Week 2: Monday

EECS 281

MAXIM ALEKSA

maximal.io

Agenda

Asymptotics

Recurrence relations

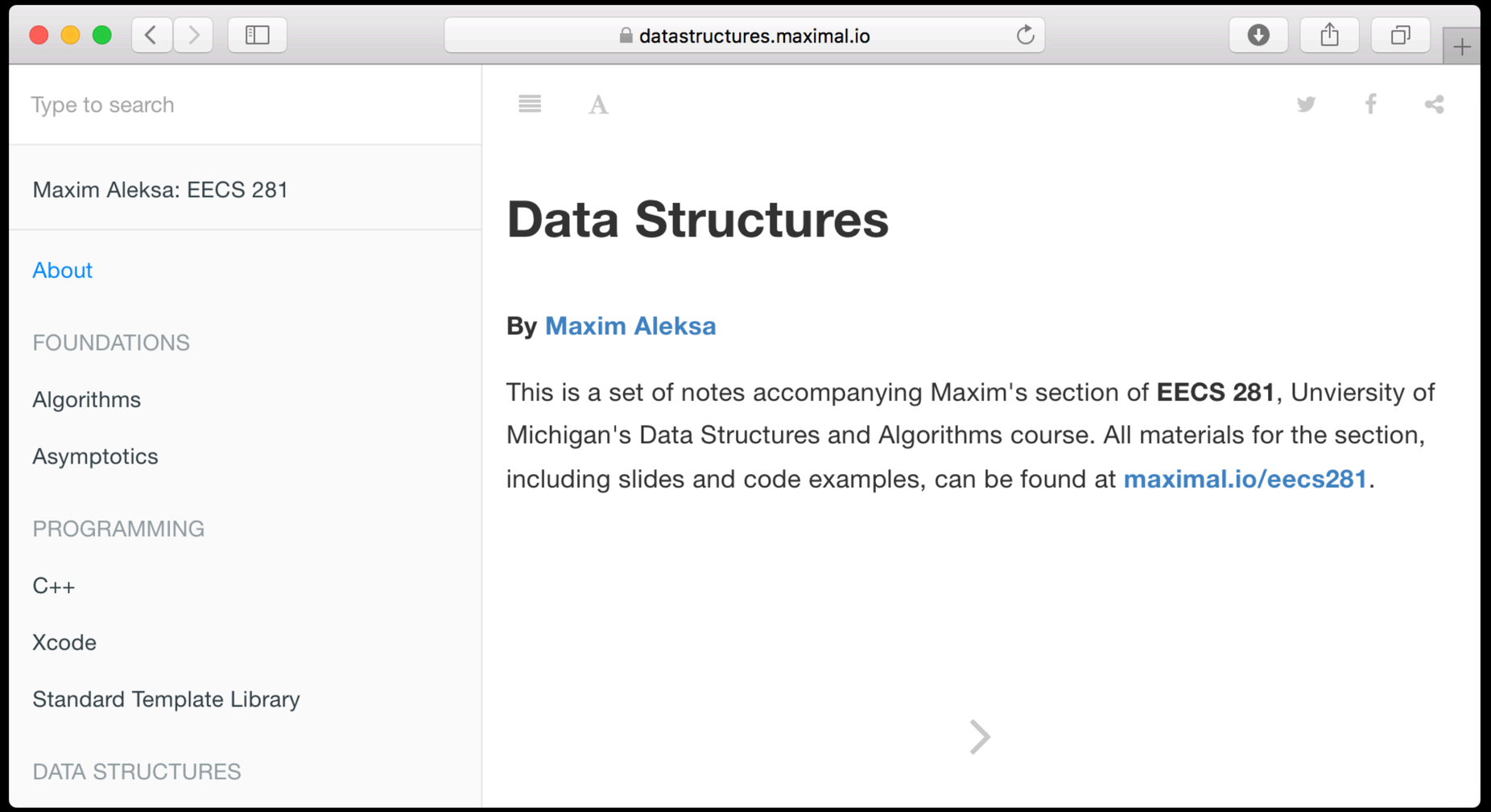
Case study: sorting

Lab 2 assignment

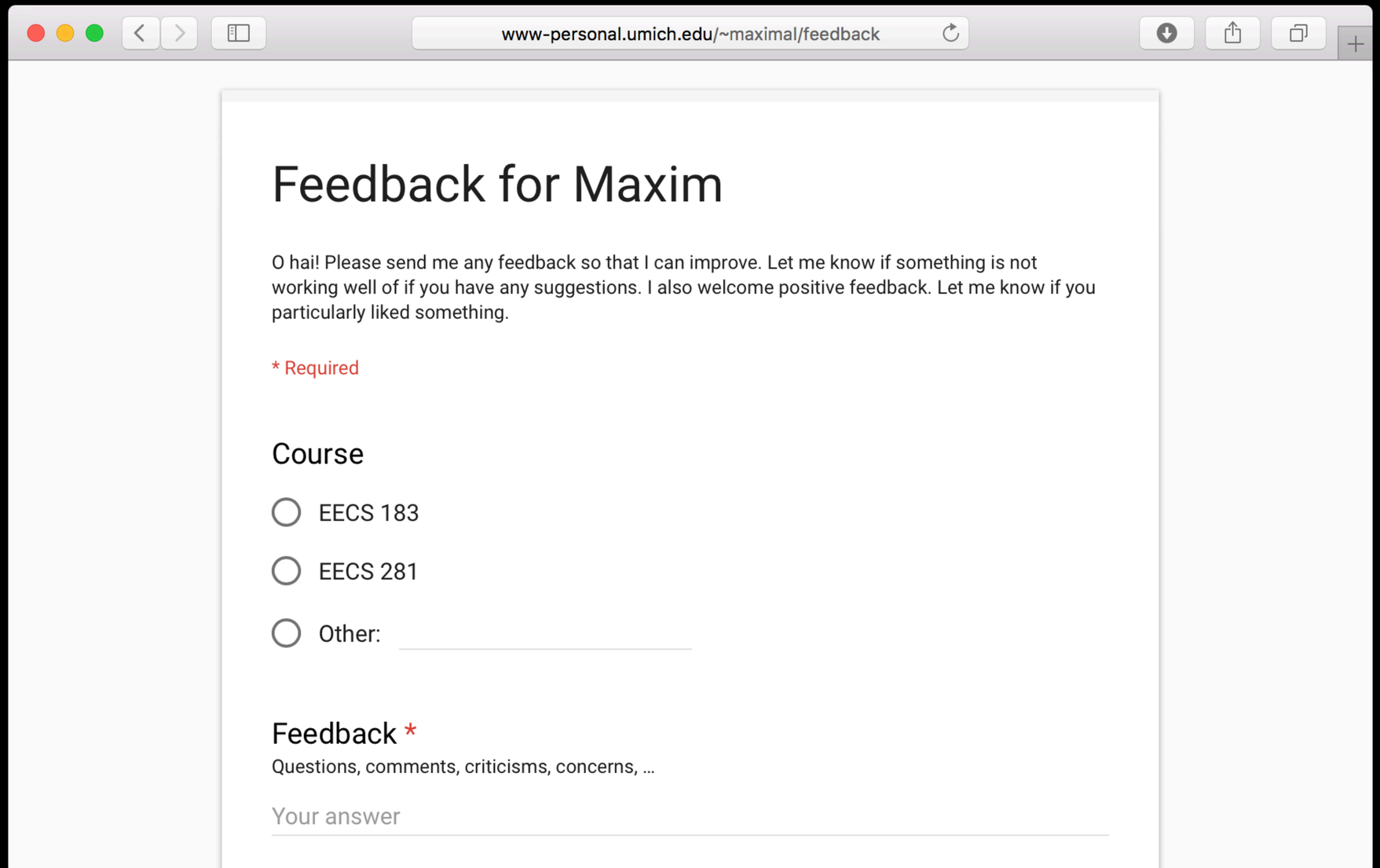
`maximal.io/eecs281`

slides, notes, source code, links, et cetera

datastructures.maximal.io



maximal.io/feedback



Boyer–Moore majority vote algorithm

```
/**
 * Changes most to be a potential majority element.
 * Returns true if most is a majority element,
 * otherwise returns false.
 * The .size() of the data vector must be at least 1.
 */
template <typename T>
bool mooresVoting(const vector<T> &data, T &most) {
    // TODO
}
```

Binomial coefficient

Implement **`binomial`**, a function that computes the Binomial coefficient of **`n`** and **`k`**.

```
uint64_t binomial(int n, int k);
```

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Binomial coefficient

Implement **`binomial`**, a function that computes the Binomial coefficient of **`n`** and **`k`**.

```
uint64_t binomial(int n, int k);
```

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Asymptotics

Asymptotic complexity

	Big O $O(f(n))$	Big Omega $\Omega(f(n))$	Big Theta $\Theta(f(n))$
Informal meaning	Order of growth is less than or equal to $f(n)$	Order of growth is greater than or equal to $f(n)$	Order of growth is $f(n)$
Example family	$O(N^2)$	$\Omega(N^2)$	$\Theta(N^2)$
Family members	$N^2 / 2$ $2N^2 + 1$ $\log(N)$	$N^2 / 2$ $2N^2 + 1$ e^N	$N^2 / 2$ $2N^2 + 1$ $N^2 + 183N + 5$

Growth complexity

Order from most to least efficient:

$O(n)$ $O(n!)$ $O(n^n)$ $O(n^2)$ $O(2^n)$

$O(\log n)$ $O(1)$ $O(\text{sqrt}(n))$ $O(n^3)$

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

Growth complexity

Order from most to least efficient:

$$O(1) \subset O(\log n) \subset O(\text{sqrt}(n)) \subset O(n) \subset$$

$$O(n \log n) \subset O(n^2) \subset O(n^2 \log n) \subset$$

$$O(n^3) \subset O(2^n) \subset O(3^n) \subset O(n!) \subset O(n^n)$$

Let $R(N)$ be the runtime of this code as a function of N.

```
bool hasDuplicates(const int numbers[], int size) {
    for (int i = 0; i < size; i += 1) {
        for (int j = i + 1; j < size; j += 1) {
            if (numbers[i] == numbers[j]) {
                return true;
            }
        }
    }
    return false;
}
```

What is the order of growth of R(N)?

A. $R(N) \in \Theta(1)$.

B. $R(N) \in \Theta(N)$.

C. $R(N) \in \Theta(N^2)$.

D. Something else

Let $R(N)$ be the runtime of this code as a function of N.

```
bool hasDuplicates(const int numbers[], int size) {
    for (int i = 0; i < size; i += 1) {
        for (int j = i + 1; j < size; j += 1) {
            if (numbers[i] == numbers[j]) {
                return true;
            }
        }
    }
    return false;
}
```

What is the order of growth of R(N)?

