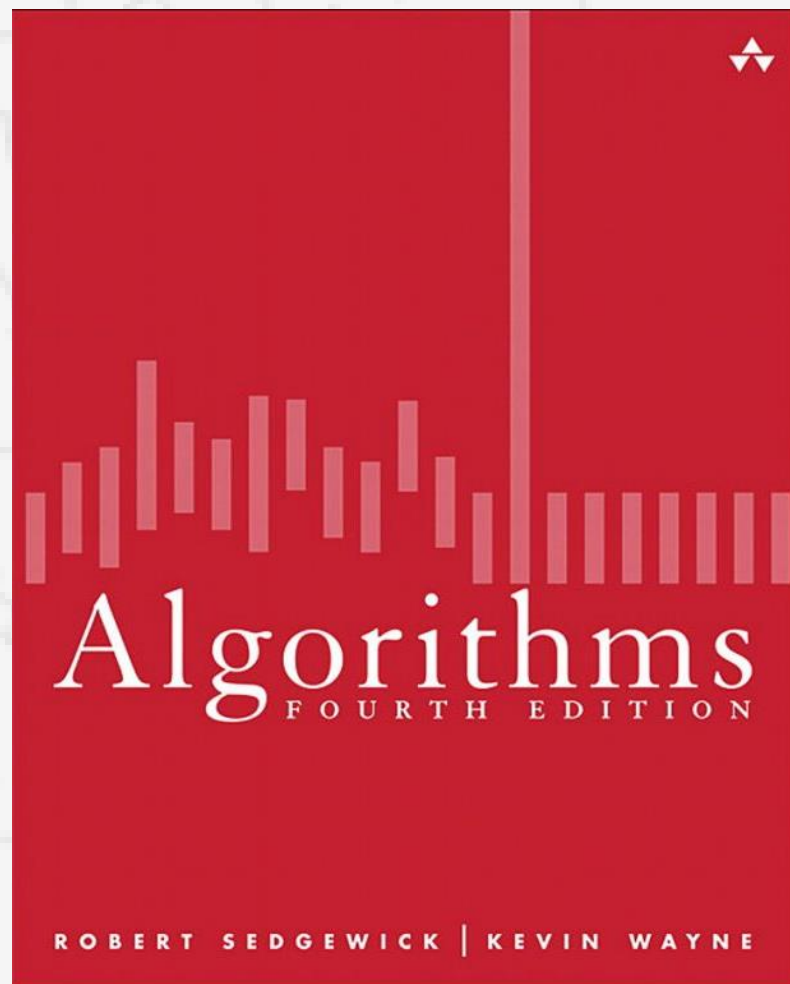
