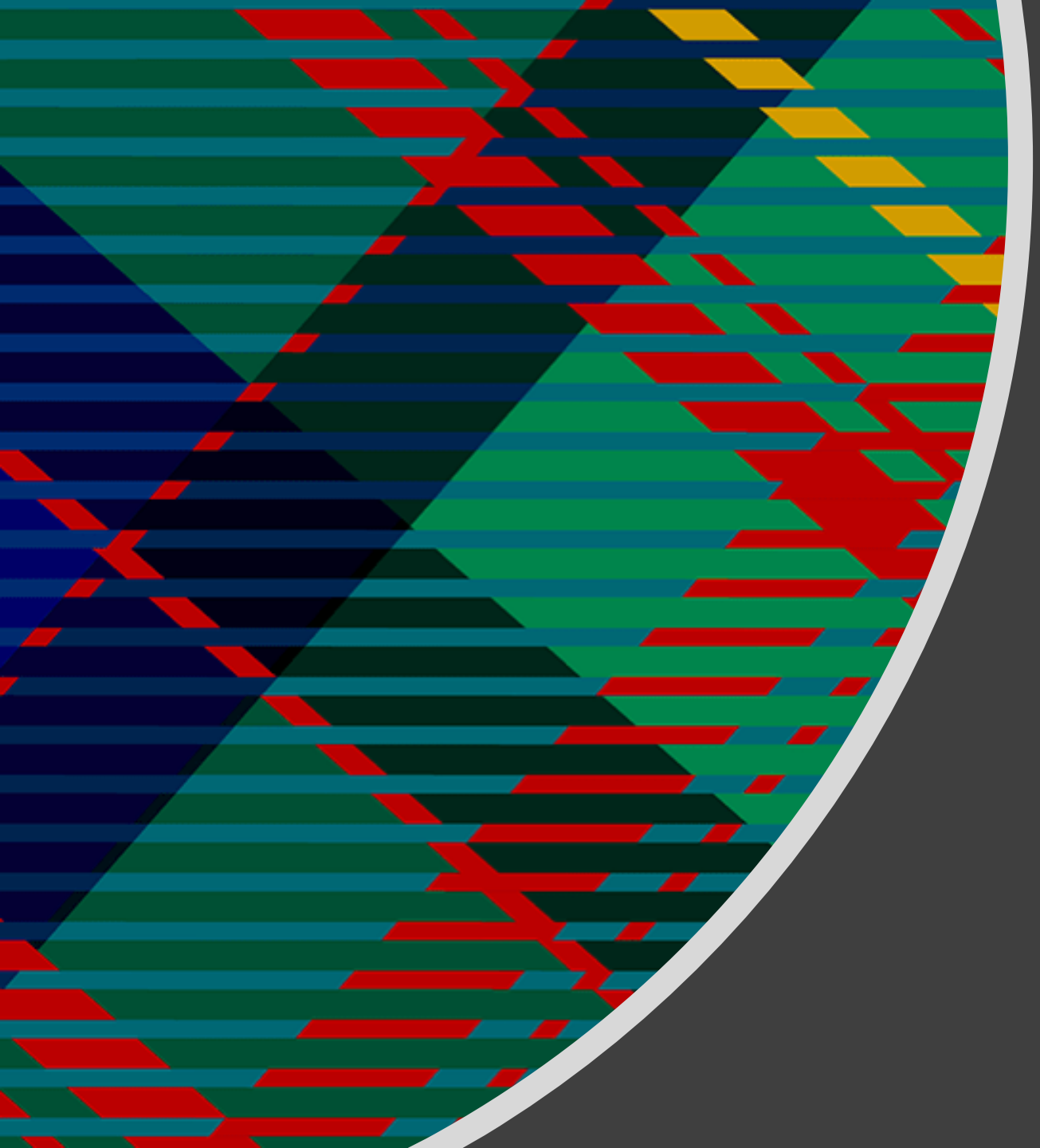


122
Cost and Sort

Last time

Search

- Linear search
- Safety & correctness
- Contract failure/exploits

Today

Cost

- Big ‘O’
- Complexity classes

Searching with order

Sorting

Next

Fast search

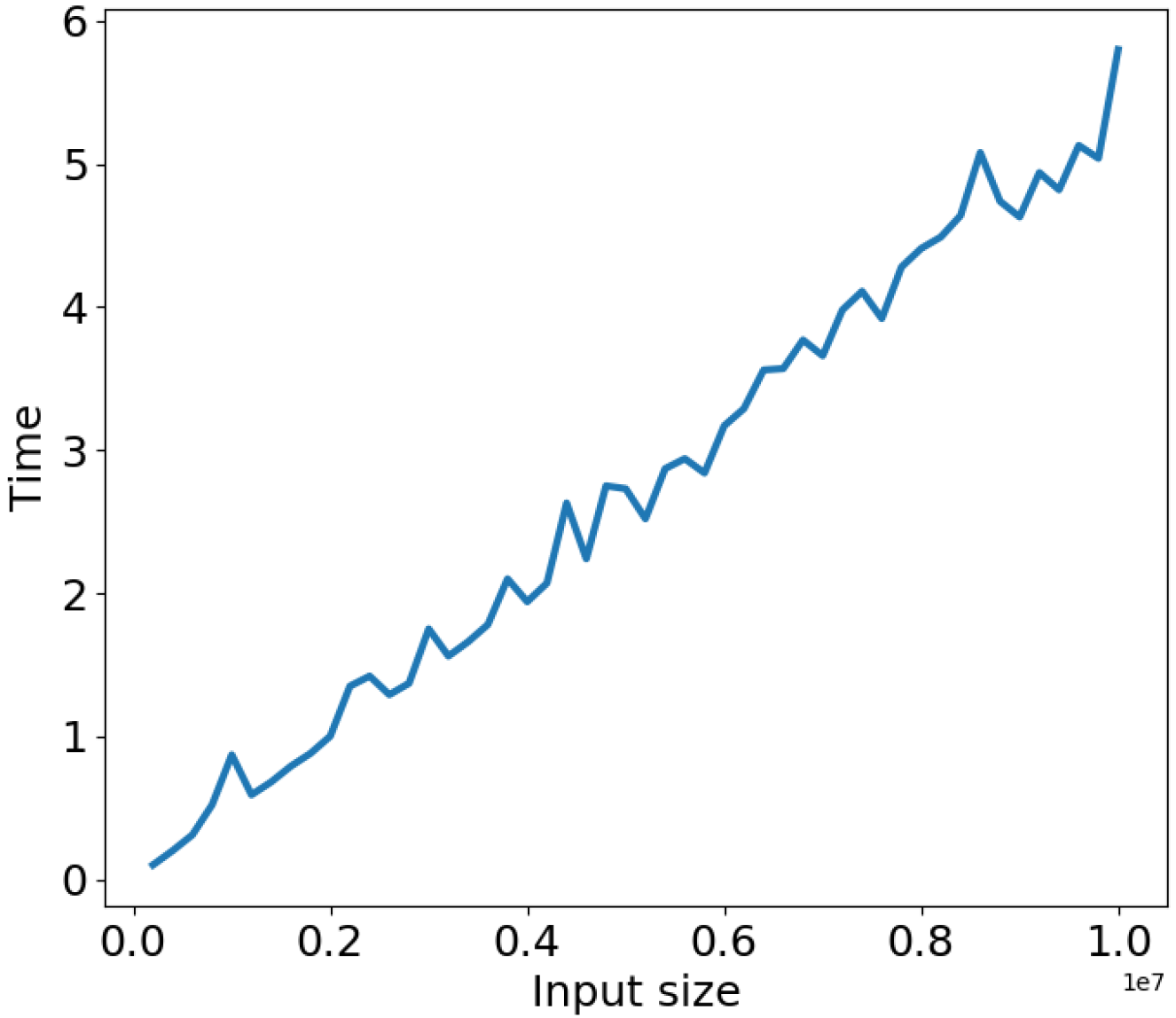
- Binary search

How fast is this code?

```
15  int search(int x, int[] A, int n)
16  {
17    for (int i = 0; i < n; i++)
18    {
19      if (A[i] == x) {
20        return i;
21      }
22    }
23    return -1;
24  }
```

```
$ cc0 arrayutil.c0 search.c0 search-time.c0
$ time ./a.out -r <num runs> -n <num elements>
```

How fast is this code?