A. $R(N) \in \Theta(1)$.

B. $R(N) \in \Theta(N)$.

C. $R(N) \in \Theta(N^2)$.

✓D. Something else

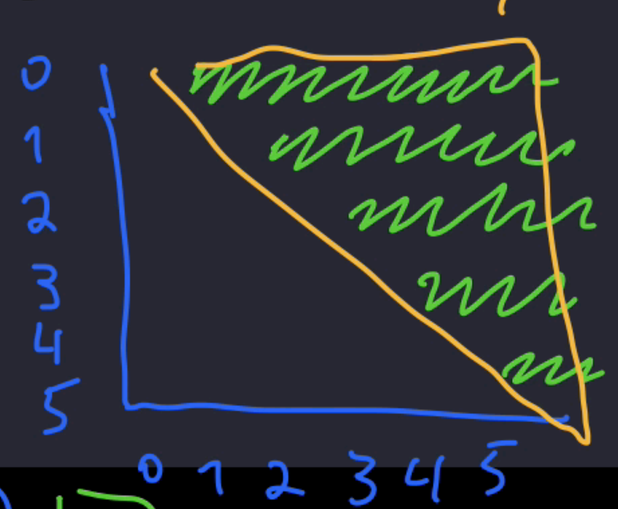
depends on input

Find a simple $f(N)$ such that the runtime $R(N) \in \Theta(f(N))$ in the worst case.

```
bool hasDuplicates(const int numbers[], int size) {
    for (int i = 0; i < size; i += 1) {
        for (int j = i + 1; j < size; j += 1) {
            if (numbers[i] == numbers[j]) {
                return true;
            }
        }
    }
    return false;
}
```

Find a simple $f(N)$ such that the runtime $R(N) \in \Theta(f(N))$ in the worst case.

```
bool hasDuplicates(const int numbers[], int size) {
    for (int i = 0; i < size; i += 1) {
        for (int j = i + 1; j < size; j += 1) {
            if (numbers[i] == numbers[j]) {
                return true;
            }
        }
    }
    return false;
}
```



$\frac{n^2}{2}$

$n + (n-1) + (n-2) + \ldots + 3 + 2 + 1$

$n + 1 + n + 1 + n + 1 + \ldots$

$= \frac{n}{2} \times (n+1)$

$\frac{n^2}{2} + \frac{n}{2}$

Best, average, worst

Best case

Average case

Worst case

Always

Best, average, worst

Sorting

duplicates

Best case

1, 2, 3, 4, 5

already sorted

Average case

1 1 1 1 1 1

all duplicates

Worst case

1 2 3 3 4 5

dups in middles

5 4 3 2 1

reverse sorted

Always

1 2 3 4 5

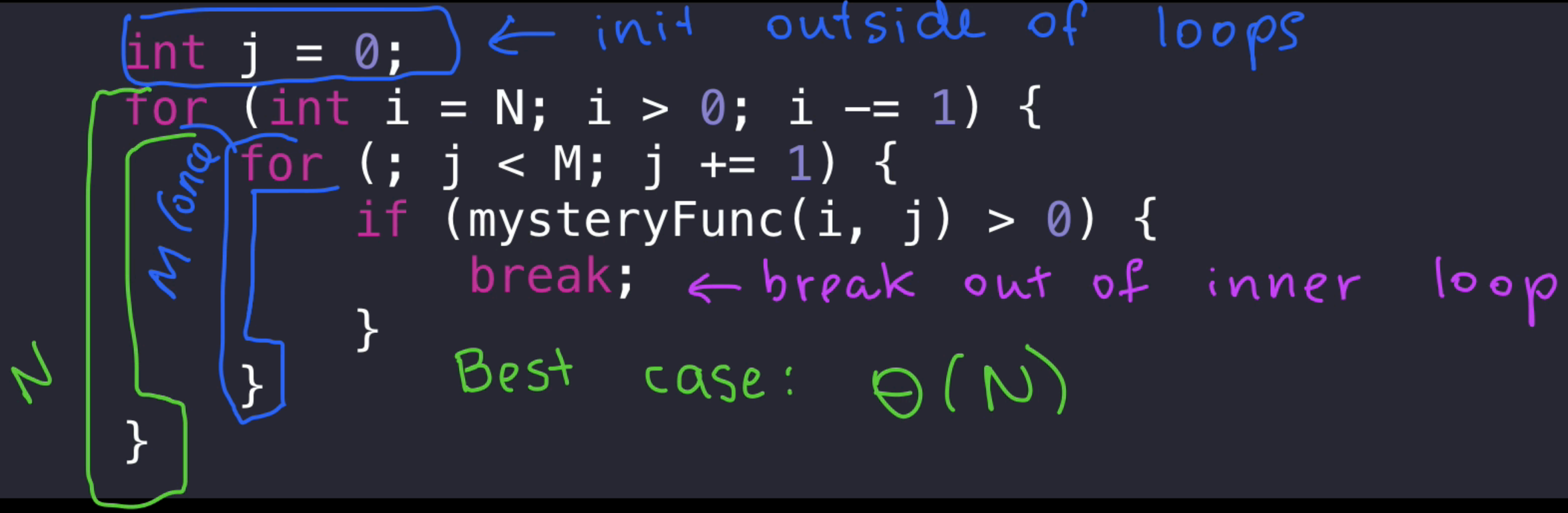
no duplicates

finding sum does not depend on input

Give the worst-case and best-case runtime in terms of *N* and *M*. Assume that **`mysteryFunc`** executes in constant time and returns a value of type **`int`**.

```
int j = 0;
for (int i = N; i > 0; i -= 1) {
    for (; j < M; j += 1) {
        if (mysteryFunc(i, j) > 0) {
            break;
        }
    }
}
```

Give the worst-case and best-case runtime in terms of *N* and *M*. Assume that **`mysteryFunc`** executes in constant time and returns a value of type **`int`**.



Worst case: $\Theta(N+M)$

Find a simple $f(N)$ such that the runtime $R(N) \in \Theta(f(N))$.

```
void printHello(int n) {
    for (int i = 1; i <= n; i *= 2) {
        for (int j = 0; j < i; j += 1) {
            cout << "hello" << endl;
        }
    }
}
```

A. 1

B. $\log n$

C. n

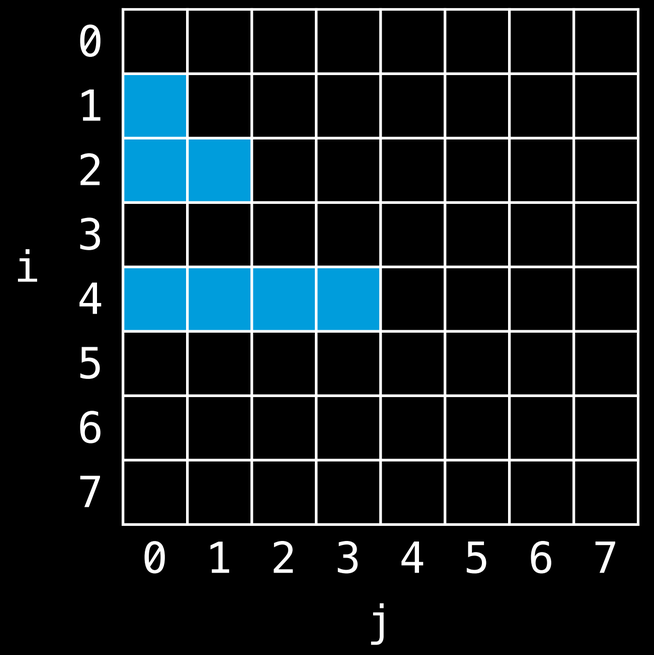
D. $n \log n$

E. n^2

F. Something else

Find a simple $f(N)$ such that the runtime $R(N) \in \Theta(f(N))$.

```
void printHello(int n) {
    for (int i = 1; i <= n; i *= 2) {
        for (int j = 0; j < i; j += 1) {
            cout << "hello" << endl;
        }
    }
}
```



A. 1

B. $\log n$

C. n

D. $n \log n$

E. n^2

F. Something else

Find a simple $f(N)$ such that the runtime $R(N) \in \Theta(f(N))$.

```
void printHello(int n) {
    for (int i = 1; i <= n; i *= 2) {
        for (int j = 0; j < i; j += 1) {
            cout << "hello" << endl;
        }
    }
}
```

A. 1

B. $\log n$

C. n

D. $n \log n$

E. n^2

F. Something else

N	$R(N)$
1	1
4	1 + 2 + 4 = 7
7	1 + 2 + 4 = 7
8	1 + 2 + 4 + 8 = 15
27	1 + 2 + 4 + 8 + 16 = 31
185	… + 64 + 128 = 255
715	… + 256 + 512 = 1023

Find a simple $f(N)$ such that the runtime $R(N) \in \Theta(f(N))$.

```
void printHello(int n) {
    for (int i = 1; i <= n; i *= 2) {
        for (int j = 0; j < i; j += 1) {
            cout << "hello" << endl;
        }
    }
}
```

A. 1
C. n
E. n^2
B. $\log n$
D. $n \log n$
F. Something else

n 2N
1 2
4 8
7 14
8 16
27 54
185 370
715 1430

N	$R(N)$
1	1
4	1 + 2 + 4 = 7
7	1 + 2 + 4 = 7
8	1 + 2 + 4 + 8 = 15
27	1 + 2 + 4 + 8 + 16 = 31
185	… + 64 + 128 = 255
715	… + 256 + 512 = 1023

Things to remember

Sum of first N numbers:

$1 + 2 + 3 + 4 + \ldots + N = N(N + 1) / 2 \in \Theta(N^2)$

Sum of the powers of 2 up to N:

$1 + 2 + 4 + 8 + \ldots + N = 2N - 1 \in \Theta(N)$

Things to remember

Sum of first N numbers:

$1 + 2 + 3 + 4 + \ldots + N = N(N+1) / 2 \in \Theta(N^2)$

Sum of first N powers of 2: up to N:

$1 + 2 + 4 + 8 + \ldots + N = 2N - 1 \in \Theta(N)$

Sum of first N powers of 2

$1 + 2 + 4 + 8 + \ldots + 2^N = 2^N - 1$

True or false?

If $f(n) \in O(n^2)$ and $g(n) \in O(n)$ are positive-valued functions, then $f(n) / g(n) \in O(n)$.

If $f(n) \in \Theta(n^2)$ and $g(n) \in \Theta(n)$ are positive-valued functions, then $f(n) / g(n) \in \Theta(n)$.

True or false?

If $f(n) \in O(n^2)$ and $g(n) \in O(n)$ are positive-valued functions, then $f(n) / g(n) \in O(n)$.

FALSE

$f(n) = n^2$

$g(n) = \frac{1}{n}$

$\rightarrow \frac{f(n)}{g(n)} = n^3$

If $f(n) \in \Theta(n^2)$ and $g(n) \in \Theta(n)$ are positive-valued functions, then $f(n) / g(n) \in \Theta(n)$.

