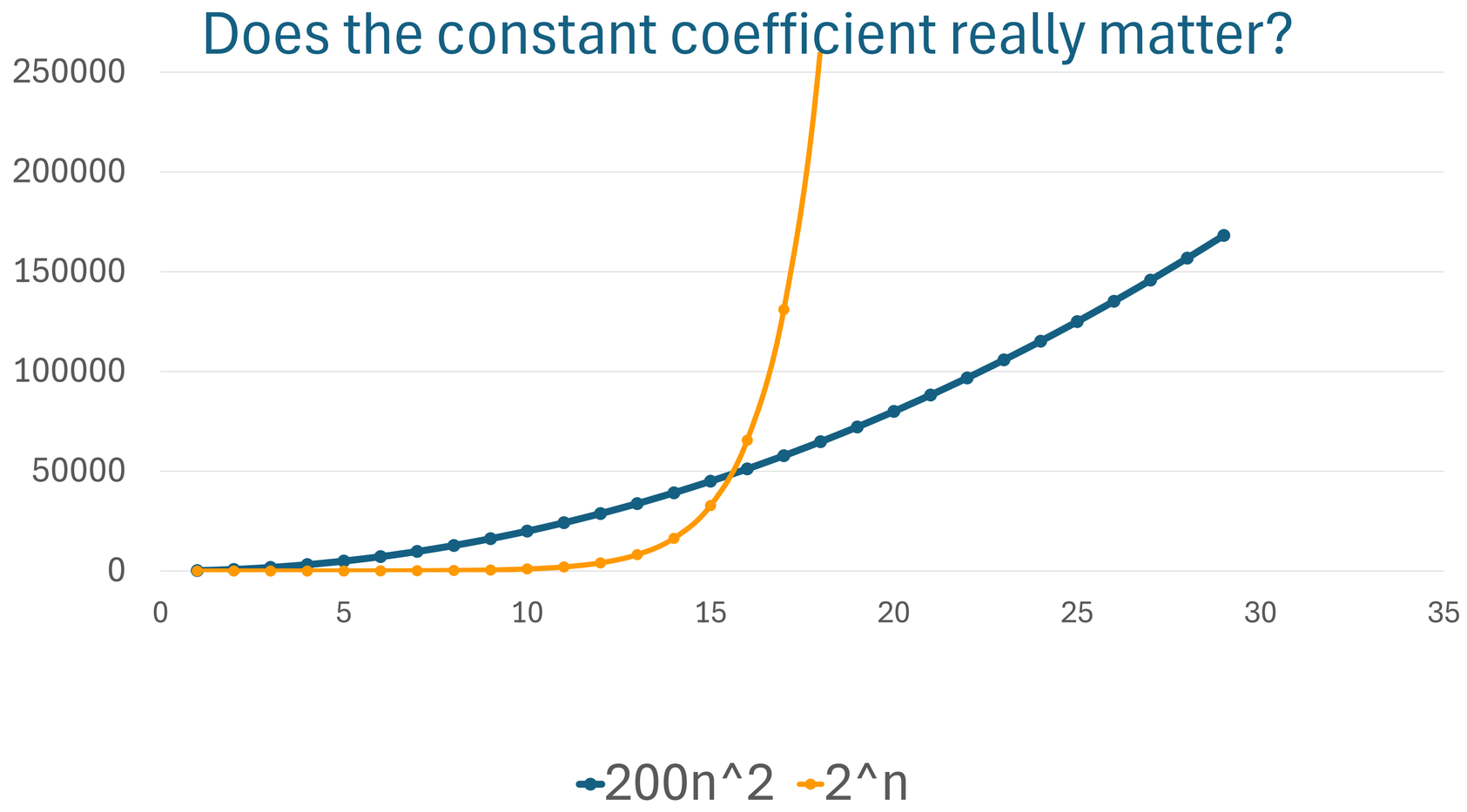
Do we really care about the multiplicative factor 6 if T(n)=6n?

