Algorithms and Data Structures 1
CS 0445

Fall 2022

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS1501 slides.)

Announcements

- Upcoming Deadlines:
  - Homework 3: this Friday @ 11:59 pm
  - Lab 2: next Monday @ 11:59 pm
  - Programming Assignment 1: Friday Oct. 7th
- Draft slides and handouts available on Canvas
- Lecture recordings are available under Panopto Video on Canvas
- Please use "Regrade Request" feature in GradeScope with any issues with grades
- Please include all instructors when sending private messages on Piazza, if possible
- **Student Support Hours** of the teaching team are posted on the Syllabus page

Previous Lecture …

- ADT Bag Implementations
  - Fixed-size array: ArrayBag
    - copy constructor
  - Resizable array: ResizableArrayBag
    - add
  - Linked implementation: LinkedBag

Muddiest Points

- **Q: What is the difference between ArrayBag and Array?**
- A: ArrayBag is an implementation that *uses* a fixed-size array to implement the ADT Bag
- **Q: I am still very confused with all the terminology. For example, “reference object”, reference variables vs. data objects**
- A: An object is an instance of a class. A reference variable points to an object.
  - For example, String x = new String();
  - *x* is a reference variable that points to the String object created by the **new** keyword
  - There is no such thing as “reference object”. Perhaps confused with reference**d** object.

Muddiest Points

- **Q: what is a null pointer exception**
- A: A NullPointerException is a run-time exception raised by the Java run-time every time a reference variable with null value is derefeferenced
  - e.g., ArrayBag<Integer> x;

        x.add(10);
  - The code above will raise a NullPointerException because the reference variable x is dereferenced (using the dot operator) while still being null.
  - How would you fix that?

Muddiest Points

- **Q: does the equal method just check that the two types of objects are the same?**
- A: Practically speaking, no.
- .equals typically checks for types and for values of instance variables.
- Theoretically, one may define equals to do whatever check one wants.

Muddiest Points

- **Q: must we type " "unchecked" " in the parentheses after a SuppressWarning**
- A: Yes, if we want to suppress the unchecked type cast warning. There are other vendor-specific warnings that can be suppressed.

Muddiest Points

- **Q: Traversing the nodes is a bit confusing**
- A: When traversing the nodes of a linked chain, we follow the following steps
  - initialize a *scout* variable to point to the first node
    - Node scout = firstNode;
  - keep moving the scout variable over the nodes until it traverses over the last node
  - How do we know that the scout traversed over the last node
    - while(scout != null)
  - How do we move the scout variable to the next node
    - scout = scout.next

```
Node scout = firstNode
while(scout != null){
  //do something with the node pointed to by scout
   scout = scout.next;
}
```

Muddiest Points

- **Q: Can you go over specific item node removal again?**
- **Q: Maybe if you could go over once more the difference between removing a specified and an unspecified item from a list.**
- A: To remove an unspecific item, we follow the following steps
  - Save the data object pointed to by the first node by making a reference variable point to it
  - Remove the first node
- A: To remove a specific item, we follow the following steps
  - traverse the nodes to find a node that points to an equal object
  - Save the object by making a reference variable point to it
  - Make the found node point to the data object of the first node
  - Remove the first node
- **Q: When removing an item from a linked list how do you make the first item = to the first node.**
- A: The first node points to the first data item.

Muddiest Points

- **Q: Why don't we add new nodes to the end instead of the beginning?**
- A: When we add to the beginning of the chain, we don’t need to traverse the chain to reach the last node. This makes adding at the beginning *faster*.
- **Q: Where is the information stored on other nodes when using .data**
- A: Objects are stored on a memory region called “the heap”

Muddiest Points

- **Q: What would make the difference between a LinkedList and a LinkedBag? the order?**
- A: Yes, and the set of operations that are make sense once you have the items ordered.
- **Q: What is the function of the referenceTo method?**
- A: To traverse the nodes until a node with an equal object found. If so, return a reference to the node; otherwise, return null.

Muddiest Points

- **Q: How can you have a node as a private variable within the Node class? Wouldn't it not be defined yet?**
- A: Java compiler parses class declarations before name resolution
- What would make the difference between a LinkedList and a LinkedBag? the order?
- What is the function of the referenceTo method?

Muddiest Points

Q: If we were provided for at least a minute or two at the end of class there may be more muddiest points provided. Ending lecture exactly (or after!) scheduled class time causes many of us to run to our next classes and forgo muddiest points.

