Sorting

Reading Assignment: Chapter 2

- **Sorting definition.** The process of ordering a sequence of objects according to some linear order.
- **Total versus partial order.** If any two elements in a set are comparable, then the set can be totally ordered otherwise partially ordered.
- Total order
  - 9 9 14 17 86
  - Coady Müller Stege Storey Thomo
- Partial order (Topological sort)
  - CSC 110<CSC 115<CSC 225 | SENG 321<SENG 371<SENG 426

Sorting

Reading Assignment: Chapter 2

- The basic unit for analyzing the time complexity of sorting algorithms is a comparison
  - $< \leq = \geq >$ int, char, string
  - $\leq = \geq$ should not be used for real or double
- Standard character sets
  - ASCII (256 characters)
  - Unicode (95,221 characters)
    http://www.i18nguy.com/unicode/char-count.html
  - ISO
  - EBCDIC

Sorting Reference Lists

Unsorted

Thagard, P., & Verbeurgt, K. (1998). *Coherence as constraint satisfaction.*

⋮

Garey, M. R. & Johnson, D. S. (1979). *Computers and intractability.*

⋮

Flood, M. M. (1956). The traveling-salesman problem. *Operations Research.*

⋮

Gross, J. & Yellen, J. (1999). *Graph theory and its applications*.

Sorted

Flood, M. M. (1956). The traveling-salesman problem. *Operations Research.*

Garey, M. R. & Johnson, D. S. (1979). *Computers and intractability.*

Gross, J. & Yellen, J. (1999). *Graph theory and its applications*.

⋮

Thagard, P., & Verbeurgt, K. (1998). *Coherence as constraint satisfaction.*

Sorting Address/Phone Books

Unsorted

J. Smith, 1420 Cook street, (250) 234 5627

⋮

D. S. Johnson, 130 Fort street, (250) 380 5627

⋮

M. Brent, 1110 Quadra street, (250) 380 9876

⋮

J. T. Vui, 230 Linden Ave., (250) 240 6517

⋮

R. Tristan, 1200 Dallas Road, (250) 340 3425

Sorted

M. Brent, 1110 Quadra street, (250) 380 9876

⋮

D. S. Johnson, 130 Fort street, (250) 380 5627

⋮

J. Smith, 1420 Cook street, (250) 234 5627

⋮

R. Tristan, 1200 Dallas Road, (250) 340 3425

⋮

J. T. Vui, 230 Linden Ave., (250) 240 6517

Index-Entries are Sorted

(after **Dictionary of Algorithms and Data Structures**
At http://www.nist.gov/dads/)

A

abstract data type

accepting state

Ackermann's function

acyclic graph

algorithm

amortized cost

approximation algorithm

array

B

backtracking

big-O notation

binary tree

M

matching

matrix

merge sort

Z

Zeller's congruence

Zipfian distribution

Zipf's law

Classes of Sorting Algorithms

Algorithm	**Complexity**	**Remarks**
Insertionsort	$O(n^2)$	Simple
Bubblesort	$O(n^2)$	
Selectionsort	$O(n^2)$	
Shellsort	$O(n^2)$	
Mergesort	O(n log n)	External sorting
Quicksort	$O(n^2)$ worst O(n log n) expected	Hoare 1962 Most practical
Heapsort	O(n log n)	
Smoothsort	O(n log n) O(n) best case	Dijkstra 1979
Lexicographic	O(n)	Restricted input
Bin or radixsort	O(n)	Restricted input