$T_1(n)=n$, $T_2(n)=6n$, $T_3(n)=n^2$



T_1 and T_2 are both linear, so they eventually look similar at large input sizes but T_3 has a distinctly different growth rate

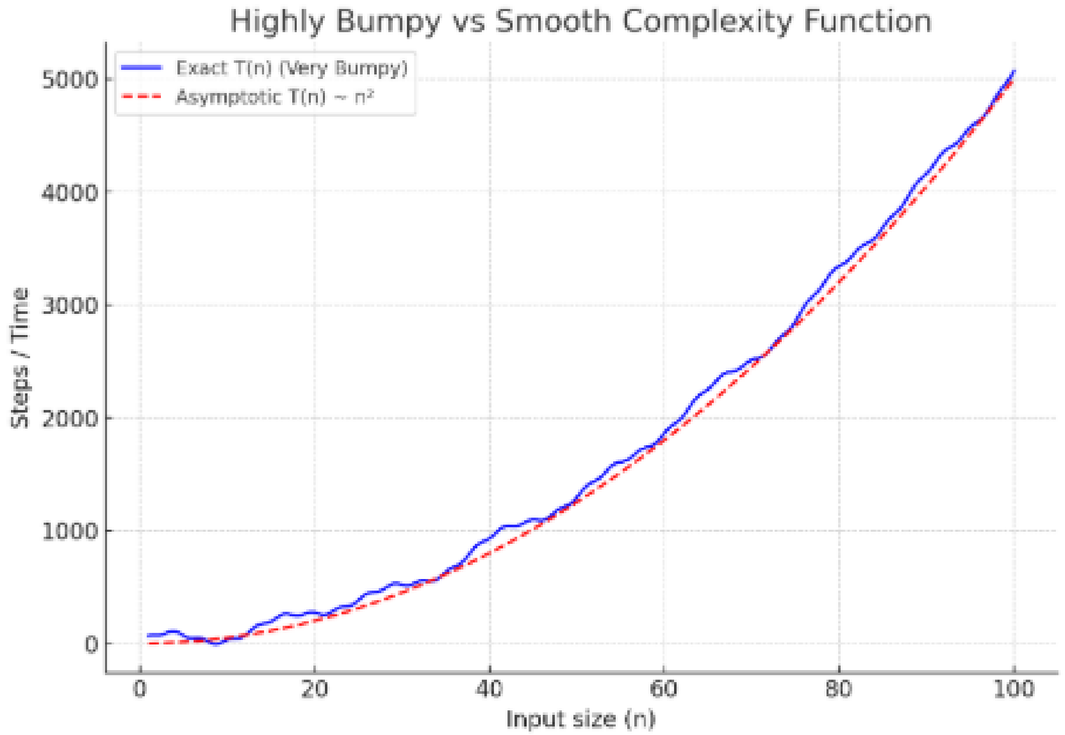
What is the smallest value of n such that an algorithm whose running time is 200n^2 runs faster than an algorithm whose running time is 2^n on the same machine?



n	200n^2	2^n
1	200	2
2	800	4
3	1800	8
4	3200	16
5	5000	32
6	7200	64
7	9800	128
8	12800	256
9	16200	512
10	20000	1024
11	24200	2048
12	28800	4096
13	33800	8192
14	39200	16384
15	45000	32768
16	51200	65536
17	57800	131072
18	64800	262144
19	72200	524288
20	80000	1048576
21	88200	2097152
22	96800	4194304
23	105800	8388608
24	115200	16777216
25	125000	33554432
26	135200	67108864
27	145800	134217728
28	156800	268435456
29	168200	536870912
30	180000	1073741824