TRUE

Recursion

Find a simple $f(N)$ such that the runtime $R(N) \in \Theta(f(N))$.

```
int recursive(int n) {
    if (n <= 1) {
        return 1;
    }
    return recursive(n - 1) + recursive(n - 1);
}
```

A. 1

B. $\log n$

C. n

D. $n \log n$

E. n^2

F. 2^n

Recurrence Relations

$$T(n) = 2T(n - 1) + 1$$

Solving recurrence relations

Substitution method

Recursion tree

Master Theorem

Substitution method

1. Guess the form of the solution
2. Verify by induction
3. Solve for constants

Substitution method

$T(n) = 2T(n - 1) + 1$

Substitution method

$T(1) = 1$

$T(n) = 2T(n-1) + 1$

Guess $k \cdot 2^n - b$

Base case $n = 1$

$T(1) = 1 \leq k \cdot 2^1 - b$

$= 2k - b$

True when $k \geq \frac{b+1}{2}$

Inductive step: Assume true for $n-1$, show for n

$T(n) = 2(n-1) + 1$

$\leq 2(k 2^{n-1} - b) + 1$ (by IH)

$= k 2^n - 2b + 1 \leq k \cdot 2^n - b$

True when $b \geq 1$

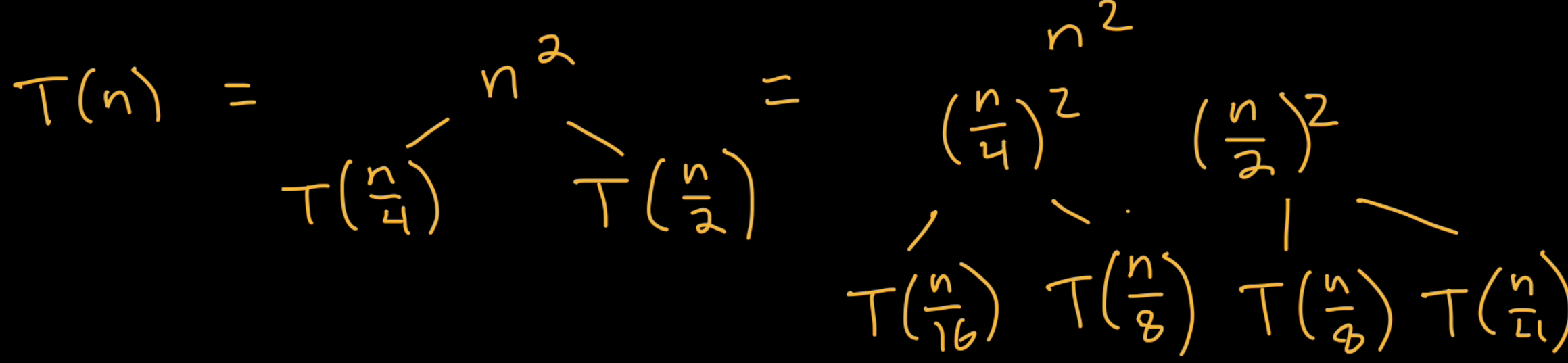
2^n

Recursion Tree

$T(n) = T(n / 4) + T(n / 2) + n^2$

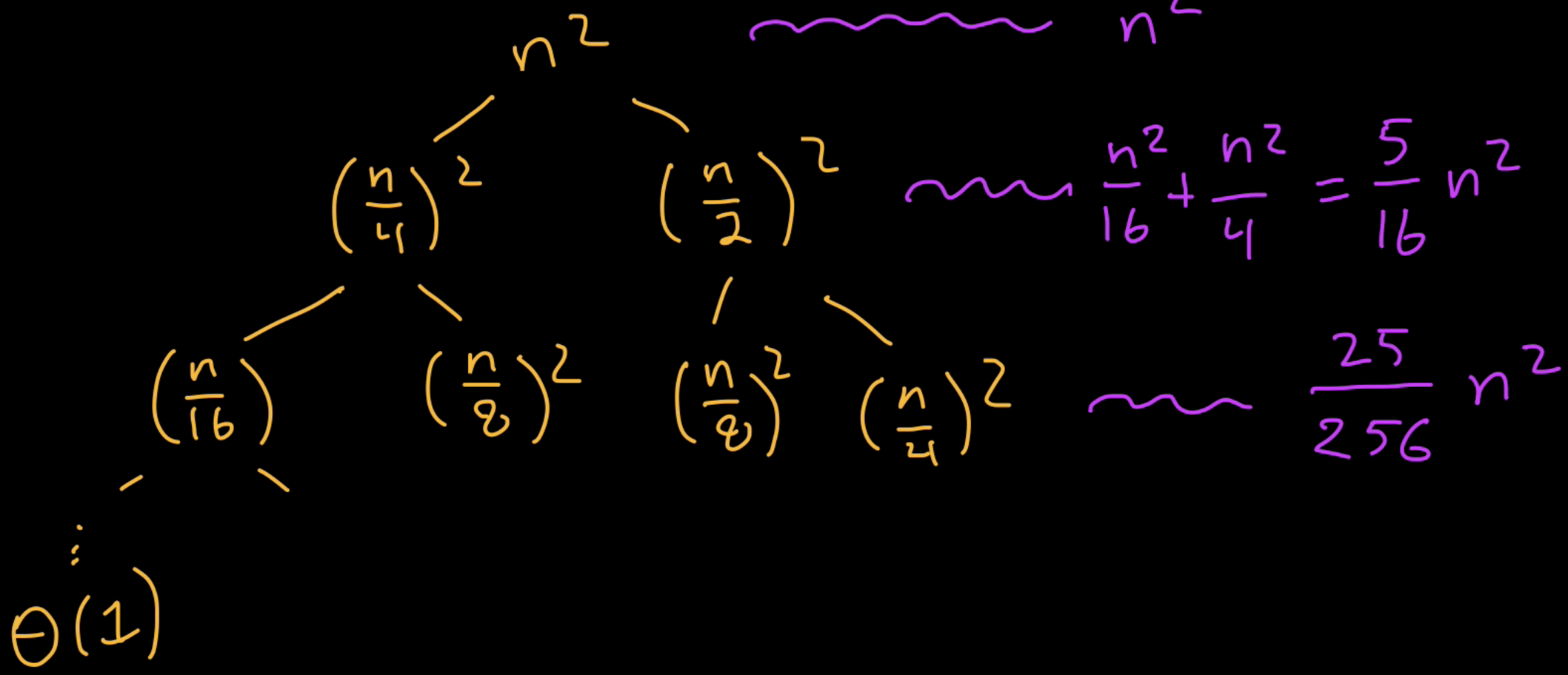
Recursion Tree

$T(n) = T(n / 4) + T(n / 2) + n^2$



Recursion Tree

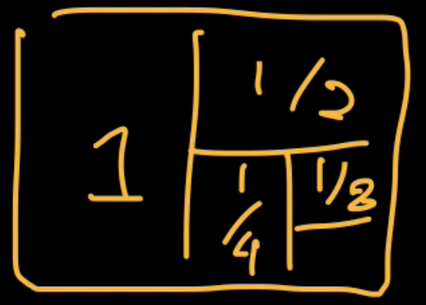
$T(n) = T(n / 4) + T(n / 2) + n^2$



Recursion Tree

$T(n) = T(n / 4) + T(n / 2) + n^2$

$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots = 2$



< 2

$O(n^2)$

$\text{Total} = \left(1 + \frac{5}{16} + \frac{25}{256} + \ldots + \frac{5^k}{16^k}\right) n^2 < 2n^2$

Master Theorem

Solve recurrences of the form $T(n) = a\,T(n / b) + f(n)$

where $a \geq 1$, $b > 1$ and f is asymptotically positive.

If $f(n) \in \Theta(n^c)$, then

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{if } a > b^c \\ \Theta(n^c \log n) & \text{if } a = b^c \\ \Theta(n^c) & \text{if } a < b^c \end{cases}$$

Master Theorem

of subproblems

work for merging

Solve recurrences of the form $T(n) = a\,T(n / b) + f(n)$

where $a \geq 1$, $b > 1$ and f is asymptotically positive.

↖ f is > 0 for large n

If $f(n) \in \Theta(n^c)$, then

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{if } a > b^c \quad \text{case 1} \\ \Theta(n^c \log n) & \text{if } a = b^c \quad \text{case 2} \\ \Theta(n^c) & \text{if } a < b^c \quad \text{case 3} \end{cases}$$

Recursion

Find a simple $f(N)$ such that the runtime $R(N) \in \Theta(f(N))$.

```
int recursive(int n) {
    if (n <= 1) {
        return 1;
    }
    return recursive(n - 1) + recursive(n - 1);
}
```

A. 1

B. $\log n$

C. n

D. $n \log n$

E. n^2

F. 2^n

Recursion

Find a simple $f(N)$ such that the runtime $R(N) \in \Theta(f(N))$.

```
int recursive(int n) {
    if (n <= 1) {
        return 1;
    }
    return recursive(n - 1) + recursive(n - 1);
}
```

A. 1

B. $\log n$

C. n

D. $n \log n$

E. n^2

F. 2^n

Find Sum

Interview Question

Given an integer **`sum`** and a sorted array **`numbers`** of **`N`** distinct integers, implement a function to find if there exist indices **`i`** and **`j`** such that **`numbers[i] + numbers[j] == x`**.

```
bool findSum(const vector<int> &numbers, int sum) {
    for (int i                                  ) {
        for (int j                                  ) {
            if (                                    ) {

            }
        }
    }
    return
}
```

Find Sum

Interview Question

Given an integer **`sum`** and a sorted array **`numbers`** of **`N`** distinct integers, implement a function to find if there exist indices **`i`** and **`j`** such that **`numbers[i] + numbers[j] == x`**.

```
bool findSum(const vector<int> &numbers, int sum) {
    for (int i = 0; i < numbers.size(); i += 1) {
        for (int j = 0; j < numbers.size(); j += 1) {
            if (numbers[i] + numbers[j] == sum) {
                return true;
            }
        }
    }
    return false;
}
```

Find Sum

Interview Question

Given an integer **`sum`** and a sorted array **`numbers`** of **`N`** distinct integers, implement a function to find if there exist indices **`i`** and **`j`** such that **`numbers[i] + numbers[j] == x`**.

```
bool betterFindSum(const vector<int> &numbers, int sum);
```

Implement a better, more efficient version of the algorithm.

```
break;
```

#general

☆ | 9 | 0 | Company-wide announ...

Search

Yesterday

dr.p 9:51 AM

uploaded and commented on this image: **New kitten!**

🙂 3 😀 3 😹 2 😻 2

Sorry I was offline yesterday, was busy getting this.