Animations of Sorting Algorithms

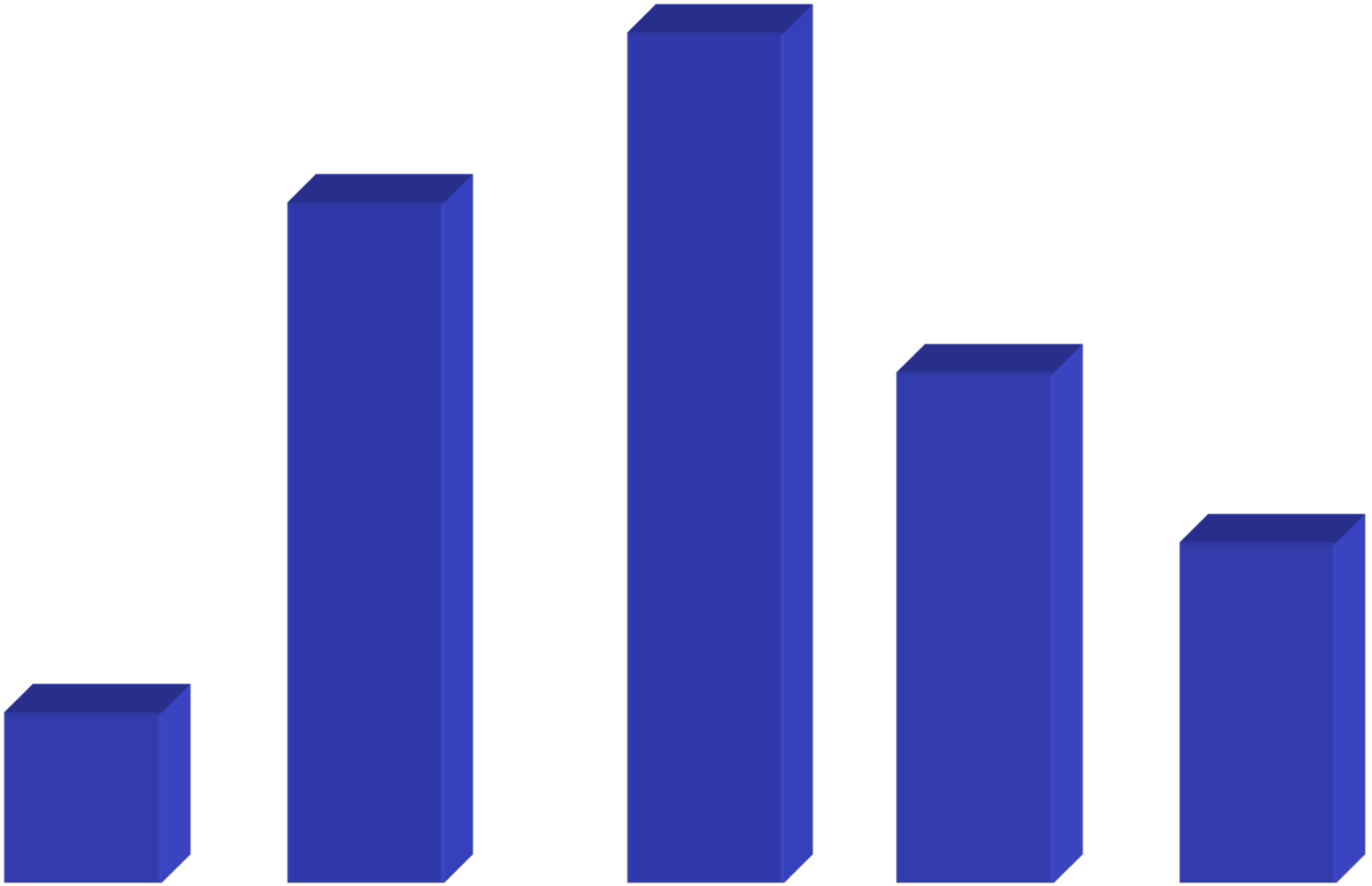
- http://research.compaq.com/SRC/JCAT/jdk10/sorting/
- http://math.baruch.cuny.edu/~bshaw/index-8.html
- http://www.cs.duke.edu/csed/jawaa2/examples/sort.html
- http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/sorting.html
- http://www.inf.ethz.ch/~staerk/algorithms/SortAnimation.html
- http://www.cs.hope.edu/~alganim/ccaa/sorting.html