Need a better way to measure

Permanent

- Independent of hardware or other running processes, etc.

General

- Applicable to a large class of programs/algorithms/problems
- Resources: time, space

Mathematically rigorous

Useful

- Not for actual run time,
- But help us select best algorithm for the task

How many statements are executed?

```
15  int search(int x, int[] A, int n)
16  {
17    for (int i = 0; i < n; i++)
18    {
19      if (A[i] == x) {
20        return i;
21      }
22    }
23    return -1;
24  }
```

If x is not in A...
how times are these statements executed?

1	`i = 0`
n+1	`i < n`
n	`if (A[i] == x)`
n	`i++`
1	`return -1`

R

$R(n) = 3n + 3$

How many operations are executed?

```
15  int search(int x, int[] A, int n)
16  {
17    for (int i = 0; i < n; i++)
18    {
19      if (A[i] == x) {
20        return i;
21      }
22    }
23    return -1;
24  }
```

$R_1(n) = 3n + 3$

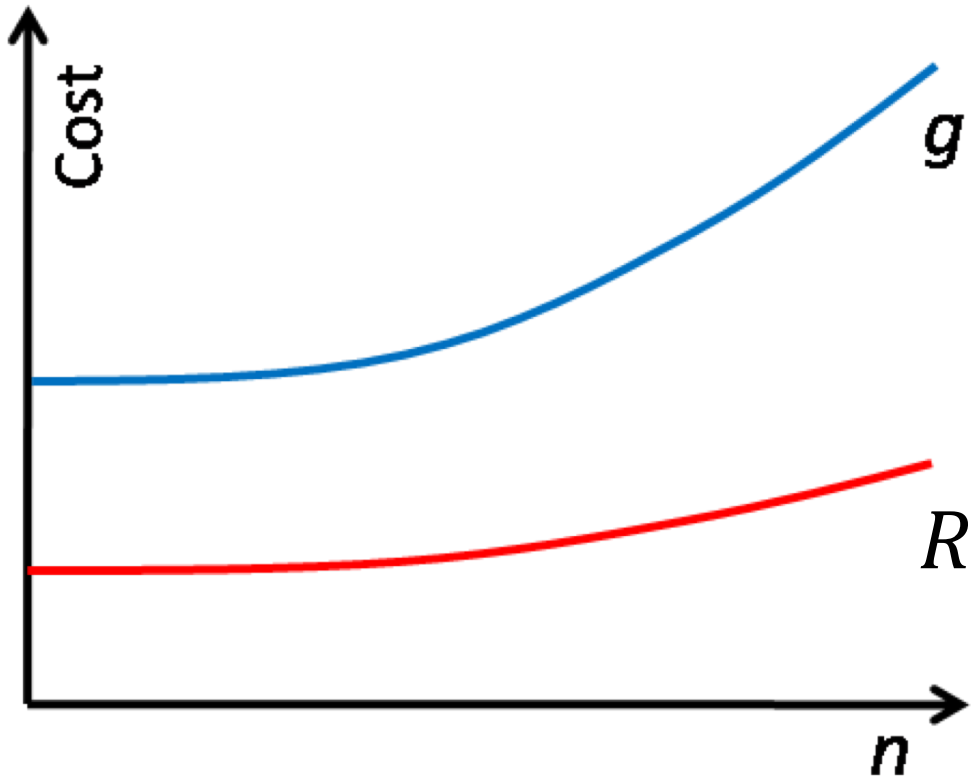
$R_2(n) = 6n + 3$

If x is not in A...
how times **operations** are executed?

1	`i = 0`
n+1	`i < n`
3 n	`if (A[i] == x)`
2 n	`i++` (`i = i + 1`)
1	`return -1`

Comparing functions of n

Which is better?



R is better than g

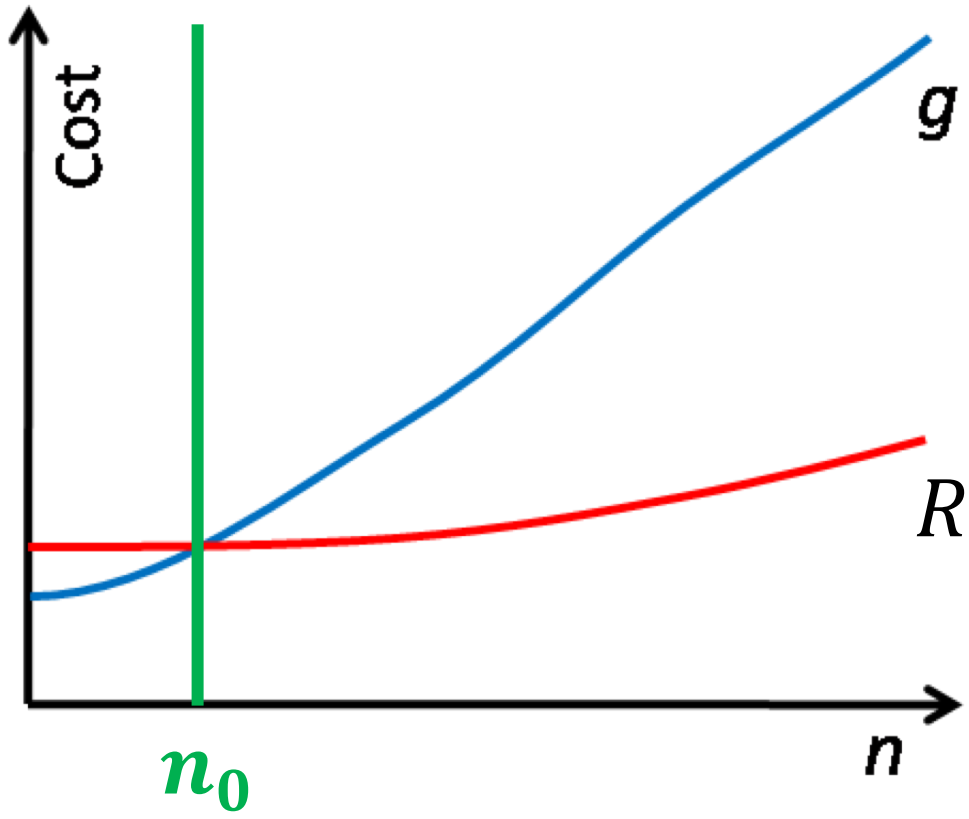
if

$$R(n) \leq g(n)$$

for all n

Comparing functions of n

Which is better?