- A: I am sorry about that. Noted!

Today's Agenda

- A final thought on LinkedBag
- Code efficiency
- ADT List
    - Fixed-size array implementation: ArrayList

Cons of Using a Chain

- Removing specific entry requires search of array or chain
- Chain requires more memory than array of same logical size
  - why?

Why do we care about efficient code?

- Computers are faster, have larger memories
  - So why worry about efficient code?
- And … how do we measure efficiency?

Example

- Consider the problem of summing: computing the sum
  1 + 2 + . . . + n for an integer n > 0

$$\sum_{k=1}^{n} k = 1 + 2 + 3 + \ldots + n$$

One solution

Algorithm A

```
sum = 0
for i = 1 to n
    sum = sum + i
```

```
// Computing the sum of the consecutive integers from 1 to n:
long n = 10000; // Ten thousand

// Algorithm A
long sum = 0;
for (long i = 1; i <= n; i++)
   sum = sum + i;
System.out.println(sum);
```

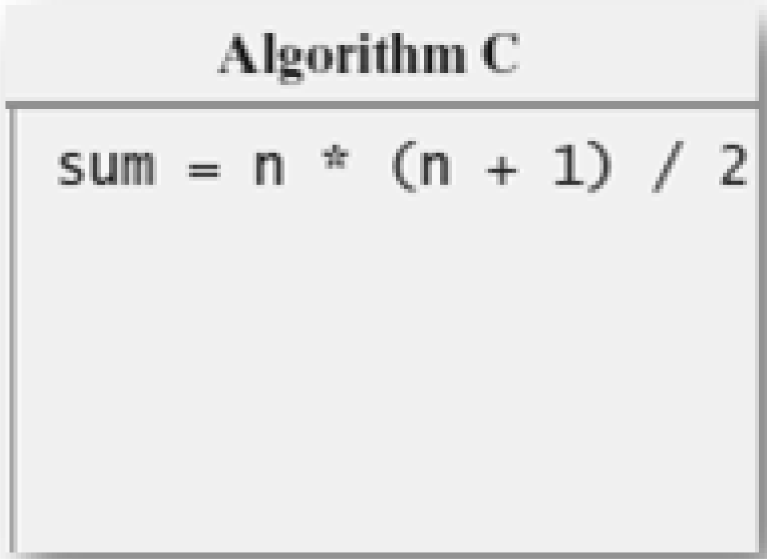
Another solution

Algorithm B

```
sum = 0
for i = 1 to n
{
    for j = 1 to i
        sum = sum + 1
}
```

```
// Algorithm B
sum = 0;
for (long i = 1; i <= n; i++)
{
   for (long j = 1; j <= i; j++)
       sum = sum + 1;
} // end for
System.out.println(sum);
```

And a third solution



Algorithm C

```
sum = n * (n + 1) / 2
```

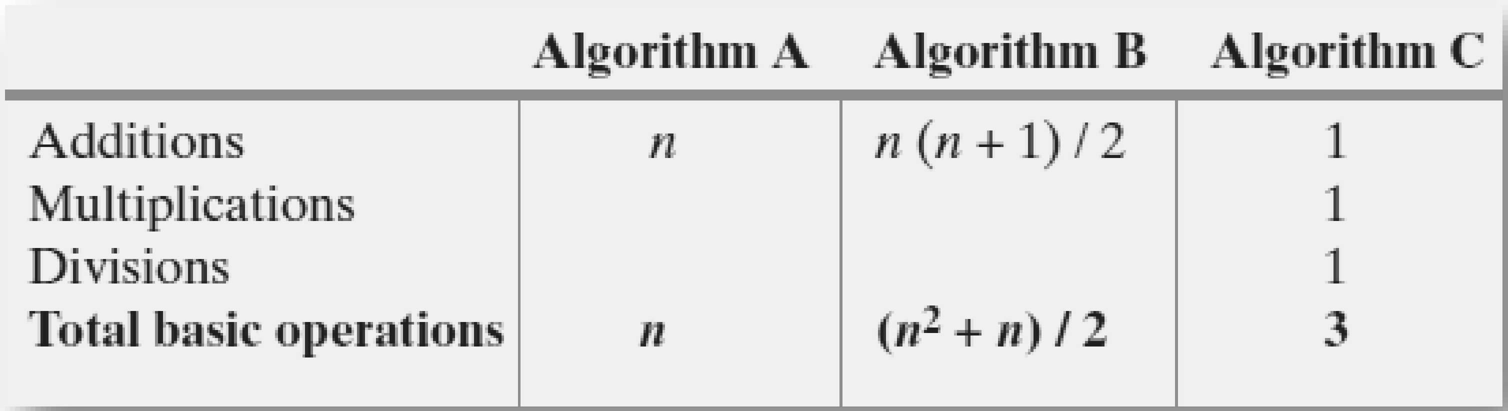
```java
// Algorithm C
sum = n * (n + 1) / 2;
System.out.println(sum);
```

Which is “best”?