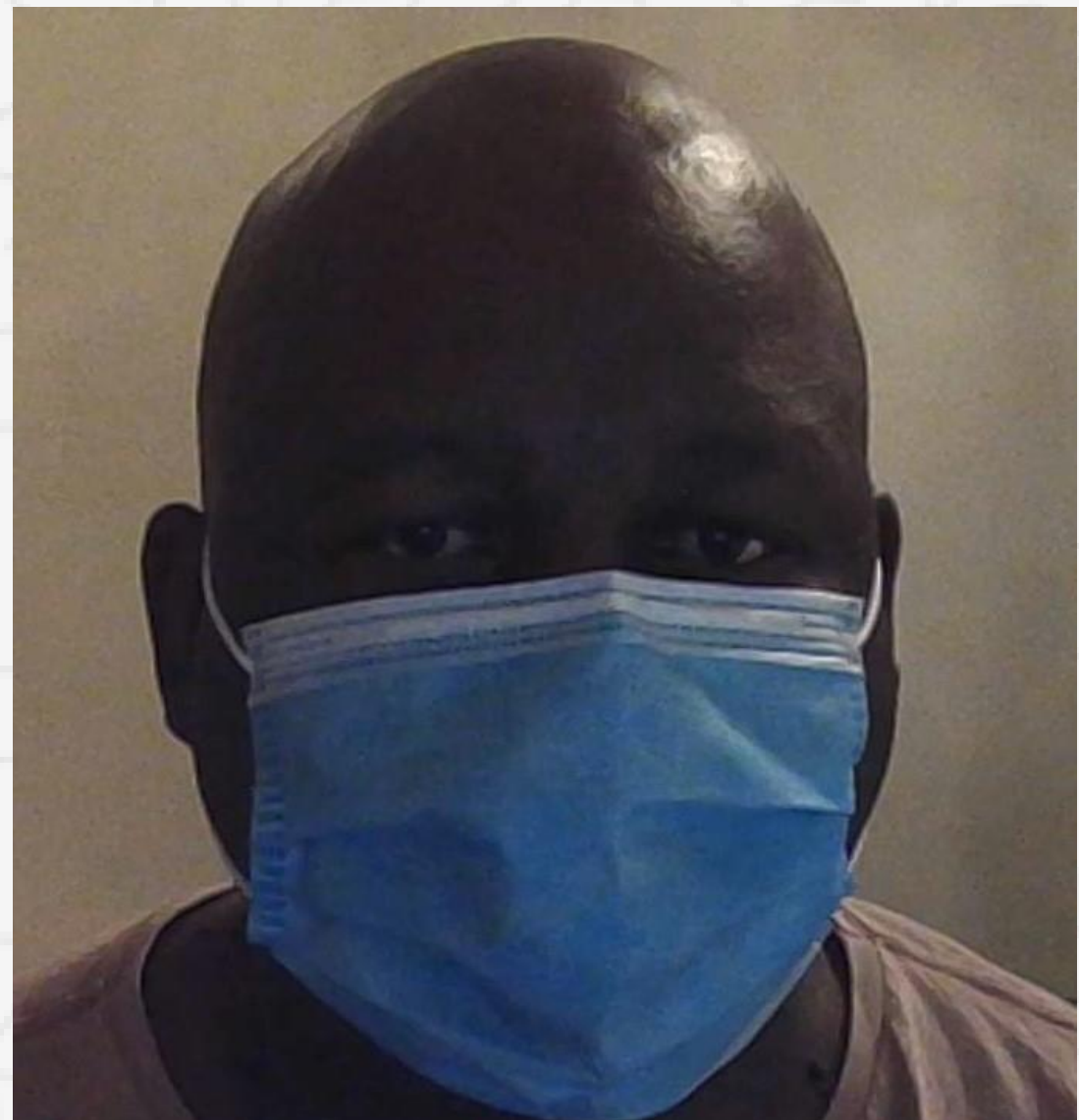