R is better than g

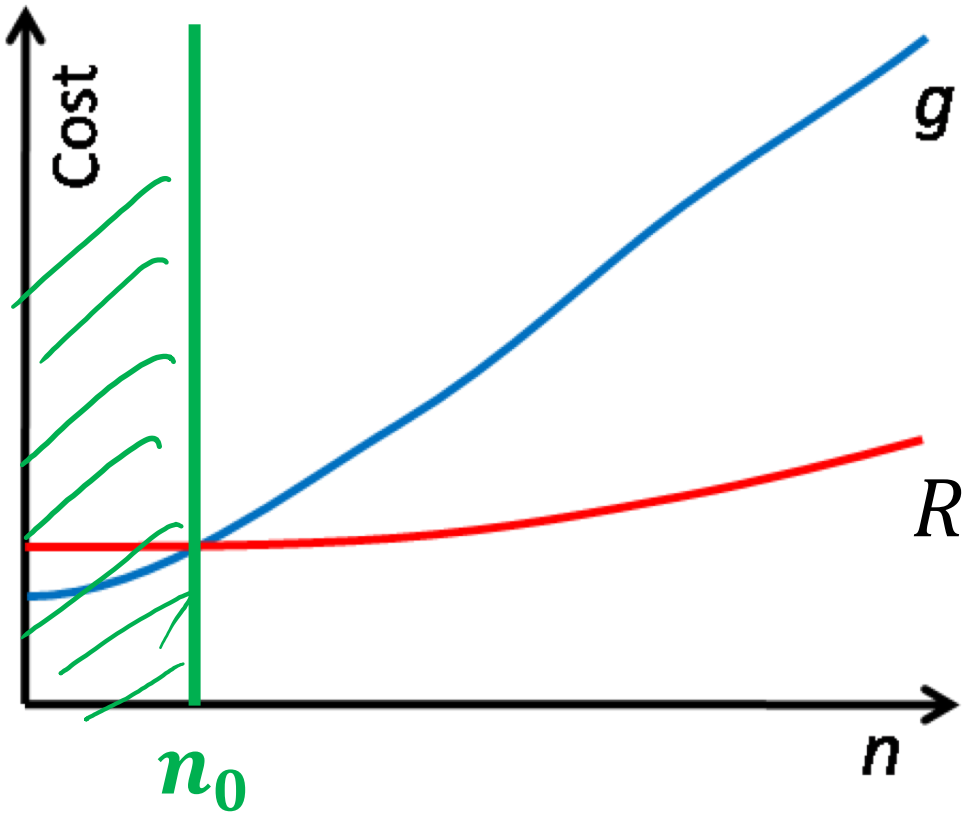
if

$$R(n) \leq g(n)$$

for all n

Comparing functions of n

Which is better?



R is better than g

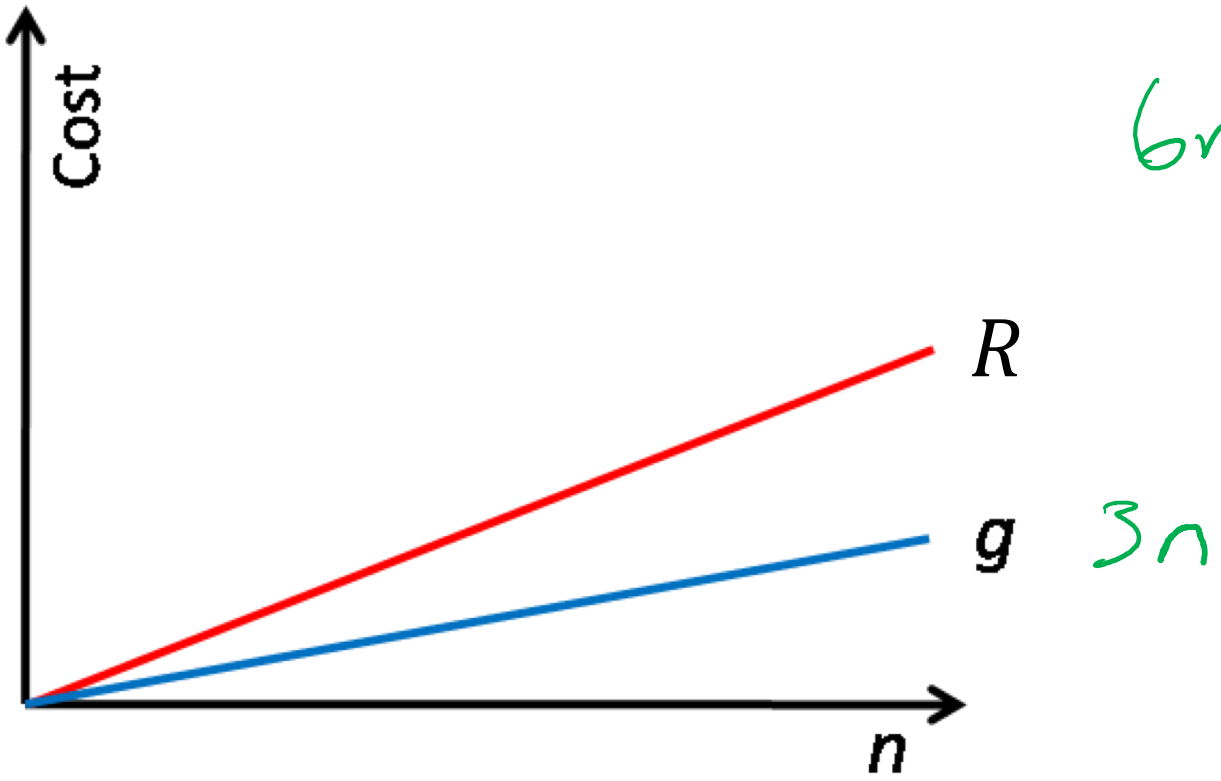
if **there exists n_0, such that**

$$R(n) \leq g(n)$$

for all $n \geq n_0$

Comparing functions of n

Which is better?