- An algorithm has both time and space constraints – that is complexity
  - Time complexity
  - Space complexity
- The study of time and space complexities of algorithms is called analysis of algorithms

Counting Basic Operations

- A basic operation of an algorithm
  - The most significant contributor to its total time requirement

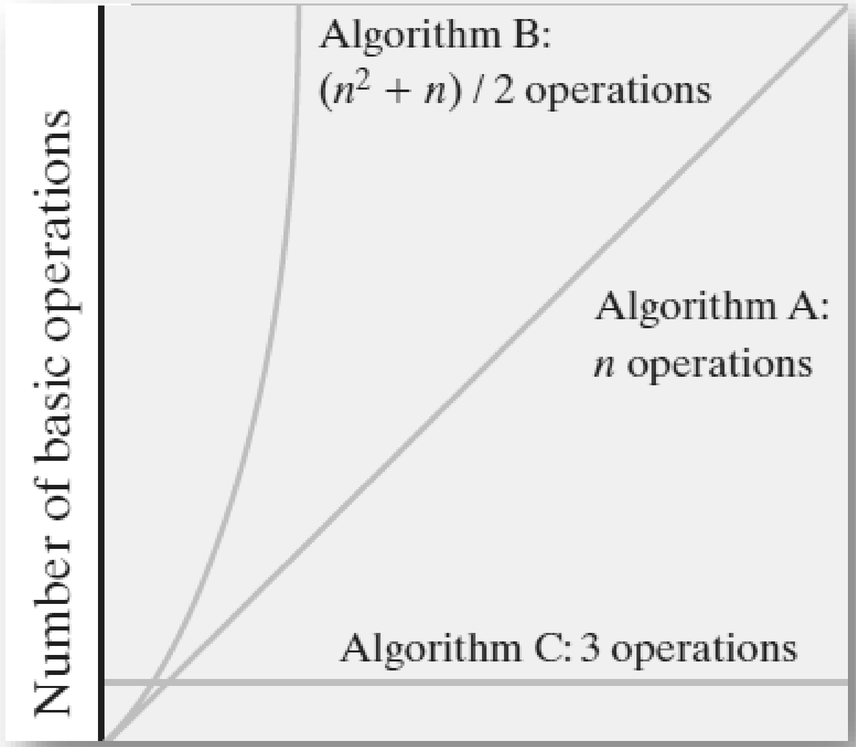


	Algorithm A	Algorithm B	Algorithm C
Additions	n	$n(n+1)/2$	1
Multiplications			1
Divisions			1
Total basic operations	n	$(n^2+n)/2$	3