STAY SAFE

WASH HANDS WITH SOAP & WATER
SANITIZE
SOCIAL DISTANCE



<https://algs4.cs.princeton.edu>



COMPUTER SCIENCE



<https://algs4.cs.princeton.edu>

CS1323

ALGORITHMS



LECTURE 2

- ▶ *RAM model of computation*
- ▶ *Decision tree model of computation*

<https://algs4.cs.princeton.edu>

**ACKNOWLEDGEMENT:
SLIDES' THEME & CONTENT FROM
ROBERT SEDGEWICK | KEVIN WAYNE**



<https://algs4.cs.princeton.edu>



<https://algs4.cs.princeton.edu>

LECTURE 2

- ▶ *RAM model of computation*
- ▶ *Decision tree model of computation*

Last time..

- We defined what a problem is: **Input instance and task to be performed**
- We defined what an algorithm is: **a mapping between valid input instances and possible outputs**
- We discussed properties of algorithms:
 - **Finiteness** – must terminate after a finite number of steps
 - **Definiteness** – each step is precisely defined
 - **Effectiveness** – operations require less space and time resources (may need to strike a balance)
 - **Input/output** – at least 0 inputs and at least 1 output
 - **Correctness** – Output should be what is desired

In this course, we will focus on **effectiveness** and **correctness**. We need a level playing field for comparing algorithms: **model computer**

How to evaluate an algorithm


- **Time complexity:** rate of growth of the time the algorithm takes as a function of the input size
- **Space complexity:** rate of growth of the space required by the algorithm as a function of the input size

Input size: reasonable measure of the quantity(integer) of input data

- for an **array**, it might be the number of elements in the array or the size of the array
- for a **graph**, it might be the number of edges + number of vertices

Time complexity is usually determined by **cost** of executing each statement and the **frequency** (number of times) of execution of each statement

Depends on computer,
compiler, operating system



Depends on input,
algorithm



We will perform time complexity analysis using **mathematical analysis** and **experimental studies** (using the scientific method)

But hardware is evolving...

- Suppose we have the following algorithms and their time complexity for problem P

Algorithm	Time complexity
A_1	n
A_2	$n \lg n$
A_3	n^2
A_4	n^3
A_5	2^n

← $\lg = \log_2$

- Assume time complexity refers to the number of time units to process input of size n . Let 1 unit of time be 1 millisecond ($1 \times 10^{-3} \text{ sec}$).

Algorithm	Time complexity	Maximum problem size in	
		1 min	1 hour
A_1	n	6.0×10^4	3.6×10^6
A_2	$n \lg n$	4893	2.0×10^5
A_3	n^2	244	1897
A_4	n^3	39	153
A_5	2^n	15	21