6n

3n

R is better than g

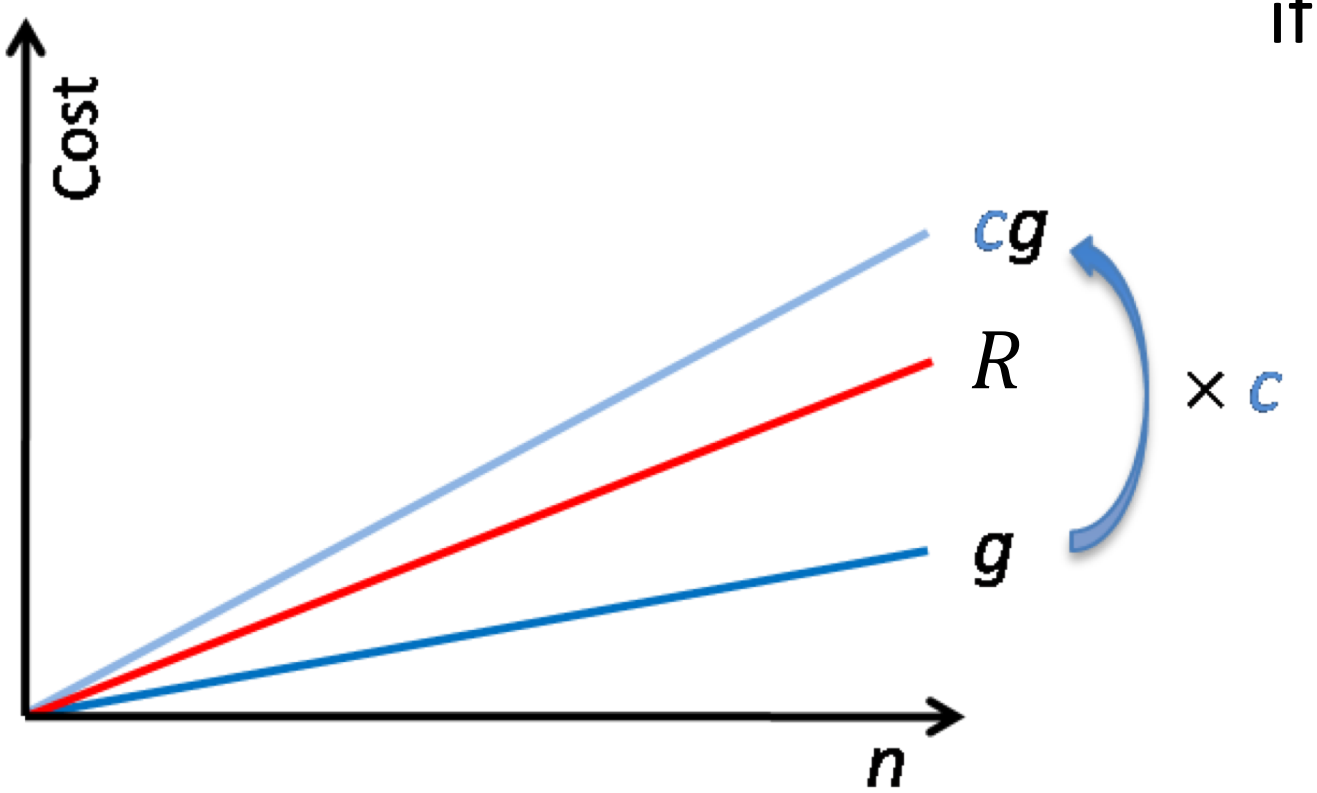
if there exists n_0, such that

$$R(n) \leq g(n)$$

for all $n \geq n_0$

Comparing functions of n

Which is better?



R is better than g

if there exists n_0, such that

$$R(n) \leq g(n)$$

for all $n \geq n_0$

Comparing functions of n

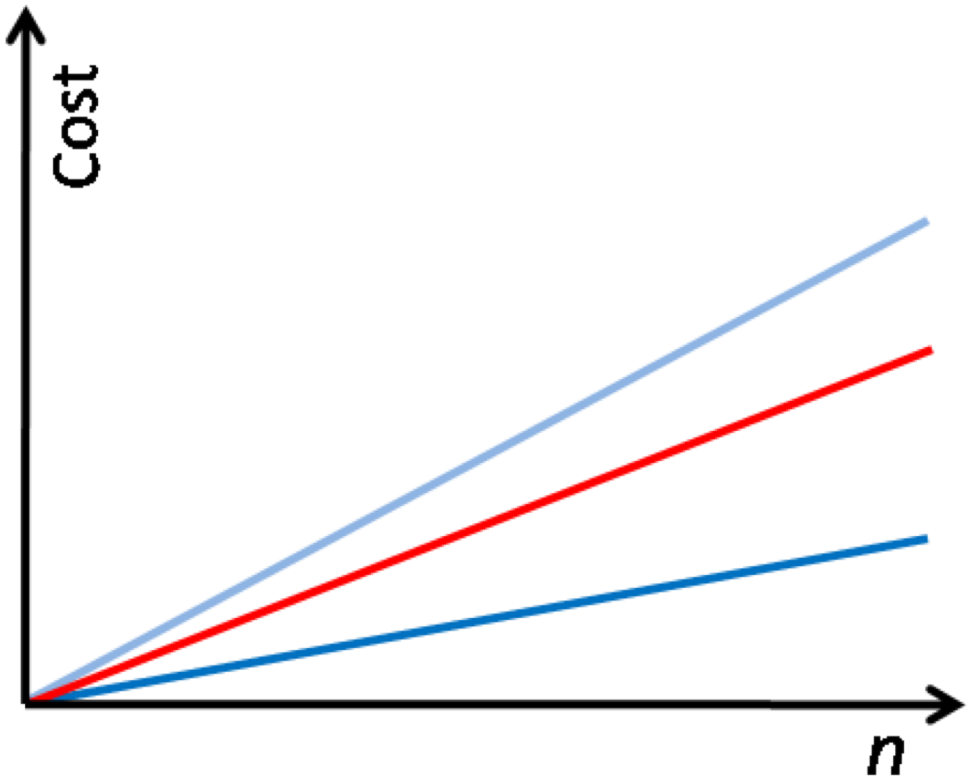
Which is better?

R is better than g

if there exists n_0 **and** c, s.t.

$$R(n) \leq cg(n)$$

for all $n \geq n_0$



$\times c = 4$

$g(n) = 3n + 3$

$cg(n) = 4\ g(n) = 12n + 12$

A set of better functions

R is better than g

if there exists n_0 and c, s.t.

$$R(n) \leq cg(n)$$

for all $n \geq n_0$

A set of better functions

The set of all functions R where

there exists n_0 and c, s.t.