- The number of basic operations required by the sum algorithms

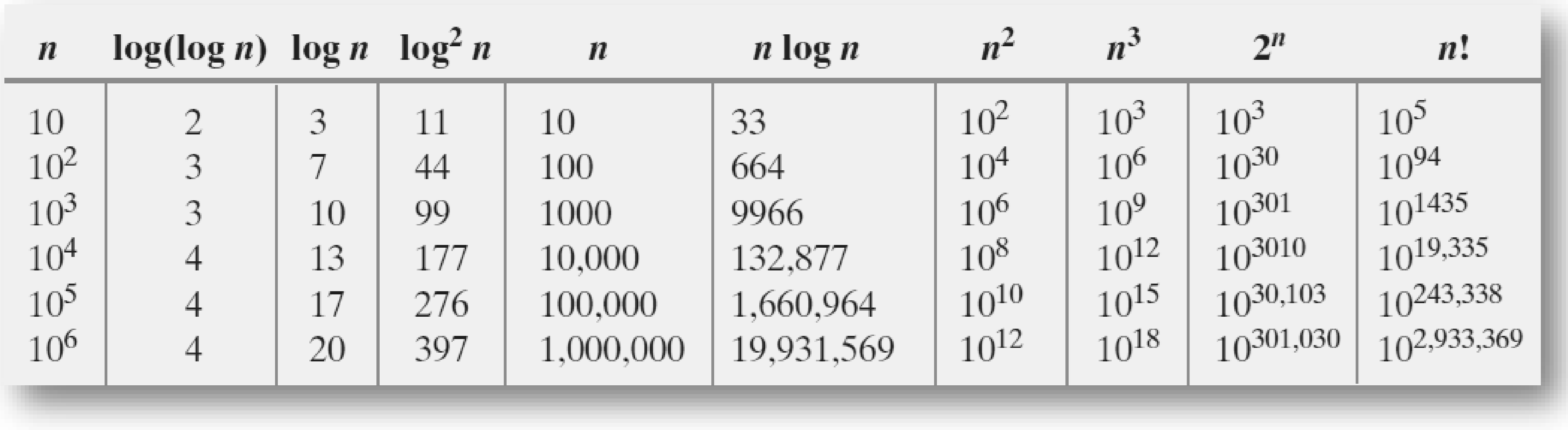
Counting Basic Operations

- The number of basic operations required by the sum algorithms as a function of *n*



Counting Basic Operations

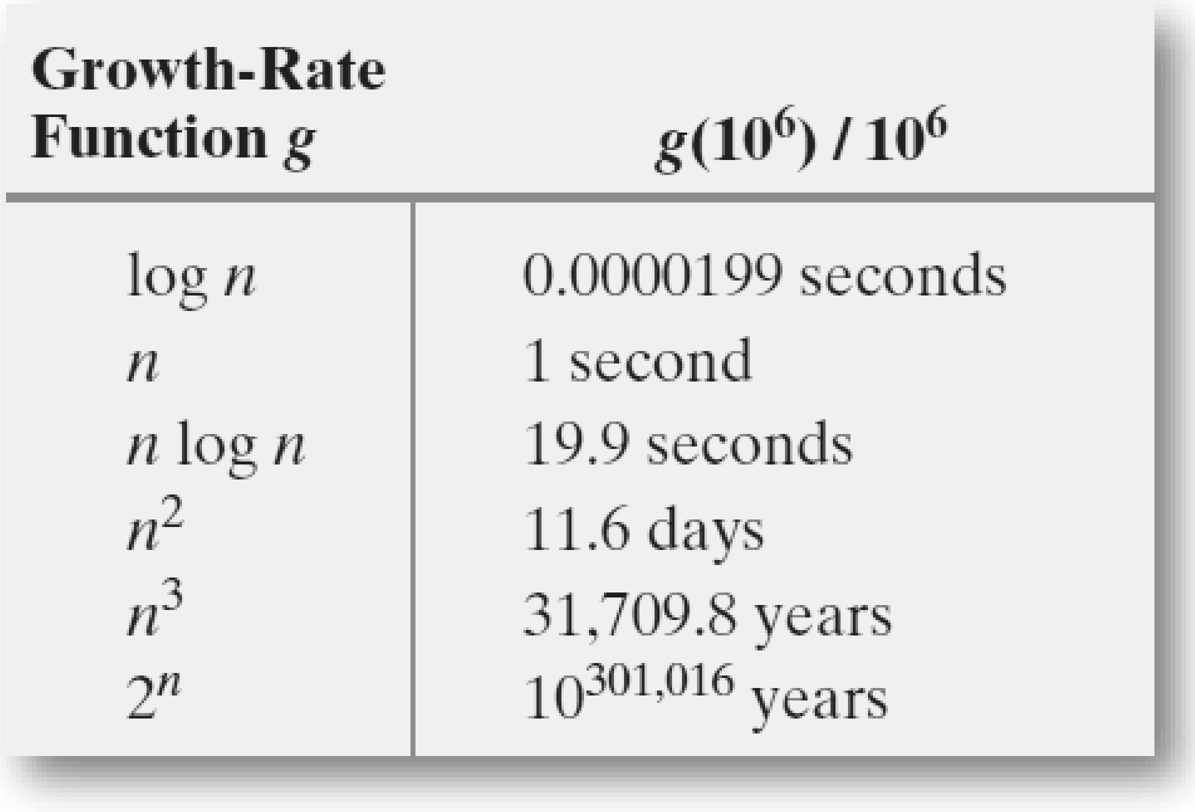
- Typical growth-rate functions evaluated at increasing values of *n*