Message #general

`perf`

Code profiling

perf

```
perf record -F 1000 --call-graph dwarf -e cycles:u ./program
perf report
```

```
Samples: 4K of event 'cpu-clock:u', Event count (approx.): 4727000000
  Children      Self  Command  Shared Object        Symbol
+   95.56%     0.00%  ship     ship                 [.] main
+   95.56%     0.00%  ship     libc-2.17.so         [.] __libc_start_main
+   95.56%     0.00%  ship     ship                 [.] _start
+   71.61%     3.77%  ship     ship                 [.] Grid::read
+   71.61%     0.00%  ship     ship                 [.] Router::Router
+   26.85%     2.67%  ship     libstdc++.so.6.0.21  [.] std::istream::_M_extract<unsigned long>
+   23.93%     0.00%  ship     ship                 [.] Router::route
+   21.22%     6.37%  ship     libstdc++.so.6.0.21  [.] std::operator>><char, std::char_traits<char> >
+   20.06%     9.39%  ship     libstdc++.so.6.0.21  [.] std::istream::sentry::sentry
+   19.06%    15.17%  ship     libstdc++.so.6.0.21  [.] std::num_get<char, std::istreambuf_iterator<char, std::char_tra
+   18.98%     0.47%  ship     libstdc++.so.6.0.21  [.] std::num_get<char, std::istreambuf_iterator<char, std::char_tra
+   17.37%     0.47%  ship     ship                 [.] Router::findHangar
+   10.64%     7.74%  ship     libstdc++.so.6.0.21  [.] std::ostream::flush
+    9.37%     1.06%  ship     ship                 [.] Location::Location
+    7.11%     0.00%  ship     ship                 [.] std::vector<Location, std::allocator<Location> >::vector
+    7.11%     0.00%  ship     ship                 [.] std::__Construct<std::vector<Location, std::allocator<Location>
```

Sorting

Improving complexity

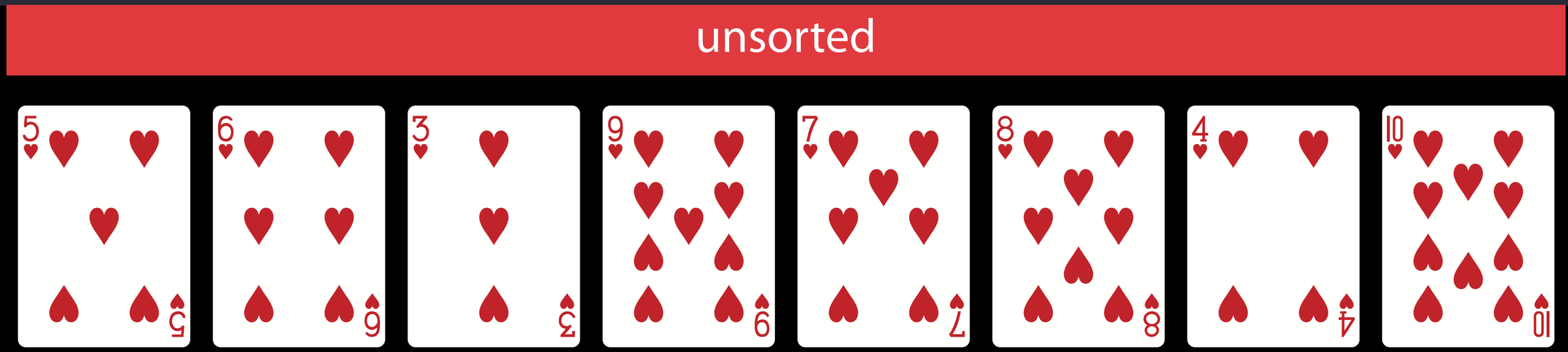
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```

What is the growth complexity of this algorithm?

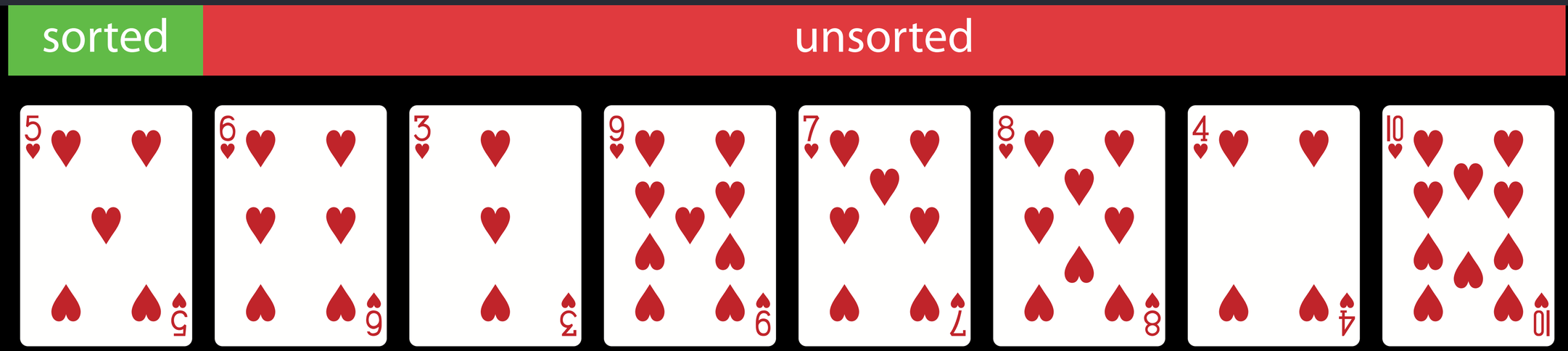
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



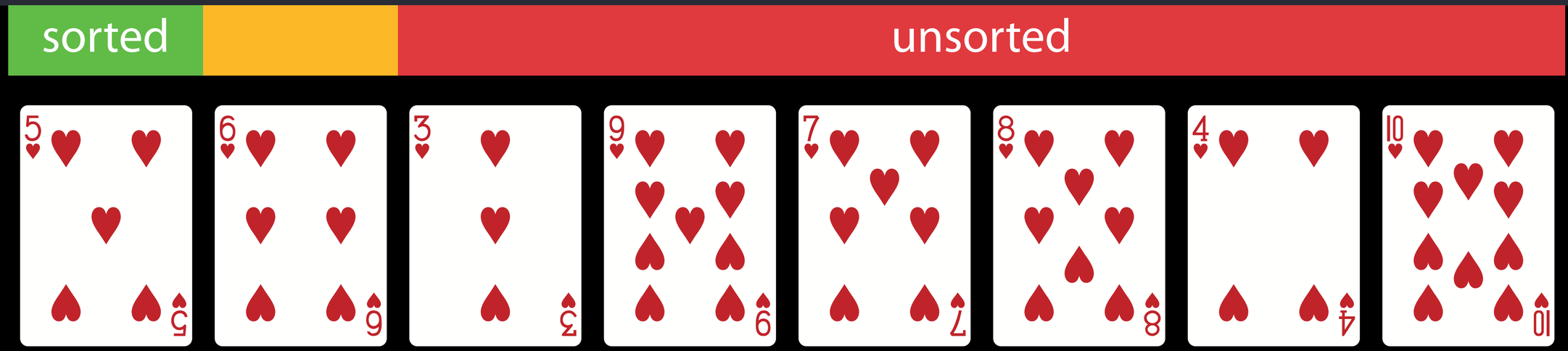
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



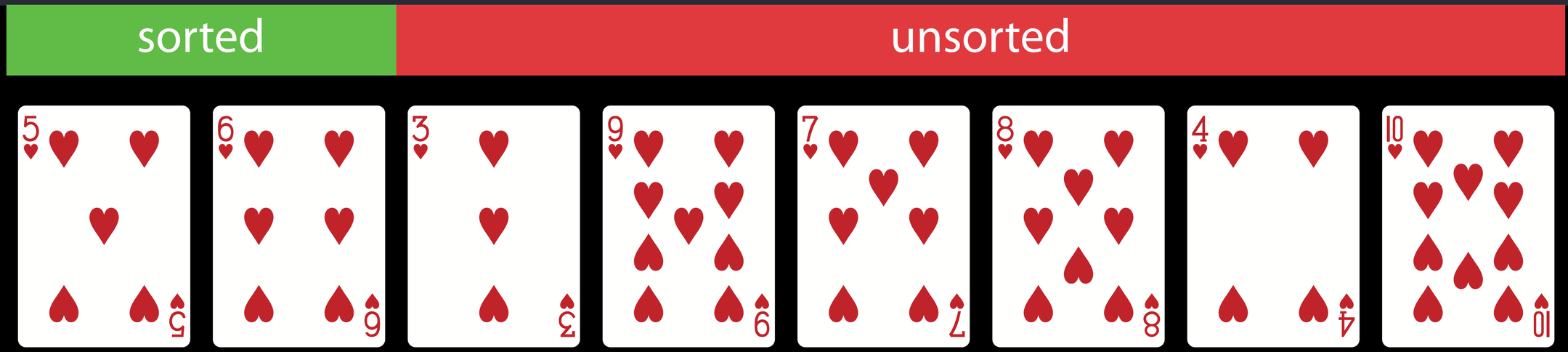
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



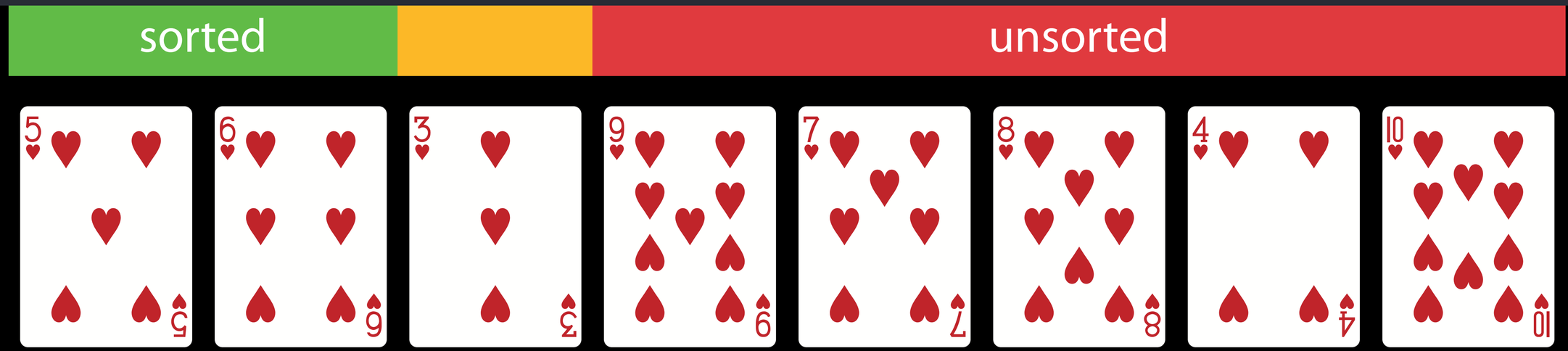
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



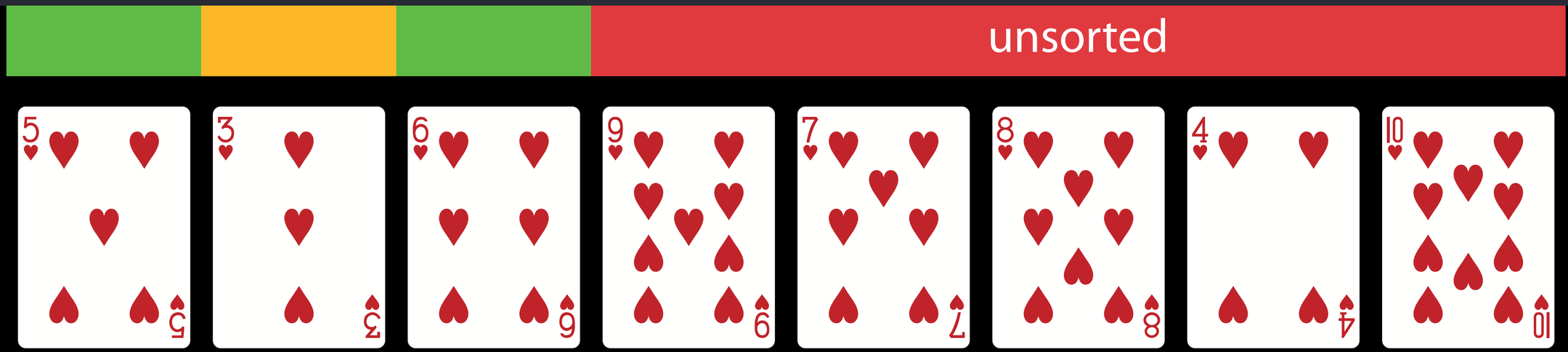
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