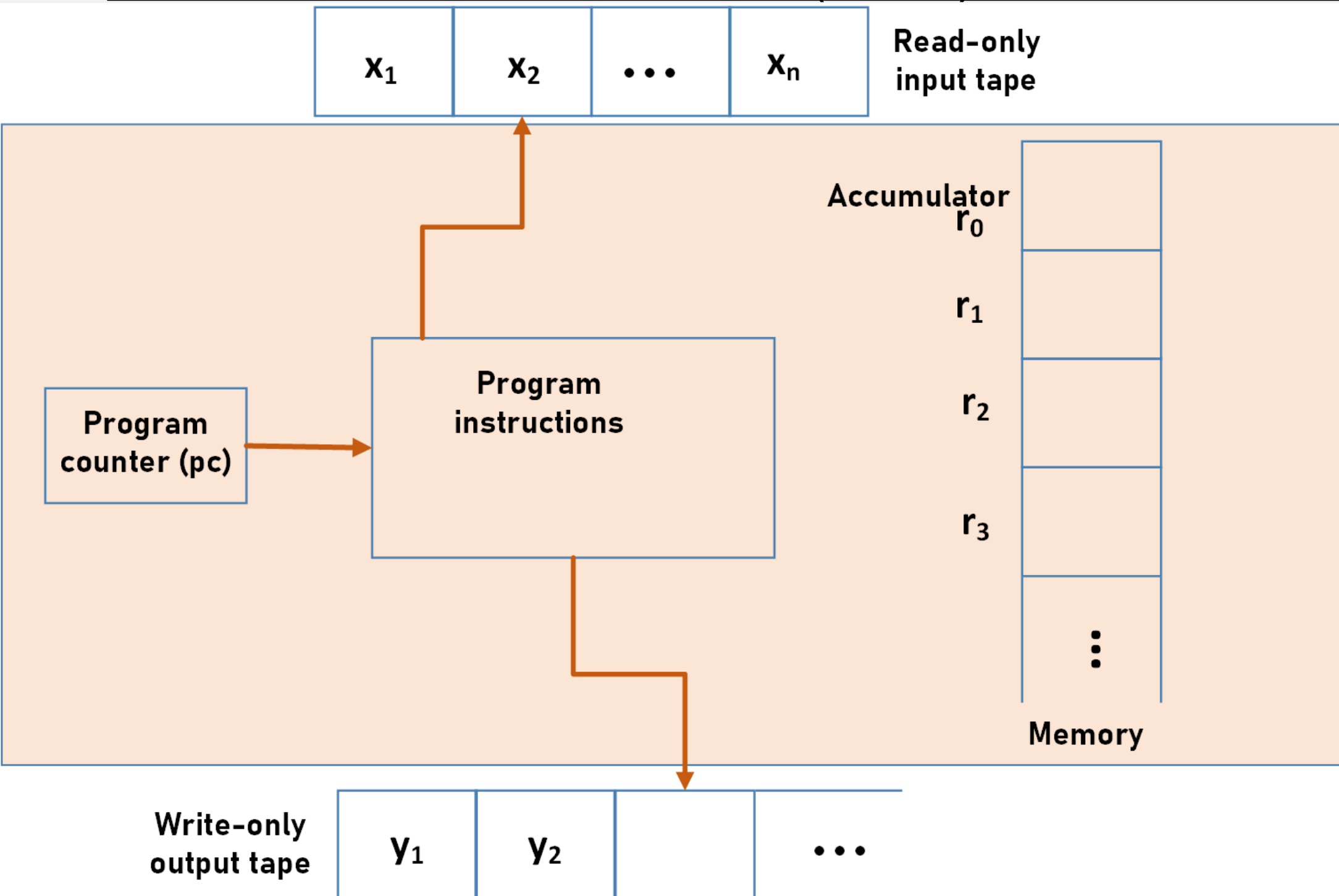
size
= *speed* \times *time*

But hardware is evolving...

- Assume time complexity refers to the number of time units to process input of size n . Let 1 unit of time be 1 millisecond.
- What if our new computer is 100 times faster?

Algorithm	Time complexity	Current computer Maximum size	New computer Maximum size
A_1	n	s_1	$100s_1$
A_2	$n \lg n$	s_2	$\approx 100s_2$ (for large s_2)
A_3	n^2	s_3	$10s_3$
A_4	n^3	s_4	$4.64s_4$
A_5	2^n	s_5	$s_5 + 6.64$

Random Access Machine (RAM) model of computation



Note

- Has 1 accumulator (r_0).
- Each register can hold an integer of arbitrary size
- Unlimited number of registers/memory
- Program is not stored in memory. So program cannot modify itself.

Instruction set

- We would have arithmetic, input/output, and indirect addressing
- All computation takes place in the first register (the accumulator), r_0
- An instruction consists of an **operation code** and an **address**

Operation code	Address
1. load	operand
2. store	operand
3. add	operand
4. sub	operand
5. mult	operand
6. div	operand
7. read	operand
8. write	operand
9. jump	label
10. jgtz	label
11. jzero	label
12. halt	

An operand could be any of

1. $= i$, indicating the integer i itself
2. A nonnegative integer i , indicating the contents of register i
3. $* i$, indicating indirect addressing. The operand is the contents of register j , where j is the integer in register i . If $j < 0$, the machine halts

Meaning of instructions

- Let $c(i)$ be the integer stored in register i . (contents of the register)
- Let $v(a)$ be the value of operand a .

$$v(= i) = i,$$

$$v(i) = c(i)$$

$$v(* i) = c(c(i))$$

Instruction	Meaning
1. <i>load a</i>	$c(0) \leftarrow v(a)$
2. <i>store i</i> <i>store * i</i>	$c(i) \leftarrow c(0)$ $c(c(i)) \leftarrow c(0)$
3. <i>add a</i>	$c(0) \leftarrow c(0) + v(a)$
4. <i>sub a</i>	$c(0) \leftarrow c(0) - v(a)$
5. <i>mult a</i>	$c(0) \leftarrow c(0) \times v(a)$
6. <i>div a</i>	$c(0) \leftarrow \left\lfloor \frac{c(0)}{v(a)} \right\rfloor$
7. <i>read i</i> <i>read * i</i>	$c(i) \leftarrow \text{current input}$ $c(c(i)) \leftarrow \text{current input}$
8. <i>write a</i>	Print $v(a)$