$$R(n) \leq cg(n)$$

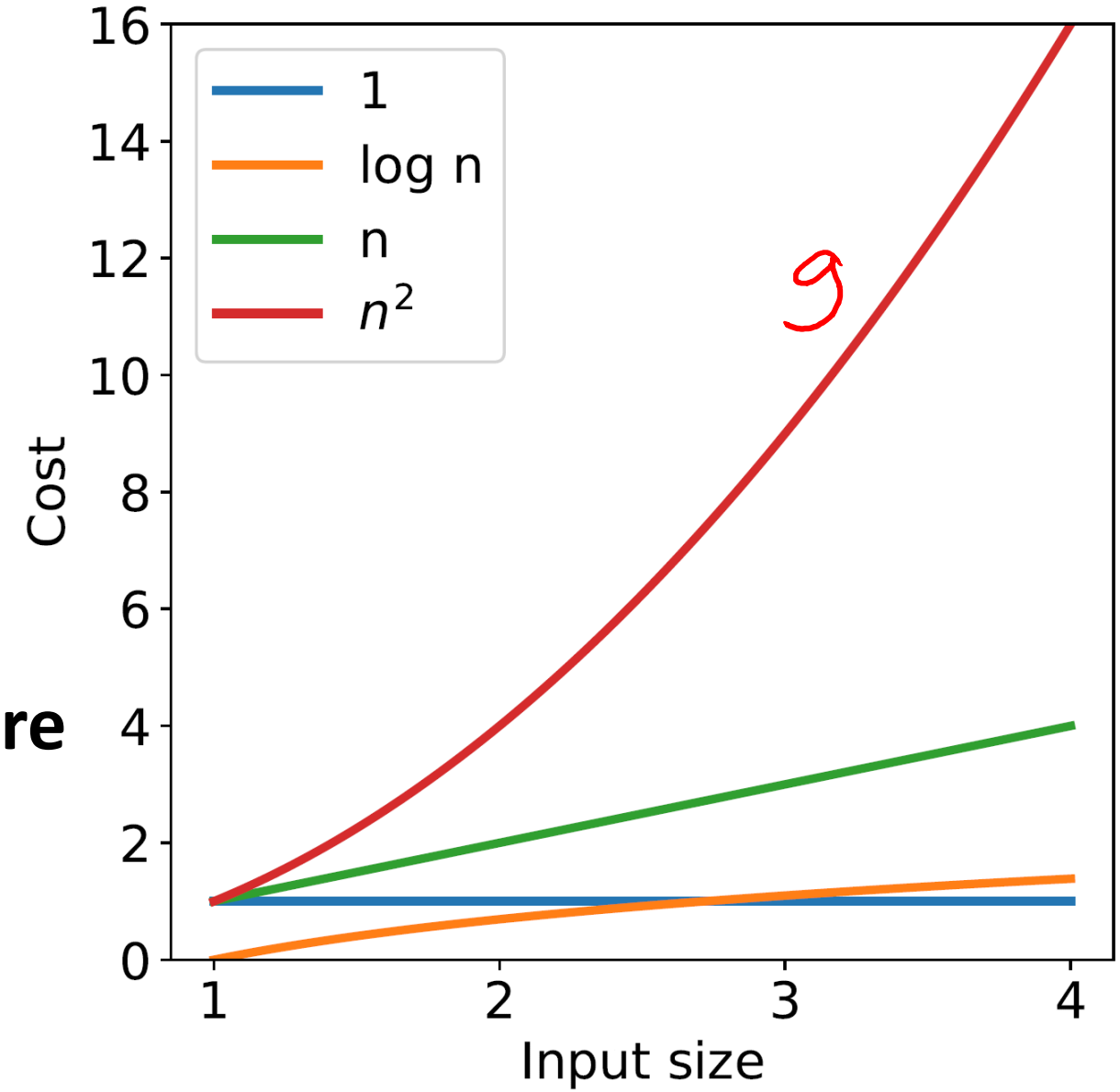
for all $n \geq n_0$

e.g. The set of all functions R where

there exists n_0 and c, s.t.

$$R(n) \leq c\boldsymbol{n^2} \leftarrow g(n)$$

for all $n \geq n_0$



A set of better functions

The set of all functions R where

there exists n_0 and c, s.t.

$$R(n) \leq cg(n)$$

for all $n \geq n_0$

} Definition of Big O of g

A set of better functions

Definition of Big O of g

The set of all functions R where there exists n_0 and c, s.t.

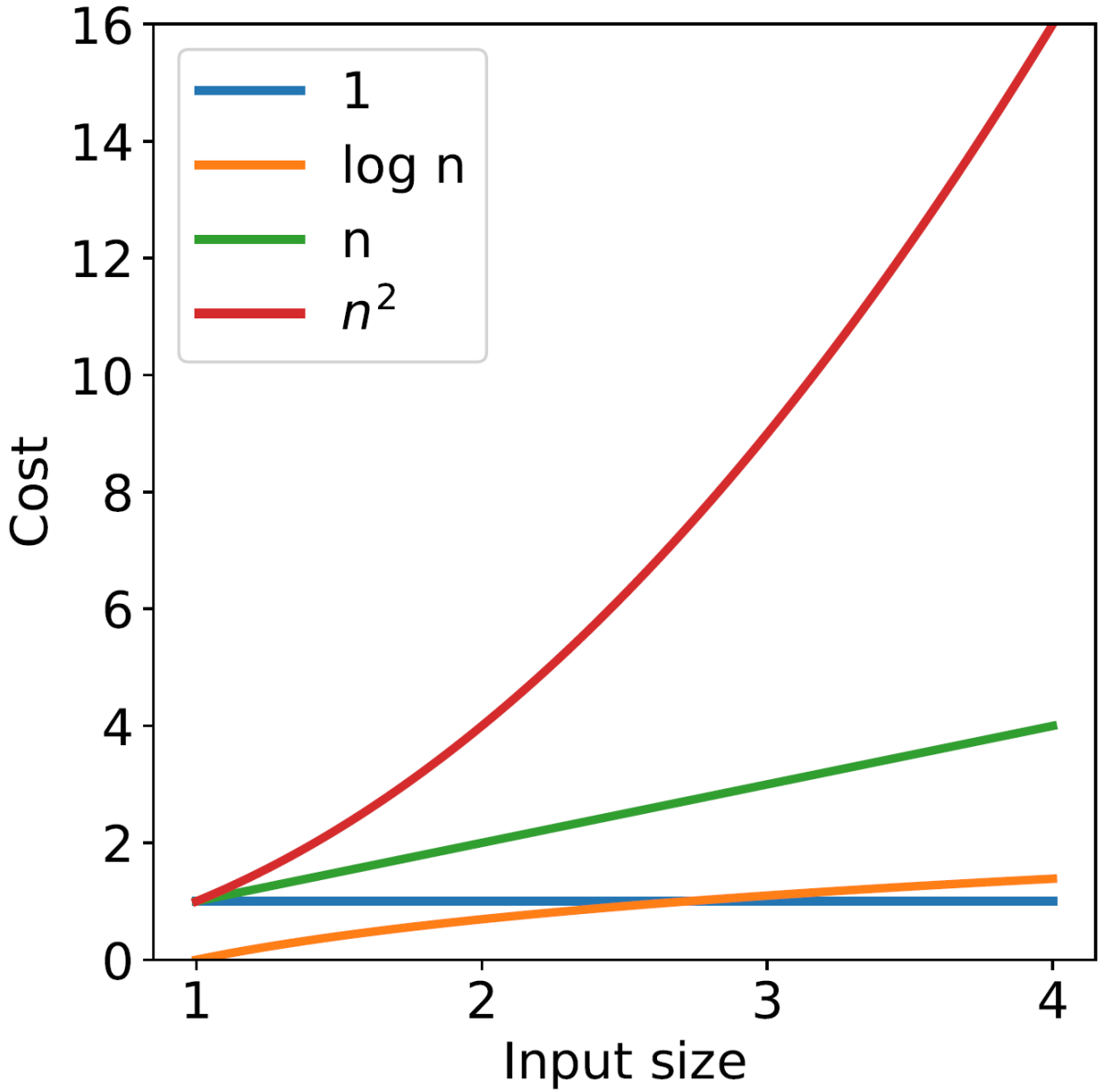
$$R(n) \leq cg(n)$$

for all $n \geq n_0$

e.g. $O(n^2)$ is a set that includes:

$R_1(n) = 1$ $\quad$ $R_3(n) = n$

$R_2(n) = \log n$ $\quad$ $R_4(n) = n^2$



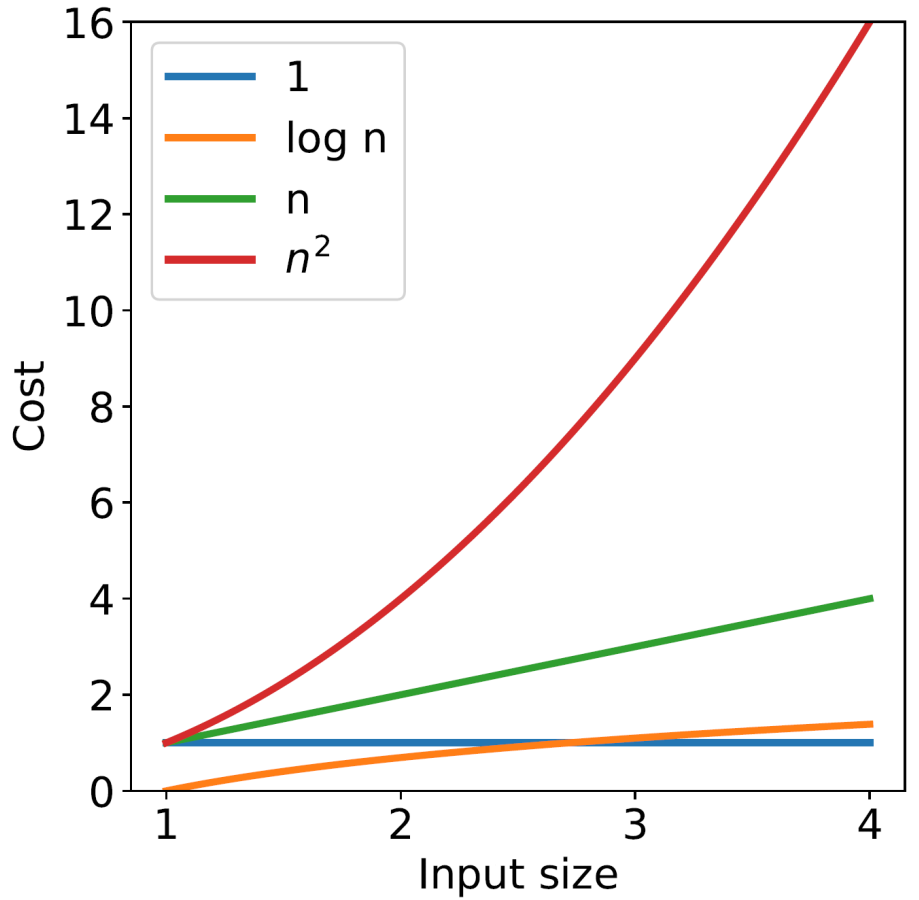
A set of better functions

Definition of Big O of g

The set of all functions R where there exists n_0 and c, s.t.

$$R(n) \leq cg(n)$$

for all $n \geq n_0$



e.g. $O(n^2)$ is a set that includes:

$R_1(n) = 1$ $\quad$ $R_3(n) = n$

$R_2(n) = \log n$ $\quad$ $R_4(n) = n^2$

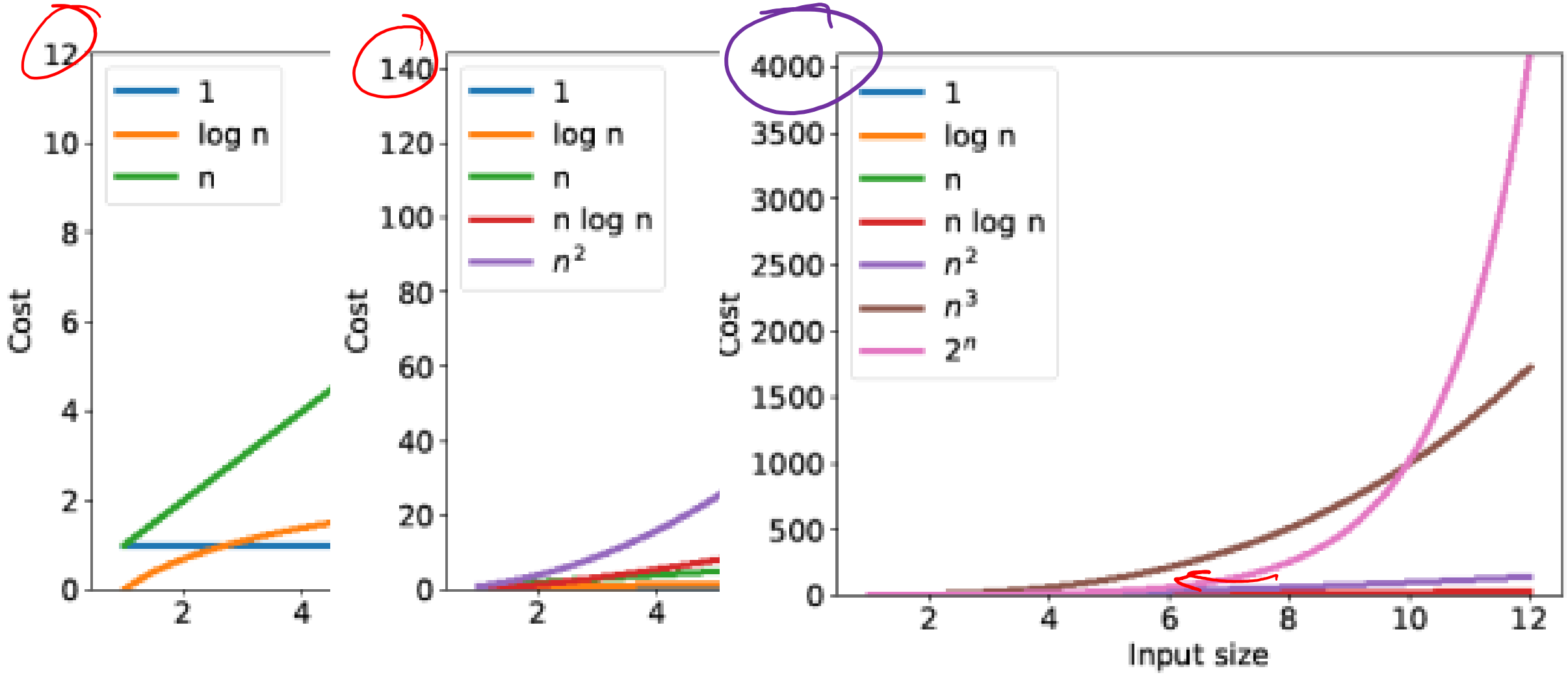
$R_1 \in O(n^2)$

$R_2 \in O(n^2)$

$R_3 \in O(n^2)$

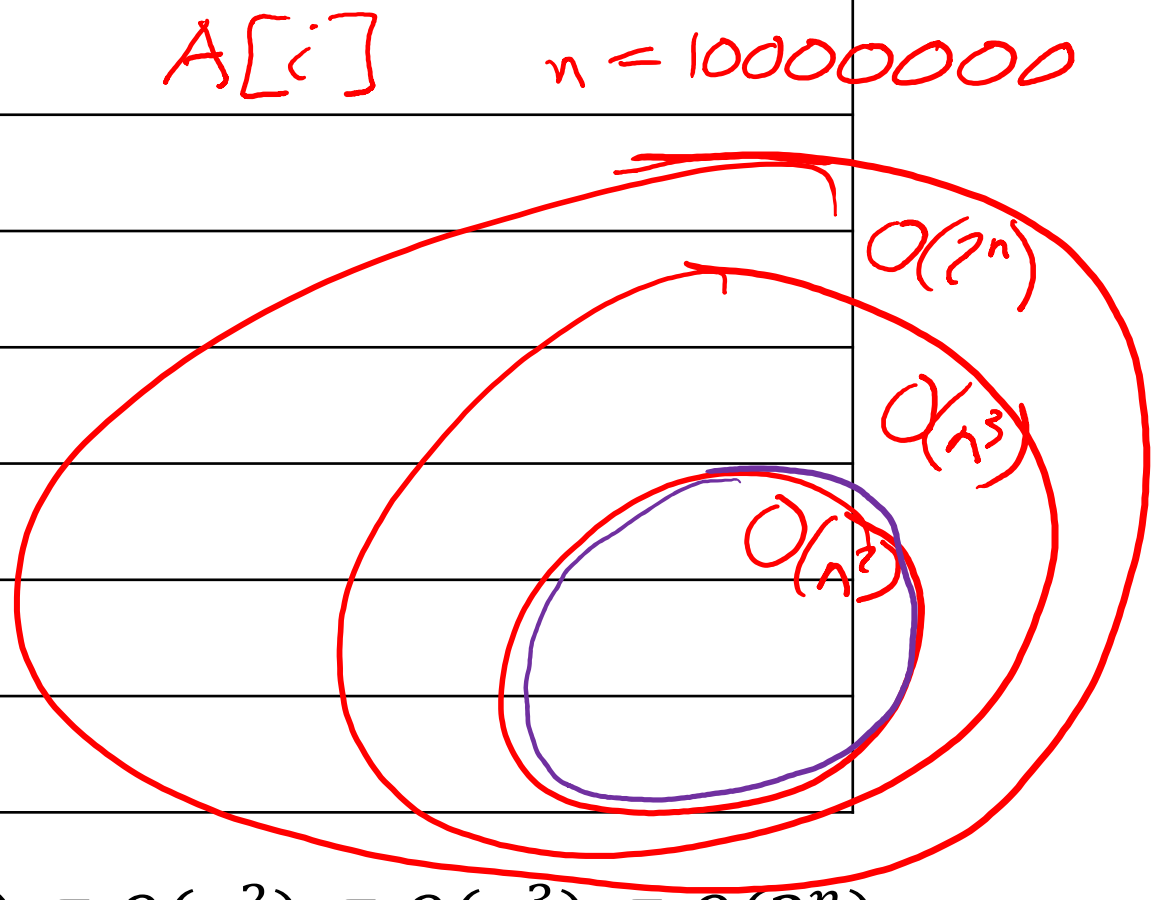
$R_4 \in O(n^2)$

Complexity Classes



Complexity Classes

Complexity class	**Conventional name**
$O(1)$	Constant
$O(\log n)$	Logarithmic
$O(n)$	Linear
$O(n \log n)$	*"n log n"*
$O(n^2)$	Quadratic
$O(n^3)$	Cubic
$O(2^n)$	Exponential



$$O(1) \subseteq O(\log n) \subseteq O(n) \subseteq O(n \log n) \subseteq O(n^2) \subseteq O(n^3) \subseteq O(2^n)$$

Complexity Classes

Can determine whether or not a problem can be solved at all!

	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

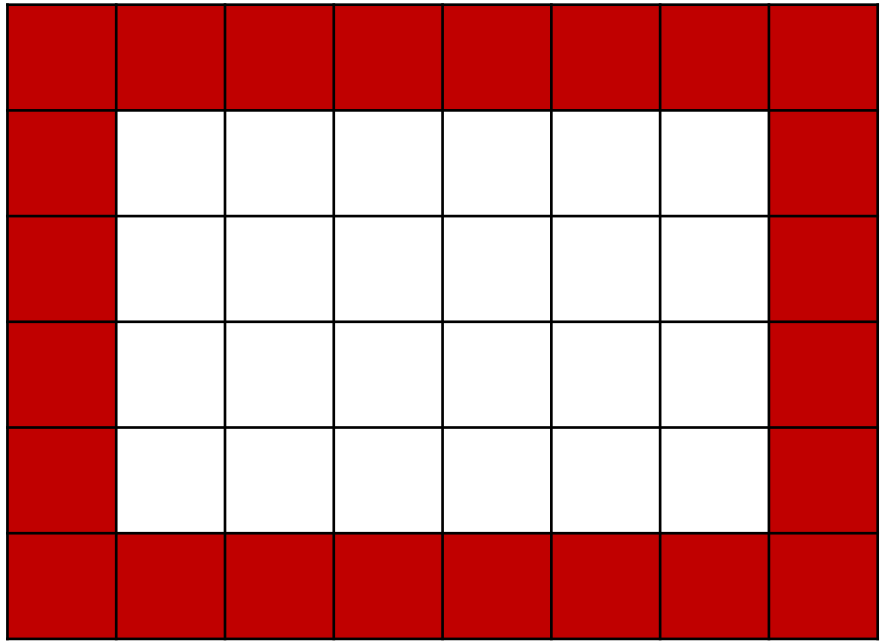
Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

Credit: Algorithm Design, Tardos, Kleinberg (table), UC Berkeley CS 61B, Hug (slide)