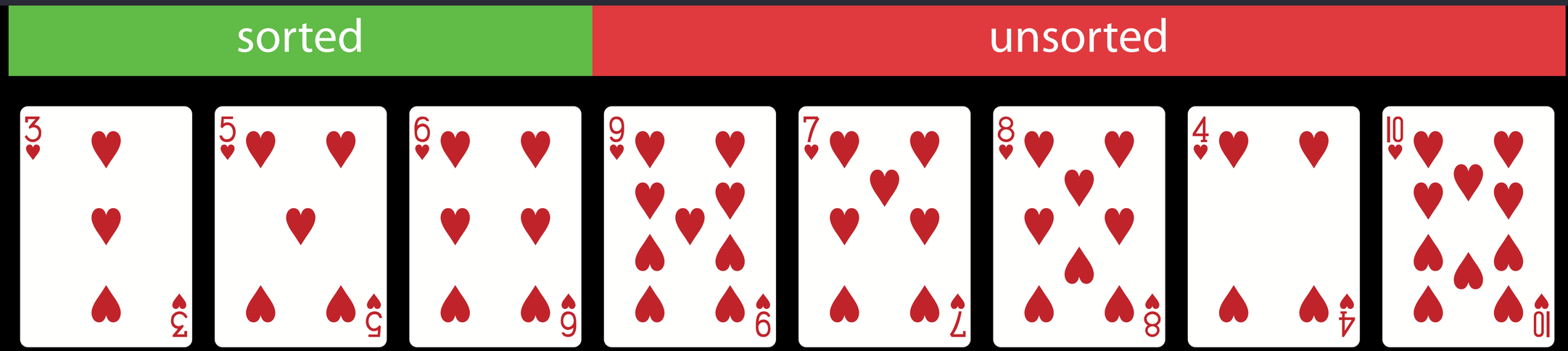
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



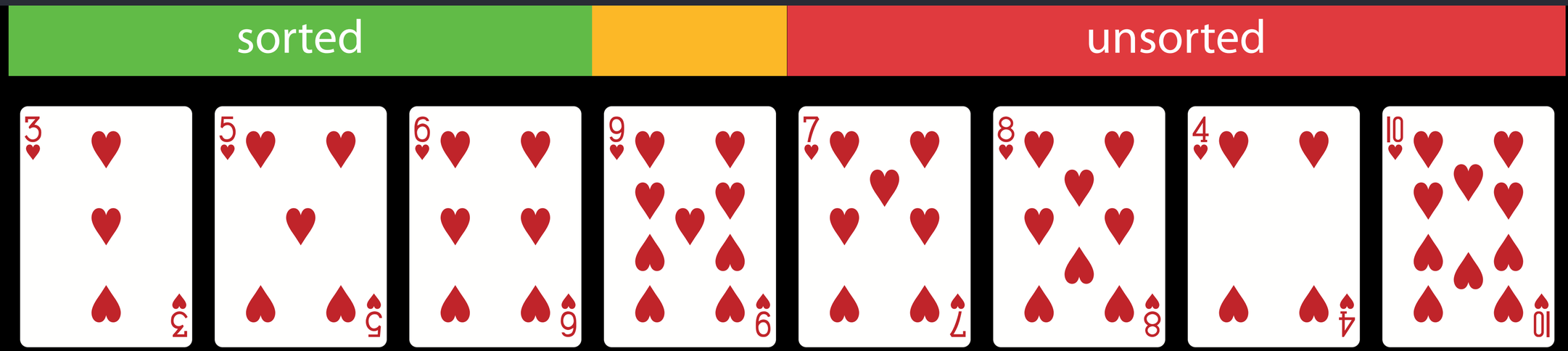
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



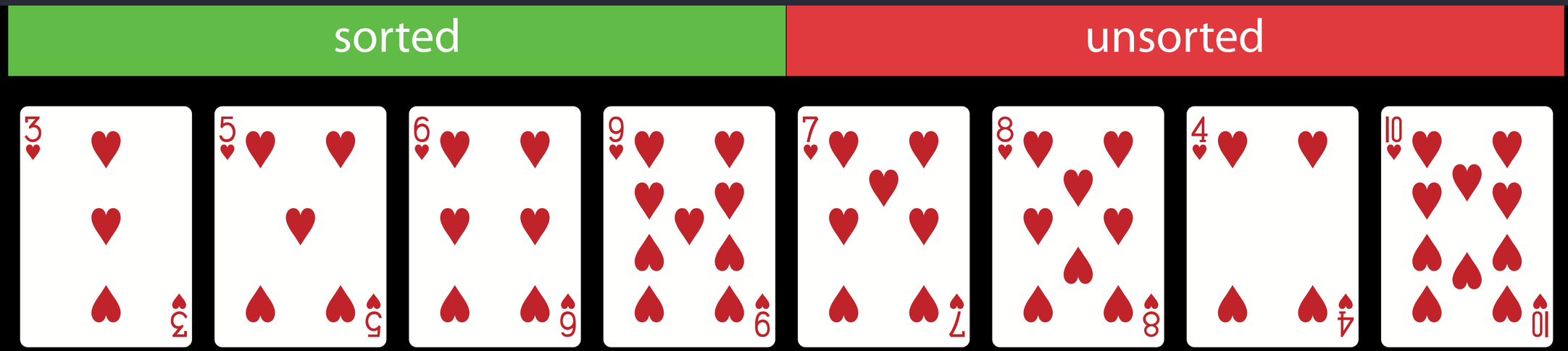
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



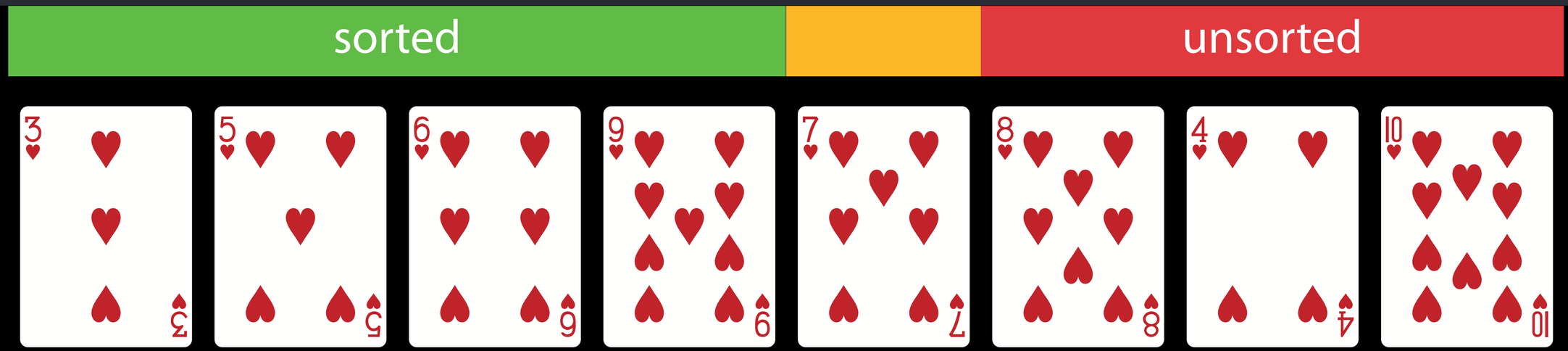
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



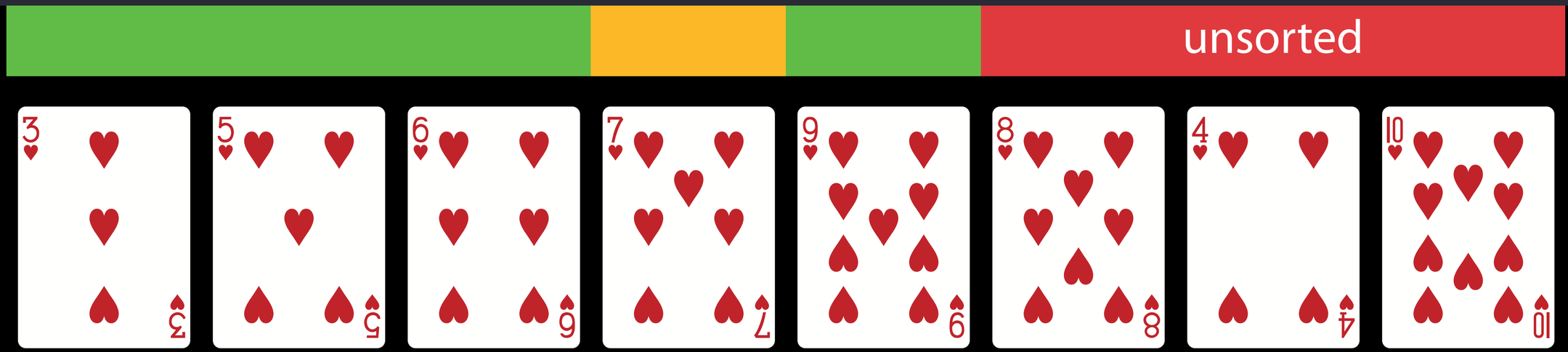
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



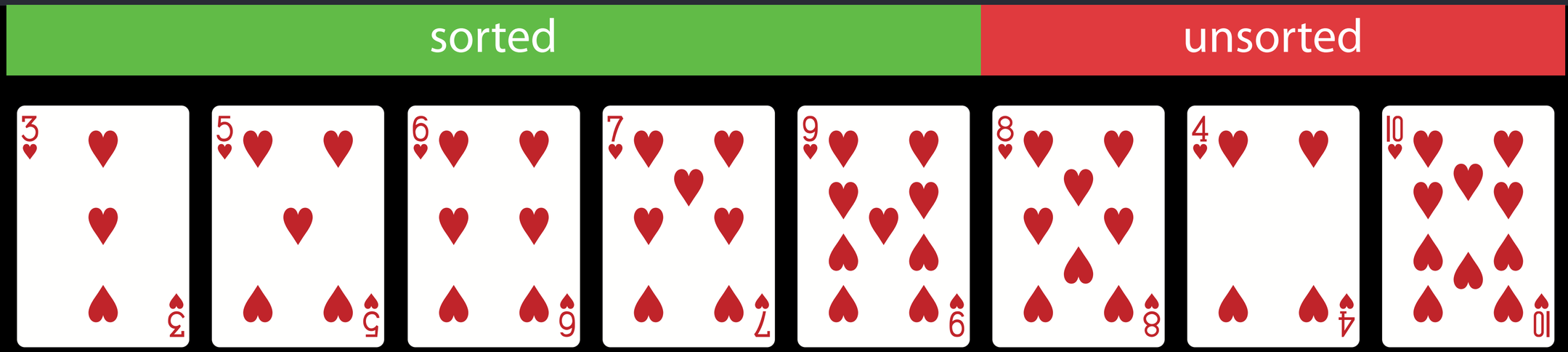
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