n	$\log(\log n)$	$\log n$	$\log^2 n$	n	$n \log n$	n^2	n^3	2^n	$n!$
10	2	3	11	10	33	10^2	10^3	10^3	10^5
10^2	3	7	44	100	664	10^4	10^6	10^{30}	10^{94}
10^3	3	10	99	1000	9966	10^6	10^9	10^{301}	10^{1435}
10^4	4	13	177	10,000	132,877	10^8	10^{12}	10^{3010}	$10^{19,335}$
10^5	4	17	276	100,000	1,660,964	10^{10}	10^{15}	$10^{30,103}$	$10^{243,338}$
10^6	4	20	397	1,000,000	19,931,569	10^{12}	10^{18}	$10^{301,030}$	$10^{2,933,369}$

Picturing Efficiency

- The time required to process one million items by algorithms of various orders at the rate of one million operations per second



Growth-Rate Function g	**$g(10^6) / 10^6$**
$\log n$	0.0000199 seconds
n	1 second
$n \log n$	19.9 seconds
n^2	11.6 days
n^3	31,709.8 years
2^n	$10^{301,016}$ years

Best, Worst, and Average Cases

- For some algorithms, execution time depends only on size of data set
- Other algorithms depend on the nature of the data itself
    - Here we seek to know best case, worst case, average case

Time complexity of an algorithm

- Count the number of **<u>executed</u>** steps (basic operations or just lines of code)