An operand could be any of

- $= i$, indicating the integer i itself
- A nonnegative integer i , indicating the contents of register i
- $* i$, indicating indirect addressing. The operand is the contents of register j , where j is the integer in register i . If $j < 0$, the machine halts

Meaning of instructions

Instruction	Meaning
1. <i>load a</i>	$c(0) \leftarrow v(a)$
2. <i>store i</i> <i>store * i</i>	$c(i) \leftarrow c(0)$ $c(c(i)) \leftarrow c(0)$
3. <i>add a</i>	$c(0) \leftarrow c(0) + v(a)$
4. <i>sub a</i>	$c(0) \leftarrow c(0) - v(a)$
5. <i>mult a</i>	$c(0) \leftarrow c(0) \times v(a)$
6. <i>div a</i>	$c(0) \leftarrow \left\lfloor \frac{c(0)}{v(a)} \right\rfloor$
7. <i>read i</i> <i>read * i</i>	$c(i) \leftarrow \text{current input}$ $c(c(i)) \leftarrow \text{current input}$
8. <i>write a</i>	Print $v(a)$
9. <i>jump b</i>	Program counter (pc) is set to the instruction labeled b
10. <i>jgtz b</i>	Program counter (pc) is set to the instruction labeled b if $c(0) > 0$, otherwise increment pc
11. <i>jzero b</i>	Program counter (pc) is set to the instruction labeled b if $c(0) = 0$, otherwise increment pc
12. <i>halt</i>	Execution halts

Example 1

Write a RAM program to compute the function $f(n) = 2n^2 + 7$


```
1. read 1
2. load 1
3. mult 1
4. store 1
5. load =2
6. mult 1
7. store 1
8. load =1
9. add 1
10.write 0
```

```
 $c(1) \leftarrow \text{current input}$ 
 $c(0) \leftarrow v(1)$ 
 $c(0) \leftarrow c(0) \times v(1)$ 
 $c(1) \leftarrow c(0)$ 
 $c(0) \leftarrow v(= 2)$ 
 $c(0) \leftarrow c(0) \times v(1)$ 
 $c(1) \leftarrow c(0)$ 
 $c(0) \leftarrow v(= 1)$ 
 $c(0) \leftarrow c(0) + v(1)$ 
print  $v(0)$ 
```

Example 2

Write a RAM program to compute the sum of all the odd integers less than or equal to a positive integer N.

```
1. read 1
2. load =0
3. store 3
4. load 1
5. store 2
6. jzero end
7. load =1
8. store 4
9. load =2
10. store 5
11. if1: load 2
12. div 5
13. mult 5
14. store 6
15. load 2
16. sub 6
17. jzero sub1
18. load 3
19. add 2
20. store 3
21. sub1: load 2
22. sub 4
23. store 2
24. jzero end
25. jump if1
26. end: write 3
27. halt
```



```
1. read r1
2. r3 =0
3. r2 = r1
4. while r2 > 0
5.     if r2 is odd
6.         r3 = r3 + r2
7.         r2 = r2 - 1
8. return r3
```


Computational complexity of RAM programs

- Interested in **time complexity** and **space complexity**
- Define the following
$$T_A(n) = T_A(x) = \text{time cost for algorithm } A \text{ on input } x$$
$$S_A(n) = S_A(x) = \text{space cost for algorithm } A \text{ on input } x$$
$$|x| = n \text{ is the quantity/size of input}$$
- But, there are many different inputs of size n

Computational complexity of RAM programs

- Worst-case **time complexity**

$$T_A(n) = \max \{T_A(x) : |x| = n\}$$

That is, maximum time taken by any input of size **n**.

- Worst-case **space complexity**

$$S_A(n) = \max \{S_A(x) : |x| = n\}$$

That is, maximum space used on input of size **n**.

We exclude the space used by input.