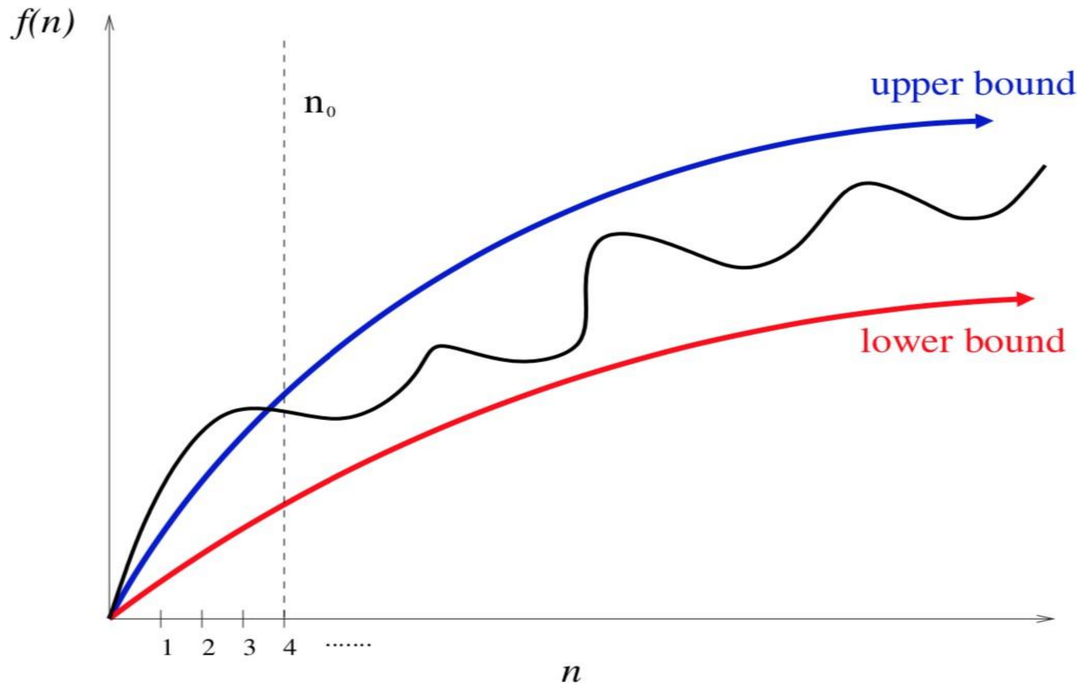
Expressing Running Time

- Time complexities are functions of input size
- In practice, these functions are hard to handle
  - $T(n) = 654n^2 + 987\log n + 1001$



Introducing Bounds

- Instead, it easier to talk about upper and lower bounds of the function
  - To achieve this, we use the asymptotic notation (O, Θ, Ω)



Entering Big-O Analysis

- The Big O simplifies our analysis by ignoring levels of detail that do not impact our comparison of algorithms

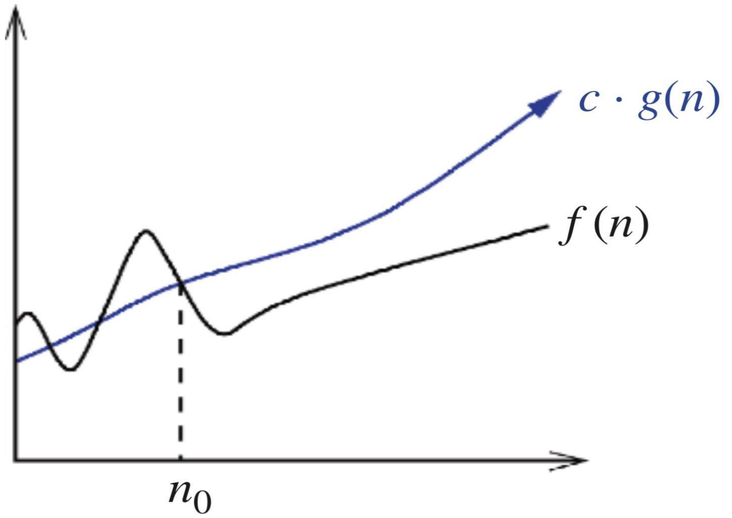
- It ignores constants, lower order terms, as well as multiplicative factors of the highest order term

- Thus, the functions $f(n) = 2n$ and $g(n) = n$ are equivalent in Big-O analysis

Asymptotic Analysis: Big-O

$f(n)$ is in $O(g(n))$ if there exists positive constants c, n_o such that for all $n \geq n_o$