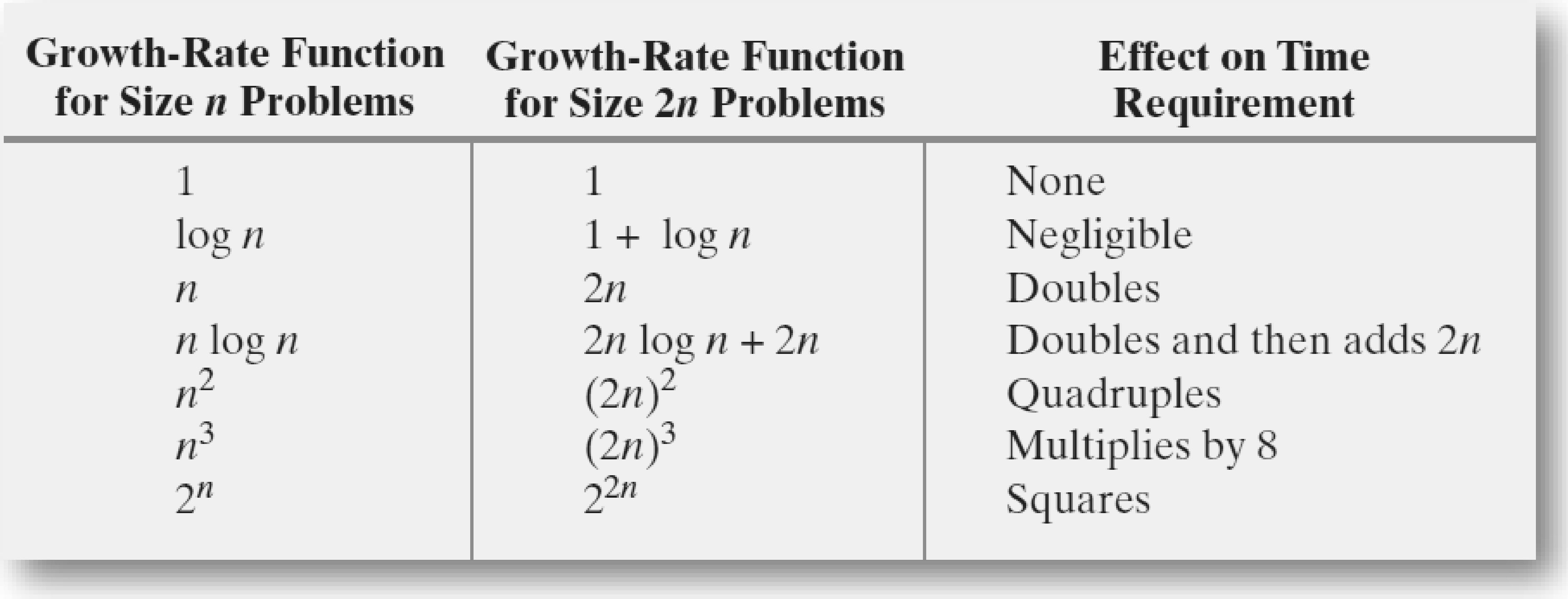
  -
    ```
    sum = 0
    for i = 1 to n
      sum = sum + i
    ```
  - Number of executed lines is 2n + 2
- Let *f(n)* = the number of executed steps
  - *n* is the input size
    - very roughly, the number of keyboard presses needed to enter the input
  - *f(n)* may depend only on *n* or on the actual values of the input
    - In the latter, need to find *f(n)* for best, average, worst cases

Time complexity of an algorithm

- Convert the function f into the ***Big-Oh notation***
  - Ignore lower order terms
    - e.g., constant $<$ log log n $<$ log n $<$ $\log^2 n$ $<$ n $<$ n log n $<$ n^2 $<$ n^3 $<$ 2^n $<$ n!
    - e.g., $n^2 + \log n = O(n^2)$
  - Ignore constant factors
    - $c*n = O(n)$, where c is a constant (doesn't depend on *n*)
    - 2^{cn} is **not** $O(2^n)$
  - $f(n) = 2n + 2 = O(2n) = O(n)$

Picturing Efficiency

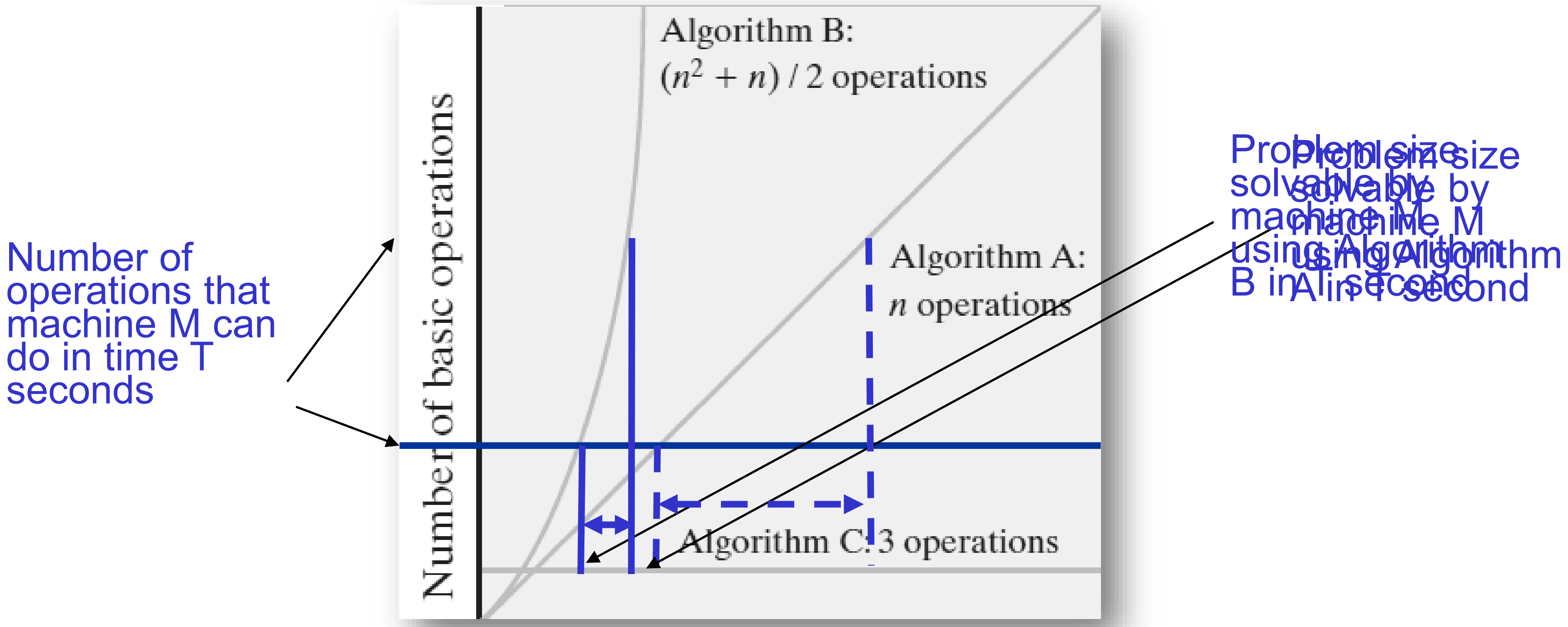
- The effect of doubling the problem size on an algorithm's time requirement



Growth-Rate Function for Size n Problems	Growth-Rate Function for Size $2n$ Problems	Effect on Time Requirement
1	1	None
$\log n$	$1 + \log n$	Negligible