- Sometimes we speak of average/expected time and space complexity

Average time complexity $E(T_A(n)) = \sum_{\{x:|x|=n\}} T_A(x) \times Pr(x)$

Average space complexity $E(S_A(n)) = \sum_{\{x:|x|=n\}} S_A(x) \times Pr(x)$

Cost criteria for RAM programs

- **Uniform cost**
 - Time complexity = number of RAM instructions executed
 - Space complexity = number of memory registers used
- **Logarithmic cost (cost is approximately proportional to the length of operands)**

Define the following

$$l(i) = \begin{cases} \lfloor \log |i| \rfloor + 1, & i \neq 0 \\ 1, & i = 0 \end{cases}$$

Cost of operands, $t(a)$

1. $= i \rightarrow t(= i) = l(i)$

2. $i \rightarrow t(i) = l(i) + l(c(i))$

3. $ i \rightarrow t(* i) = l(i) + l(c(i)) + l(c(c(i)))$*

size of i

size of $c(i)$

size of $c(c(i))$

Logarithmic cost of RAM instructions

Instruction	Cost
1. <i>load a</i>	$t(a)$
2. <i>store i</i> <i>store * i</i>	$l(c(0)) + l(i)$ $l(c(i)) + l(i) + l(c(0))$
3. <i>add a</i>	$l(c(0)) + t(a)$
4. <i>sub a</i>	$l(c(0)) + t(a)$
5. <i>mult a</i>	$l(c(0)) + t(a)$
6. <i>div a</i>	$l(c(0)) + t(a)$
7. <i>read i</i> <i>read * i</i>	$l(input) + l(i)$ $l(c(i)) + l(i) + l(input)$
8. <i>write a</i>	$t(a)$
9. <i>jump b</i>	1
10. <i>jgtz b</i>	$l(c(0))$
11. <i>jzero b</i>	$l(c(0))$
12. <i>halt</i>	1

Logarithmic space complexity

- Sum, over all registers, of $l(x_i)$, where x_i is the integer of largest magnitude stored in any register during computation

Example 1 – Computational complexity

Write a RAM program to compute the function $f(n) = 2n^2 + 7$

```
1. read 1
2. load 1
3. mult 1
4. store 1
5. load =2
6. mult 1
7. store 1
8. load =1
9. add 1
10. write 0
11. halt
```

Uniform cost

- $T(n) = 11$
- $S(n) = 2$

Logarithmic cost
Largest integer ever stored is $2n^2 + 7$ in register r_0


- $T(n) \leq 11 \times 2 \lg n$
- $S(n) = 2 \times 2 \lg n$

As $n \rightarrow \infty, 2n^2 + 7 \approx 2n^2$

Example 2

Write a RAM program to compute the sum of all the odd integers less than or equal to a positive integer N.

```
1. read 1
2. load =0
3. store 3
4. load 1
5. store 2
6. jzero end
7. load =1
8. store 4
9. load =2
10. store 5
11. if1: load 2
12. div 5
13. mult 5
14. store 6
15. load 2
16. sub 6
17. jzero sub1
18. load 3
19. add 2
20. store 3
21. sub1: load 2
22. sub 4
23. store 2
24. jzero end
25. jump if1
26. end: write 3
27. halt
```