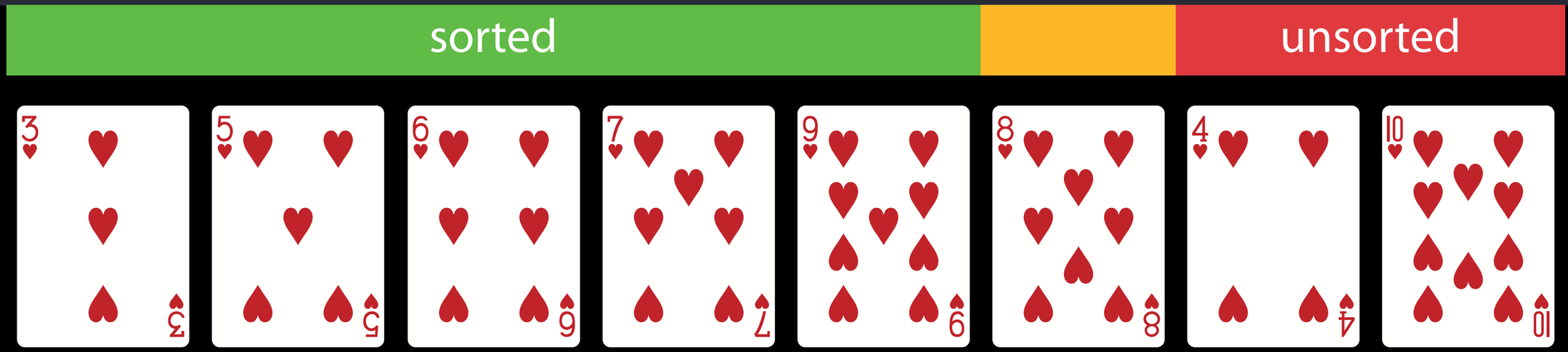
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



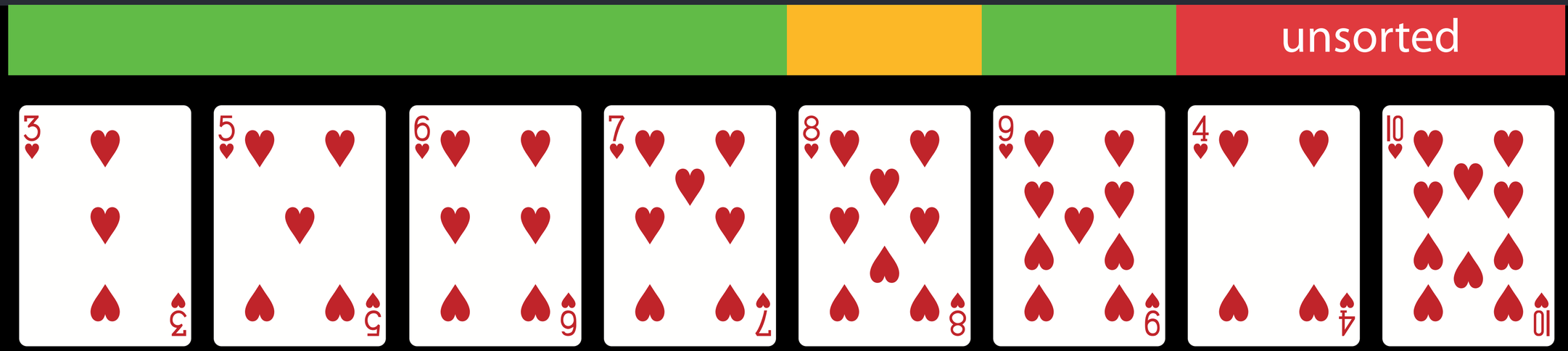
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



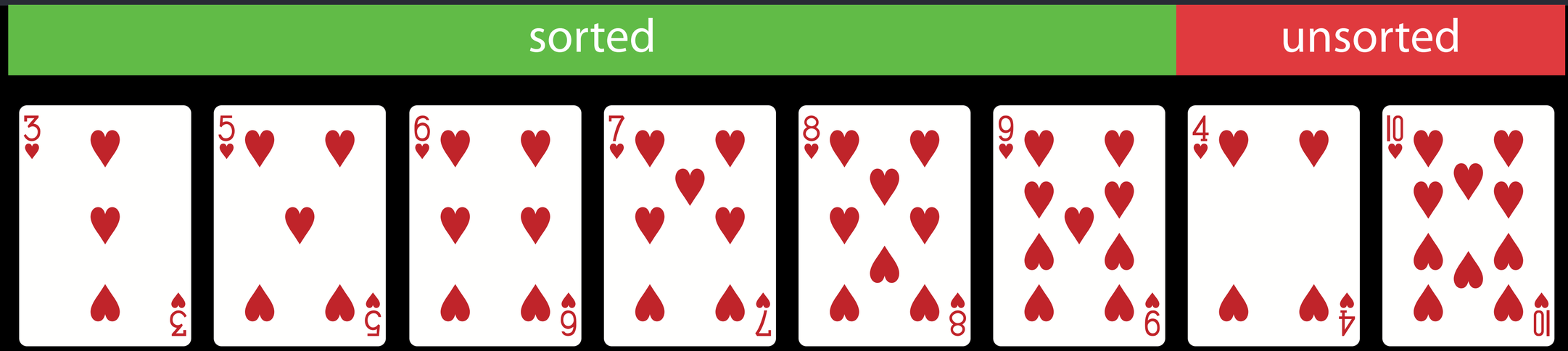
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



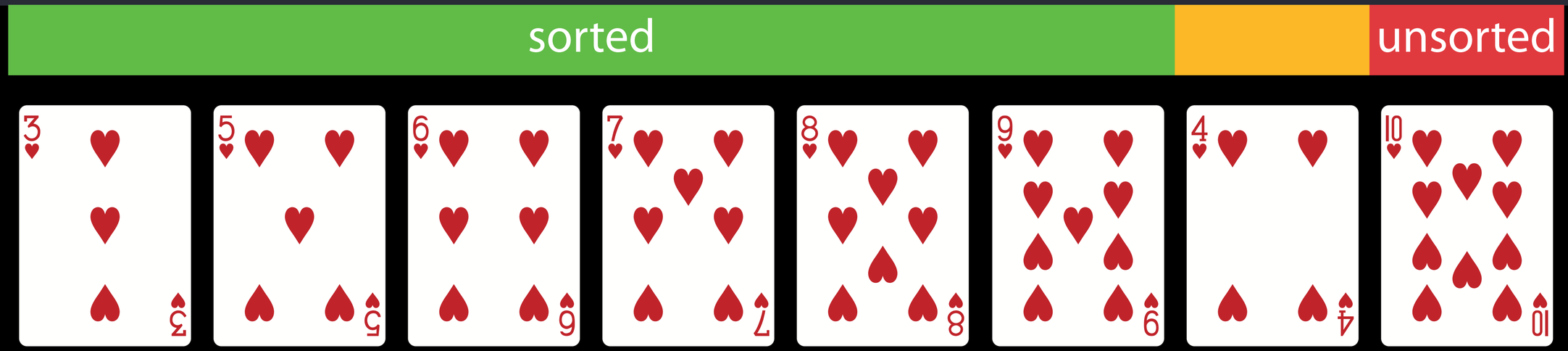
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



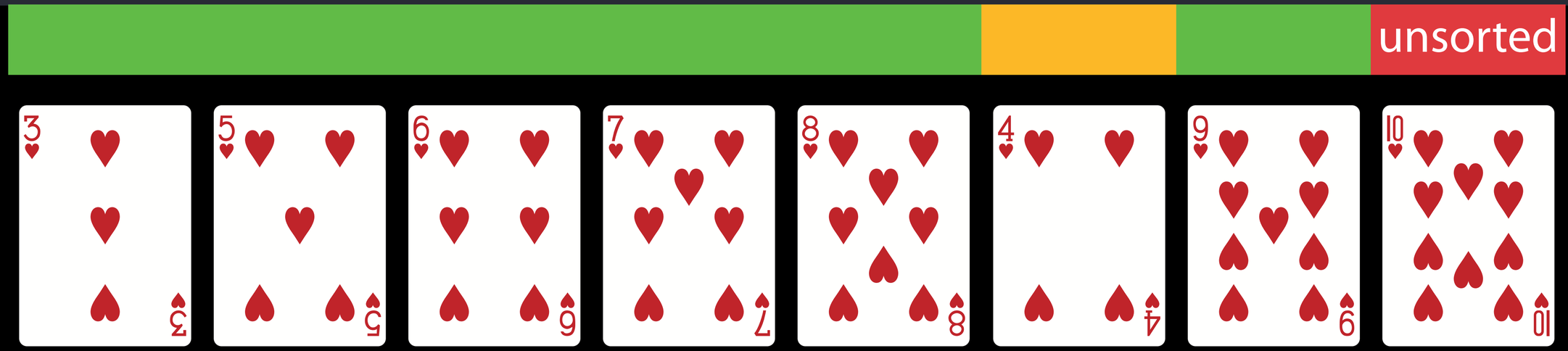
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



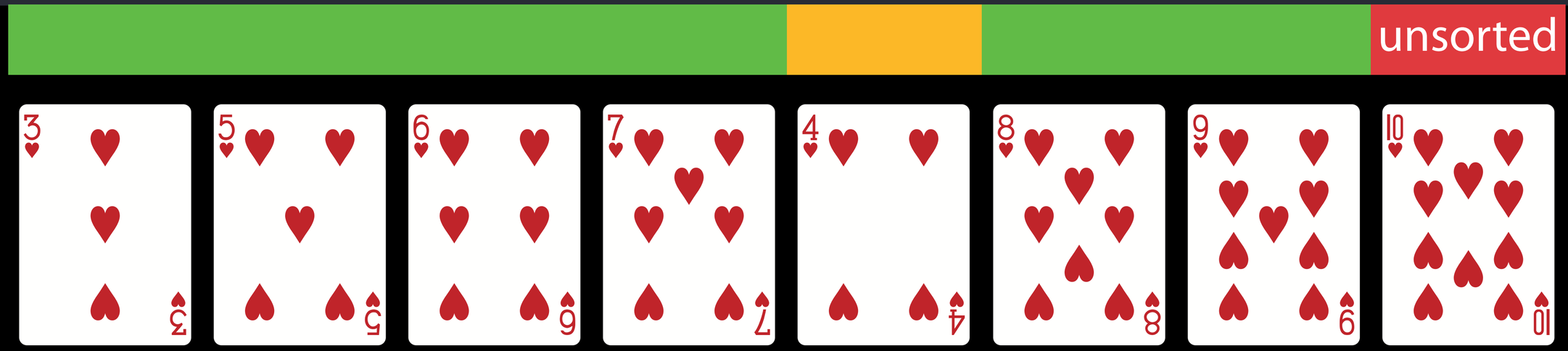
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



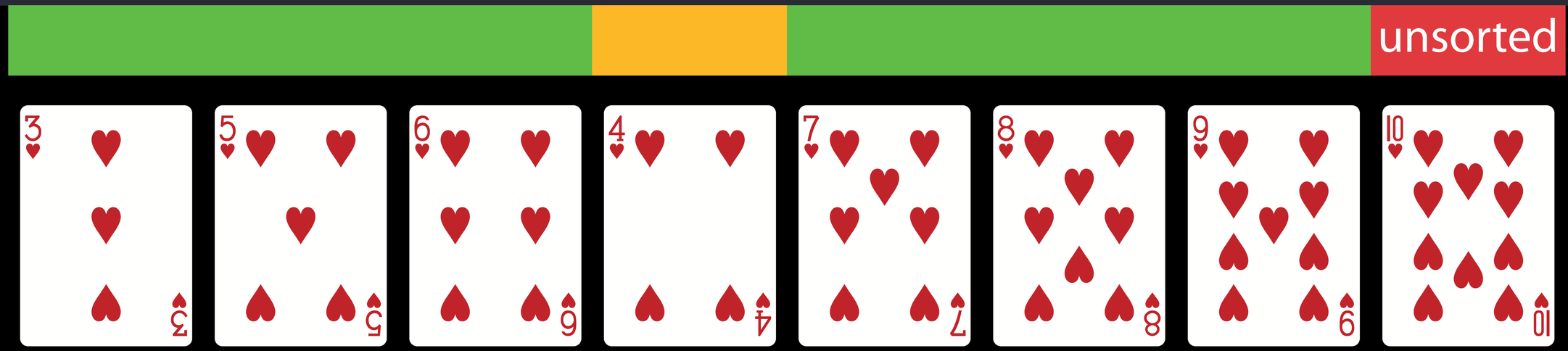
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