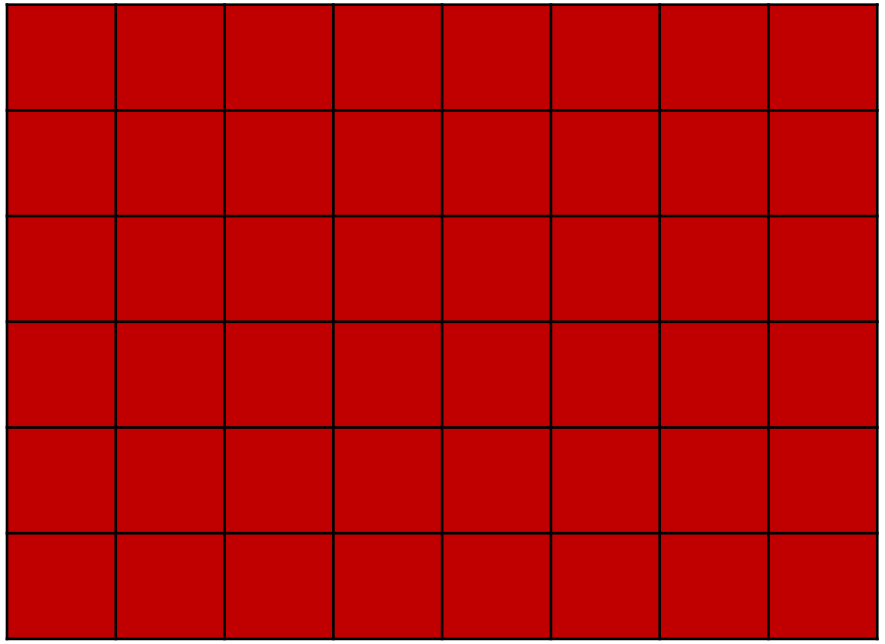
Input

Nothing special about variable n

$O(w + h)$



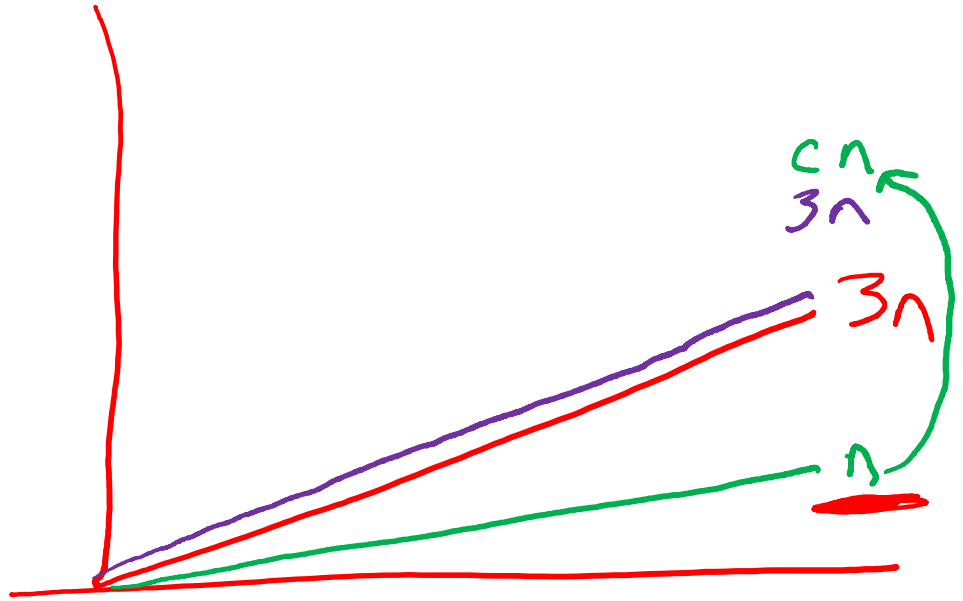
$O(wh)$



How fast is this code?

```
15  int search(int x, int[] A, int n)
16  {
17    for (int i = 0; i < n; i++)
18    {
19      if (A[i] == x) {
20        return i;
21      }
22    }
23    return -1;
24  }
```

O(1) O(1) O(1) O(1) O(1) O(1)



→ n

```
O(1)  i = 0
O(1)  i < n
O(1)  if (A[i] == x)
O(1)  i++
O(1)  return -1
```

$R(n) \in O(n)$

$R(n) = 3n + 2$

Computing complexity

Can we do better if it is sorted?

```
15  int search(int x, int[] A, int n)
16  {
17    for (int i = 0; i < n; i++)
18    {
19      if (A[i] == x) {
20        return i;
21      }
22    }
23    return -1;
24  }
```

```
i = 0

i < n

if (A[i] == x)

i++

return -1
```

Can we do better if it is sorted?

```
15  int search_sorted(int x, int[] A, int n)
16  {
17    for (int i = 0; i < n; i++)
18    {
19      if (A[i] == x) {
20        return i;
21      }
22      if (x < A[i]) {
23        return -1;
24      }
25    }
26    return -1;
27  }
```

O(n)

```
i = 0
i < n          O(1)
if (A[i] == x) O(1)
if (x < A[i])  O(1)
i++            O(1)
                   } n
return -1
```

4n + 2

Sorting

Sorting

Selection sort

```
 6 int find_min(int[] A, int lo, int hi)
...
...
23 void sort(int[] A, int lo, int hi)
24 {
25   for (int i = lo; i < hi; i++)
26   {
27     int m = find_min(A, i, hi);
28     swap(A, i, m);
29   }
30 }
```

What do we know before iteration `i`?

Selection sort

```
 6 int find_min(int[] A, int lo, int hi)
...
...
23 void sort(int[] A, int lo, int hi)
24 {
25   for (int i = lo; i < hi; i++)
26   {
27     int m = find_min(A, i, hi);
28     swap(A, i, m);
29   }
30 }
```

What do we know before iteration `i`?

Selection sort

```
 6 int find_min(int[] A, int lo, int hi)
...
...
23 void sort(int[] A, int lo, int hi)
24 {
25   for (int i = lo; i < hi; i++)
26   //@loop_invariant lo <= i && i <= hi;
27   //@loop_invariant is_sorted(A, lo, i);
28   //@loop_invariant le_segs(A, lo, i, A, i, hi);
29   {
30     int m = find_min(A, i, hi);
31     swap(A, i, m);
32   }
33 }
```

```
 6 int find_min(int[] A, int lo, int hi)
 7 //@requires 0 <= lo && lo < hi && hi <= \length(A);
 8 //@ensures lo <= \result && \result < hi;
 9 //@ensures le_seg(A[\result], A, lo, hi);
...
...
23 void sort(int[] A, int lo, int hi)
24 //@requires 0 <= lo && lo <= hi && hi <= \length(A);
25 //@ensures is_sorted(A, lo, hi);
26 {
27   for (int i = lo; i < hi; i++)
28   //@loop_invariant lo <= i && i <= hi;
29   //@loop_invariant is_sorted(A, lo, i);
30   //@loop_invariant le_segs(A, lo, i, A, i, hi);
31   {
32     int m = find_min(A, i, hi);
33     swap(A, i, m);
34   }
35 }
```

INIT:

PRES:

- lo <= i && i <= hi
- is_sorted(A, lo, hi)
  - i' = i + 1
  - A[lo, i) <= A[i, hi)
    by LI 3
  - is_sorted(A, lo, i+1)

```
 6 int find_min(int[] A, int lo, int hi)
 7 //@requires 0 <= lo && lo < hi && hi <= \length(A);
 8 //@ensures lo <= \result && \result < hi;
 9 //@ensures le_seg(A[\result], A, lo, hi);
...
...
23 void sort(int[] A, int lo, int hi)
24 //@requires 0 <= lo && lo <= hi && hi <= \length(A);
25 //@ensures is_sorted(A, lo, hi);
26 {
27   for (int i = lo; i < hi; i++)
28   //@loop_invariant lo <= i && i <= hi;
29   //@loop_invariant is_sorted(A, lo, i);
30   //@loop_invariant le_segs(A, lo, i, A, i, hi);
31   {
32     int m = find_min(A, i, hi);
33     swap(A, i, m);
34   }
35 }
```

INIT:

PRES:

- lo <= i && i <= hi
- is_sorted(A, lo, hi)
- A[lo, i) <= A[i, hi)
  - At end of loop, A[i] < A[i+1, hi) from line 8
  - A[lo, i+1) is A[lo, i) then A[i]

```
 6 int find_min(int[] A, int lo, int hi)
 7 //@requires 0 <= lo && lo < hi && hi <= \length(A);
 8 //@ensures lo <= \result && \result < hi;
 9 //@ensures le_seg(A[\result], A, lo, hi);
...
...
23 void sort(int[] A, int lo, int hi)
24 //@requires 0 <= lo && lo <= hi && hi <= \length(A);
25 //@ensures is_sorted(A, lo, hi);
26 {
27   for (int i = lo; i < hi; i++)
28   //@loop_invariant lo <= i && i <= hi;
29   //@loop_invariant is_sorted(A, lo, i);
30   //@loop_invariant le_segs(A, lo, i, A, i, hi);
31   {
32     int m = find_min(A, i, hi);
33     swap(A, i, m);
34   }
35 }
```

INIT:

PRES:

- lo <= i && i <= hi
- is_sorted(A, lo, hi)
- A[lo, i) <= A[i, hi)

EXIT:

- is_sorted(A, lo, i)
  by LI 2 on line 29
- i == hi
- is_sorted(A, lo, hi)

Cost / Complexity

```
13 int find_min(int[] A, int lo, int hi)
14 {
15   int min_i = lo;
16   for (int i = lo+1; i < hi; i++) {
17     if (A[i] < A[min_i])
18       min_i = i;
19   }
20   return min_i;
21 }
22
23 void sort(int[] A, int lo, int hi)
24 {
25   for (int i = lo; i < hi; i++)
26   {
27     int m = find_min(A, i, hi);
28     swap(A, i, m);
29   }
30 }
```

$\in O(n)$

$O(1)$
$O(n)$
$O(1)$
} n

$O(n^2)$