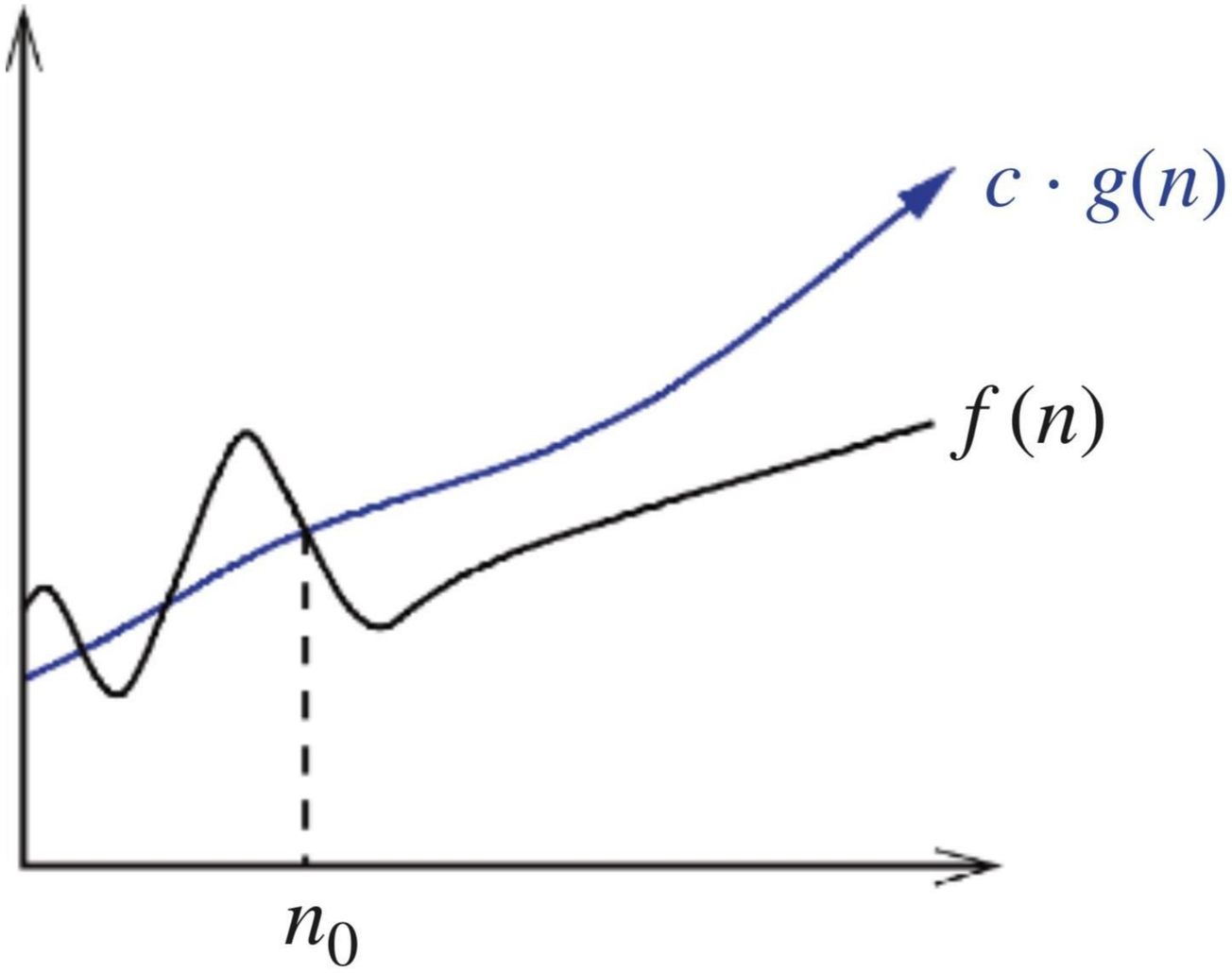
$$f(n) \leq c.O(g(n))$$



Note: c > 0 and $n_0 \geq 1$ (their values must NOT depend on n)

Informally, this means is $f(n)$ is less than some constant multiple of $g(n)$

Big-O



Big-O provides the upper bound on the growth rate.

The highest-order term is what drives growth!

Let's do Some Practice!

- Is $10n + 5 \in O(n)$?

$f(n)$ is $O(g(n))$ if there exists positive constants c, n_o such that for all $n \geq n_o$

$$f(n) \leq c.O(g(n))$$

Solve the inequality for c to find any (c, n_o) pair of values that satisfy the definition

Summary

- Data structures are systematic organizations of data
- They impact the efficiency of different operations
  - How fast can we search , insert or update/ delete a data item?

- Asymptotic analysis is used to measure the performance
- Big-O gives the upper bound on the growth rate
- $f(n)\ is\ in\ g(n)$ means that the growth rate of $f(n)$ is no larger than the growth rate of c.$(g(n)$

Questions