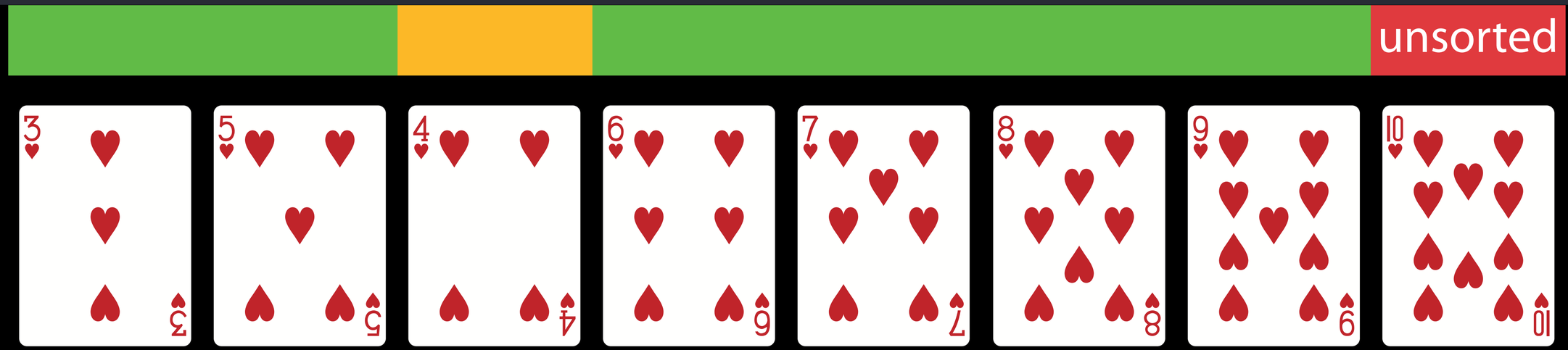
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



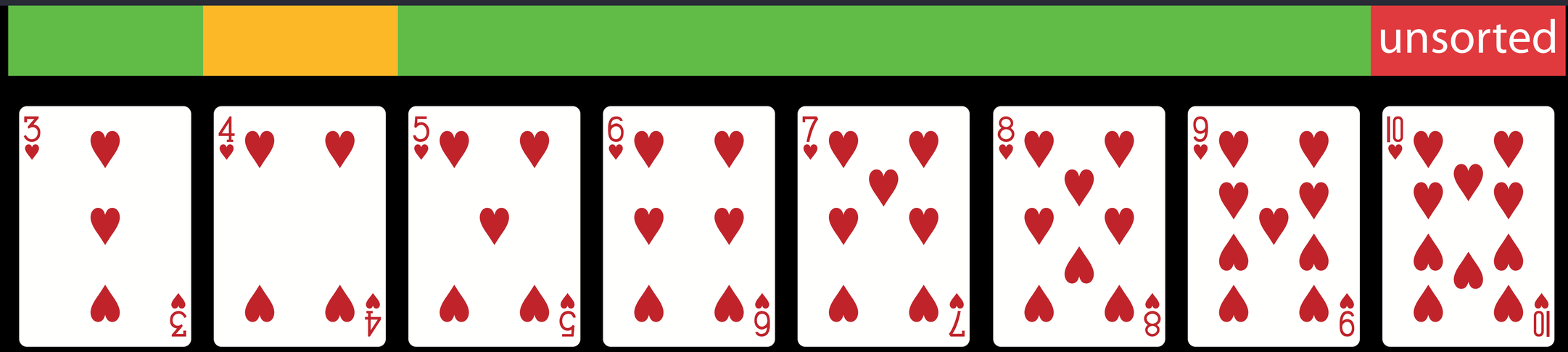
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



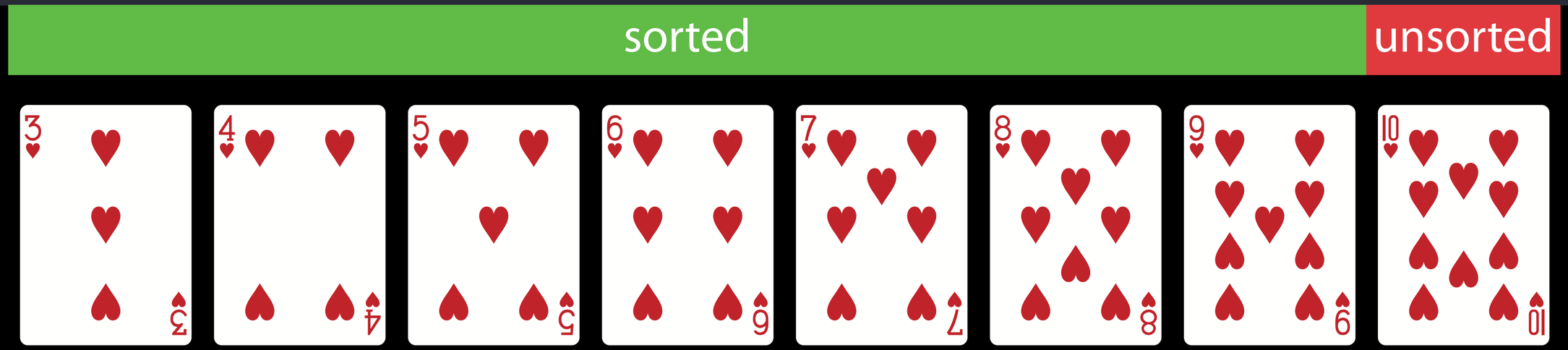
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



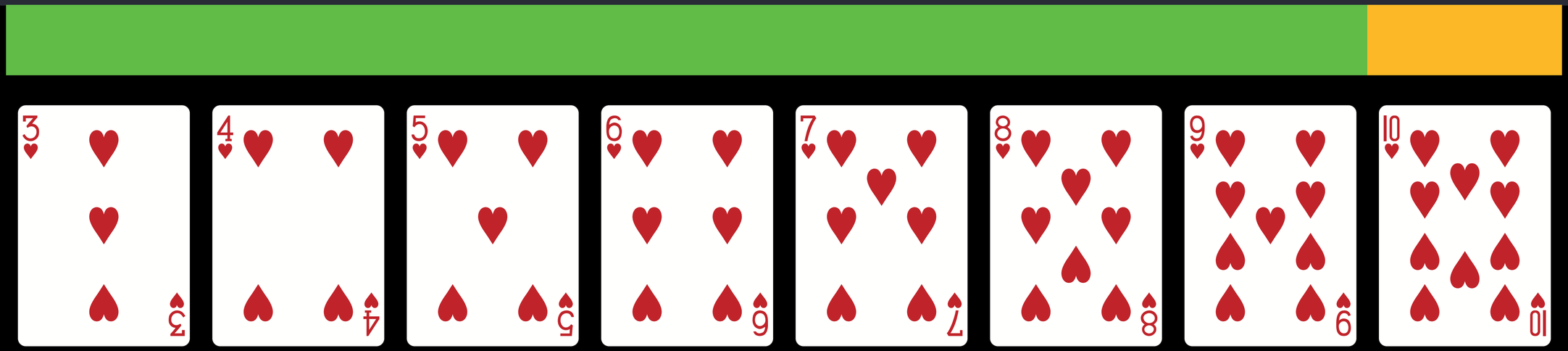
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



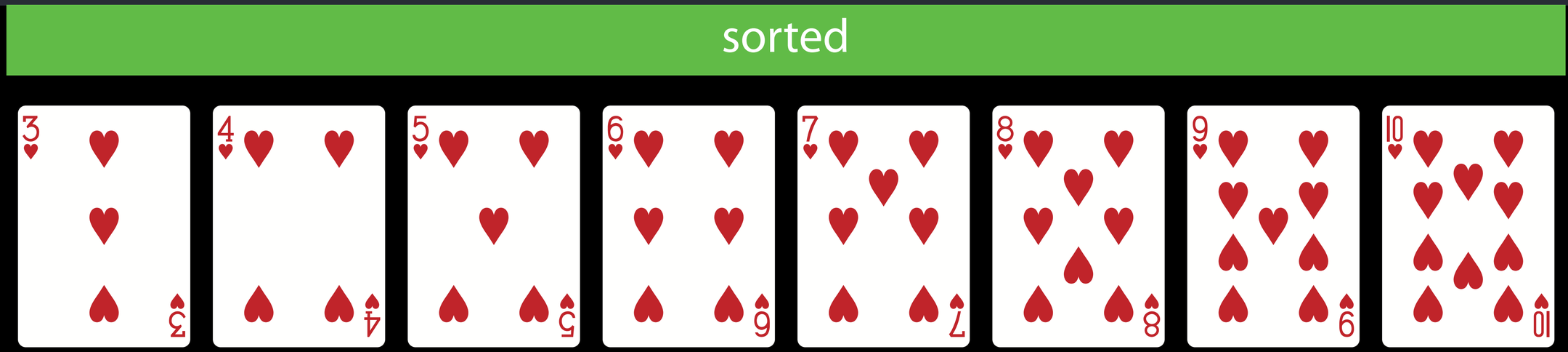
Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```



Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j – 1] > a[j])
        swap a[j] and a[j – 1]
        --j
```

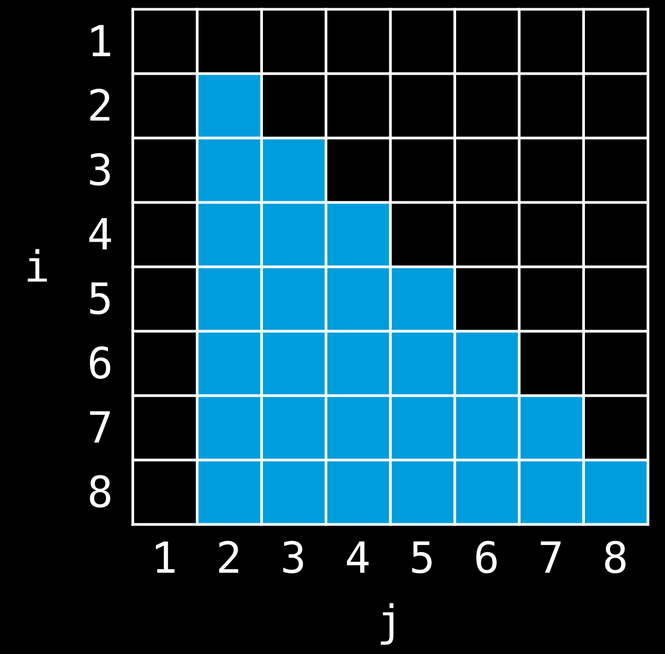
What is the growth complexity of this algorithm?

Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j − 1] > a[j])
        swap a[j] and a[j − 1]
        −−j
```