n	$2n$	Doubles
$n \log n$	$2n \log n + 2n$	Doubles and then adds $2n$
n^2	$(2n)^2$	Quadruples
n^3	$(2n)^3$	Multiplies by 8
2^n	2^{2n}	Squares

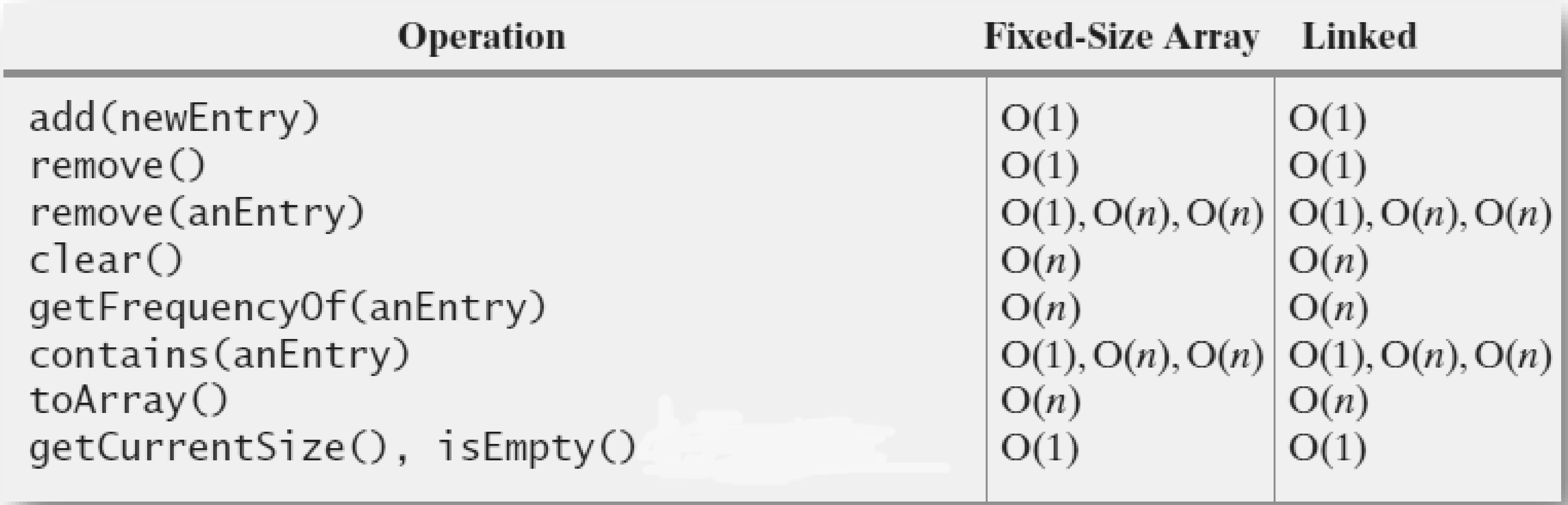
Riding Moore’s law

- Writing an efficient algorithm (with less time complexity) is important
  - Such algorithm rides the exponentially-growing curve of hardware-speed ``better”


Algorithm B: $(n^2 + n) / 2$ operations
Algorithm A: n operations
Algorithm C: 3 operations
Number of basic operations
Number of operations that machine M can do in time T seconds
Problem size solvable by machine M using Algorithm B in T second
Problem size solvable by machine M using Algorithm A in T second

Efficiency of Implementations of ADT Bag

- The time efficiencies of the ADT bag operations for two implementations, expressed in Big Oh notation



Operation	Fixed-Size Array	Linked
`add(newEntry)`	$O(1)$	$O(1)$
`remove()`	$O(1)$	$O(1)$
`remove(anEntry)`	$O(1), O(n), O(n)$	$O(1), O(n), O(n)$
`clear()`	$O(n)$	$O(n)$
`getFrequencyOf(anEntry)`	$O(n)$	$O(n)$
`contains(anEntry)`	$O(1), O(n), O(n)$	$O(1), O(n), O(n)$
`toArray()`	$O(n)$	$O(n)$
`getCurrentSize(), isEmpty()`	$O(1)$	$O(1)$