Uniform cost

- $T(n) = 11 + 25n$
- $S(n) = 6$

Worst case

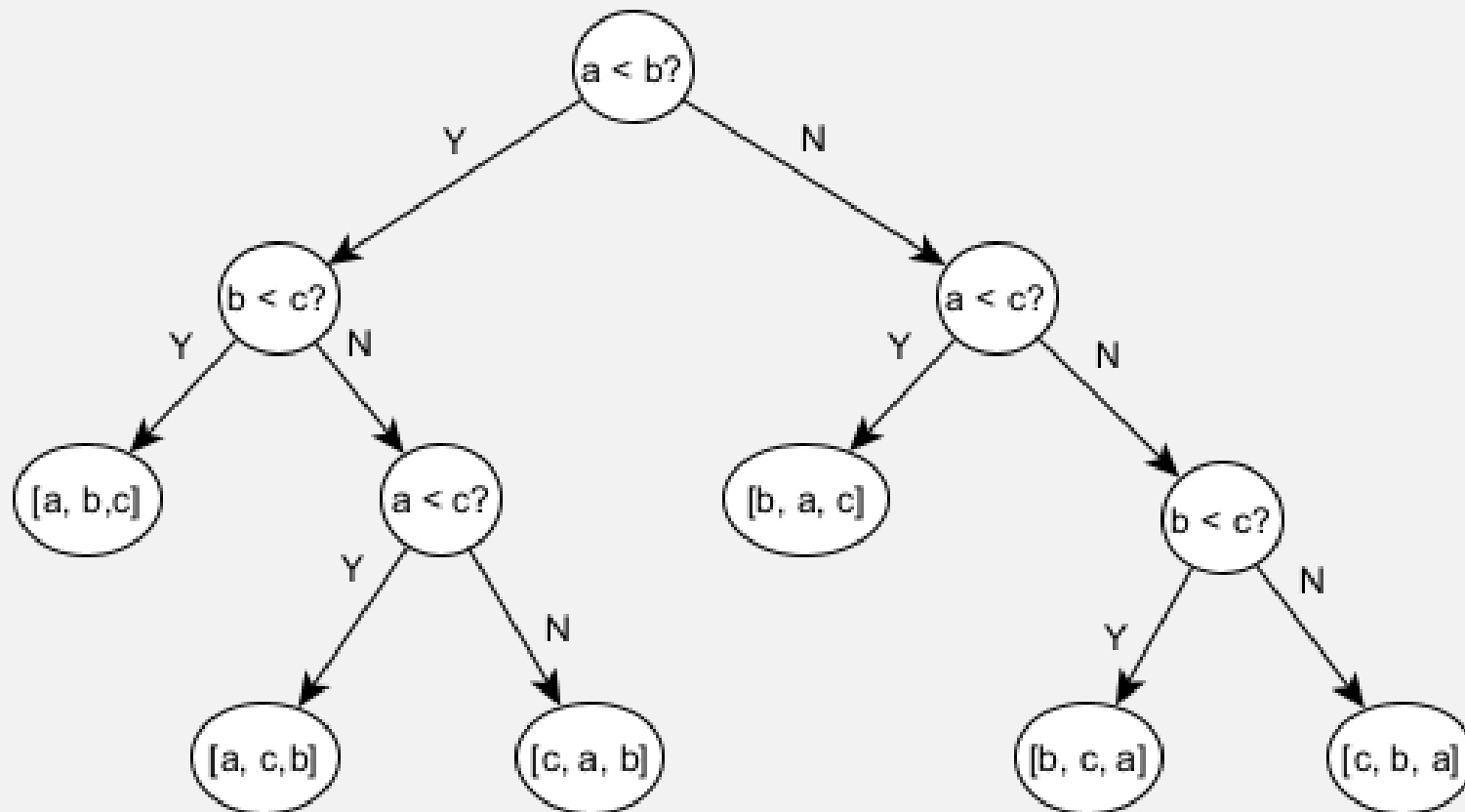


Which one is better?

- Depends on the problem being analyzed.
- **We will use the uniform cost criterion**
- Multiplying two large integers takes 1 unit of time under the uniform cost criterion, but in reality depends on the size of the two integers.

Decision tree model

- Sometimes we are only interested in the number of comparisons we make during a computation.
- **This is so when doing comparison-based sorting or searching**
- Consider a comparison-based sorting algorithm for three values: **a, b, c**



To sort n keys

- Any decision tree must have $n!$ Leaves – corresponding to each of the possible permutations
- Height h of the tree satisfies $h \geq \lg(n!)$

Summary

- Hardware evolution is important, but algorithm engineering is even more important!
- **RAM is a model of computation, abstract, but relevant to the analysis of algorithm.**
- Decision tree model is applicable when performing comparison-based computations.
- Two cost criteria can be applied to RAM programs: **uniform** cost and **logarithmic** cost