Sorting

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j − 1] > a[j])
        swap a[j] and a[j − 1]
        −−j
```



Sorting

N 32 32

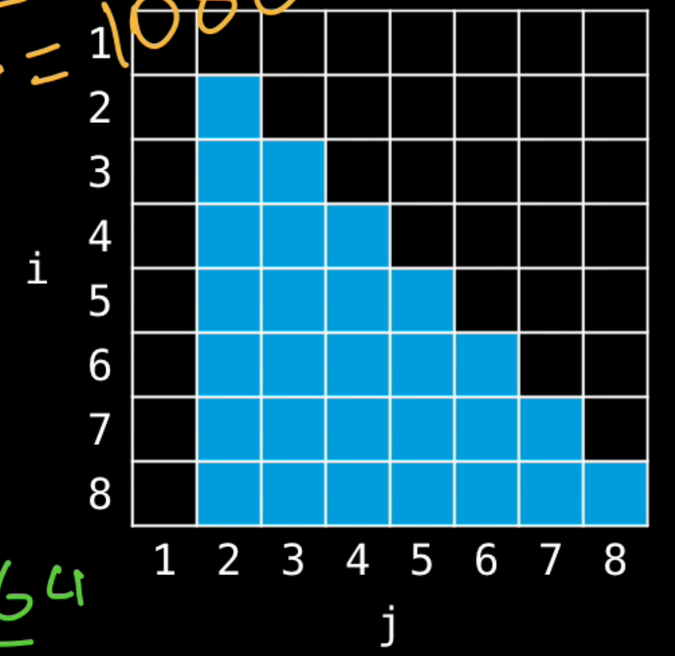
sort $\frac{32 \times 32}{2} = 512$ + 512

merging 64

512+512+64 = 1088

$\frac{N^2}{4} + \frac{N^2}{4} + N$

```
Algorithm sort0(a[], N):
for i=2 to N
    j=i
    while (j > 1) and (a[j - 1] > a[j])
        swap a[j] and a[j - 1]
        --j
```

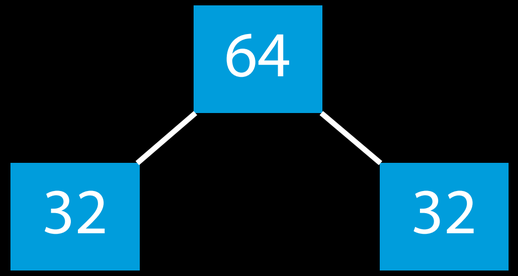


N = 64, # of comparisons ≈ $\frac{64 \times 64}{2}$

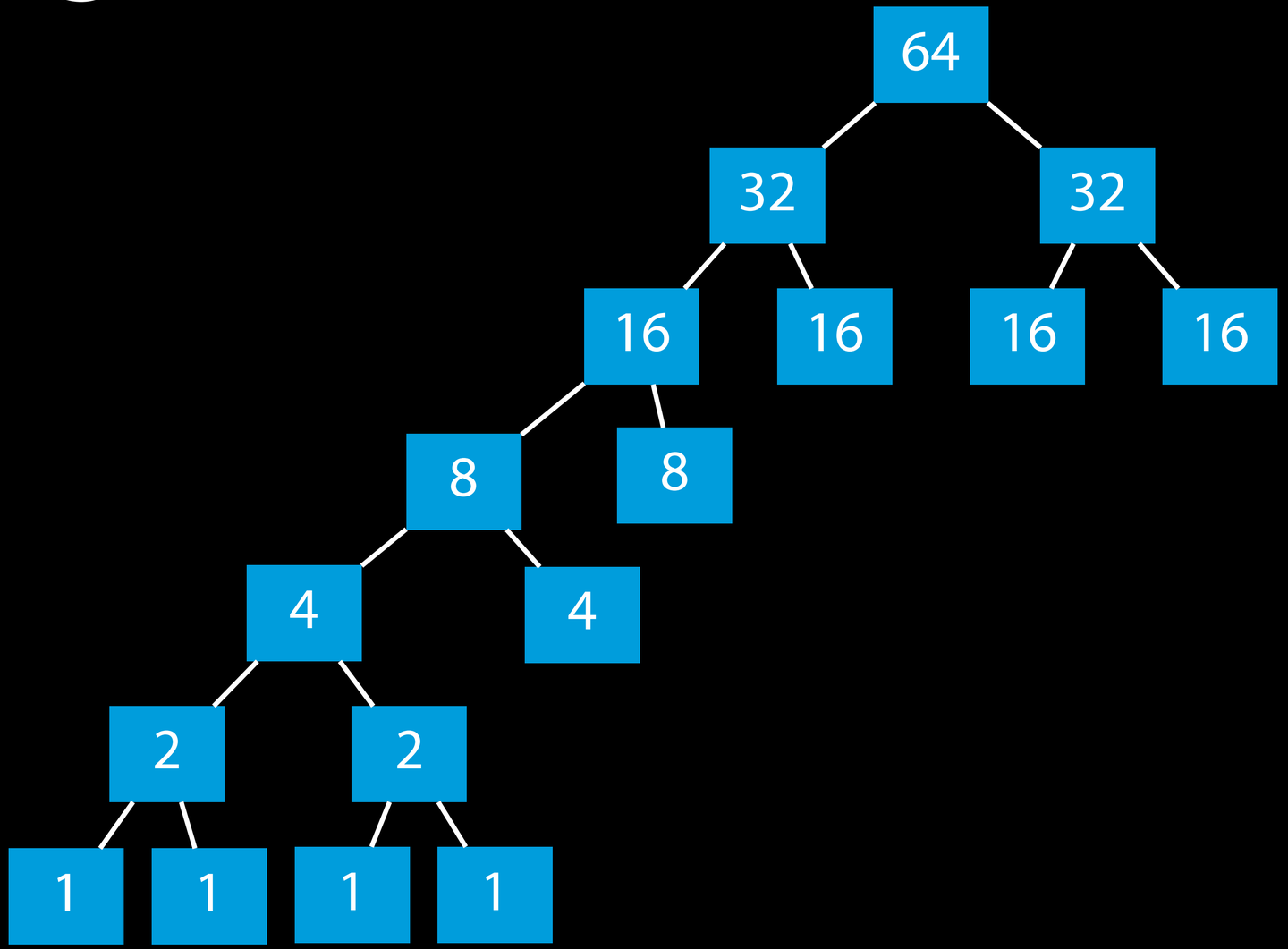
≈ 2048 comparisons

$\left(\frac{N}{2}\right)^2 + \left(\frac{N}{2}\right)^2 + N$

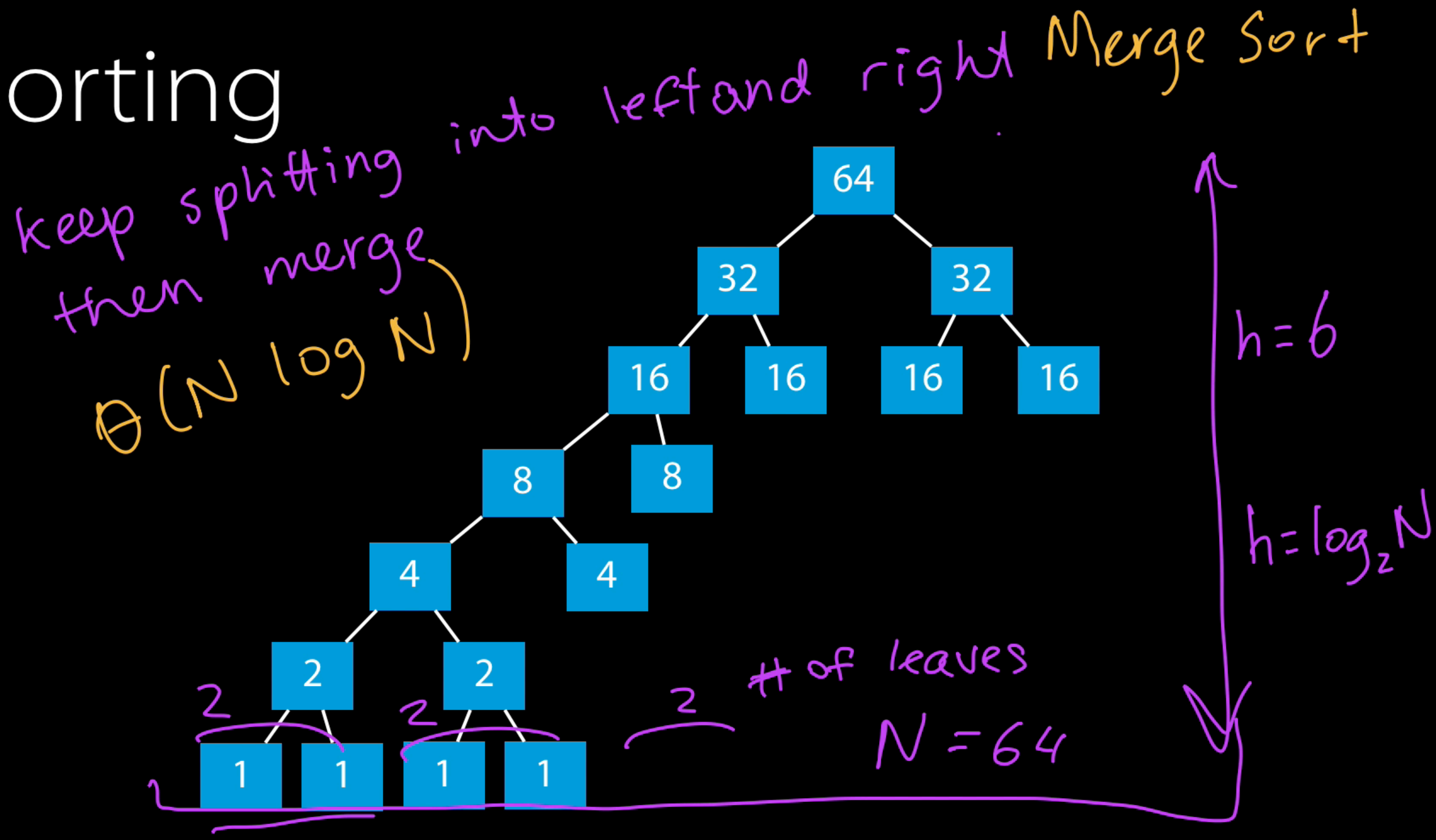
Sorting



Sorting



Sorting


Merge Sort
keep splitting into left and right
then merge
$\Theta(N \log N)$
64
32
32
16
16
16
16
8
8
4
4
2
2
2
2
2
1
1
1
1
# of leaves
$N = 64$
$h = 6$
$h = \log_2 N$

Lab 2 Assignment

1. Purrfect
   Use the `perf` tool.
2. Time to Sort
   Implement a sorting algorithm and use the `time` tool.
3. Getting Classy
   Answer questions on classes and polymorphism.
4. *O*(MG)
   Answer questions on asymptotics, recurrence relations.

merge

Implement **merge**, a function that combines the elements of two sorted vectors into a single vector and returns the result.

```
vector<int> merge(const vector<int> &a, const vector<int> &b);
```

For example, if a contains {1, 2, 4} and b contains {2, 3, 5}, **merge** should return a vector containing {1, 2, 2, 3, 4, 5}.

Q&A