Algorithm Design Technique
Brute Force

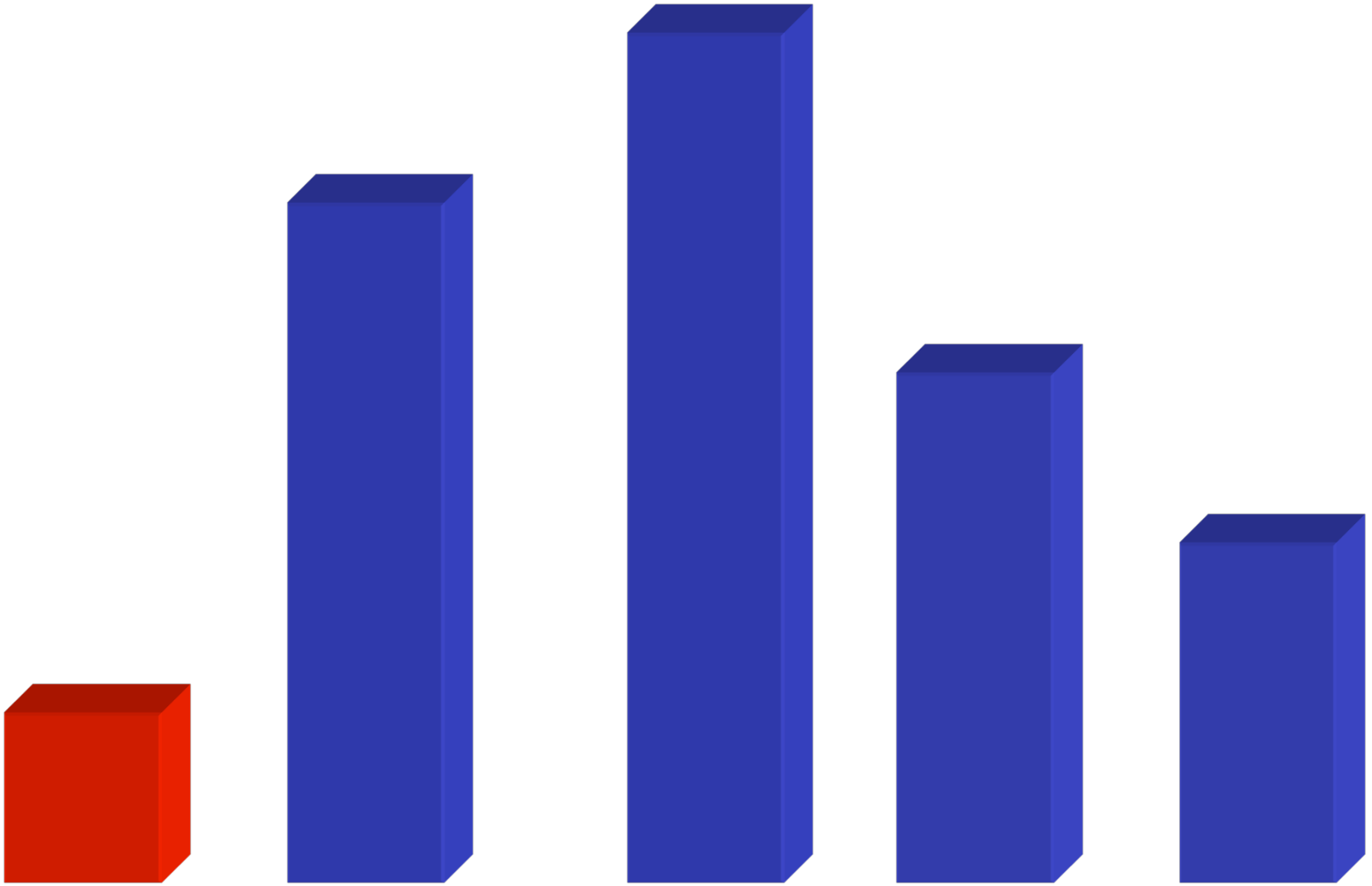
- **Brute force** is a straightforward approach to solving a problem, usually directly based on the problem statement and definitions of the concepts involved.

- Simplest algorithm design technique
- Does not usually produce the most elegant or most efficient algorithm
- Examples of the Brute Force technique
  - Selection sort
  - Bubble sort
  - Sequential search
  - Exhaustive search

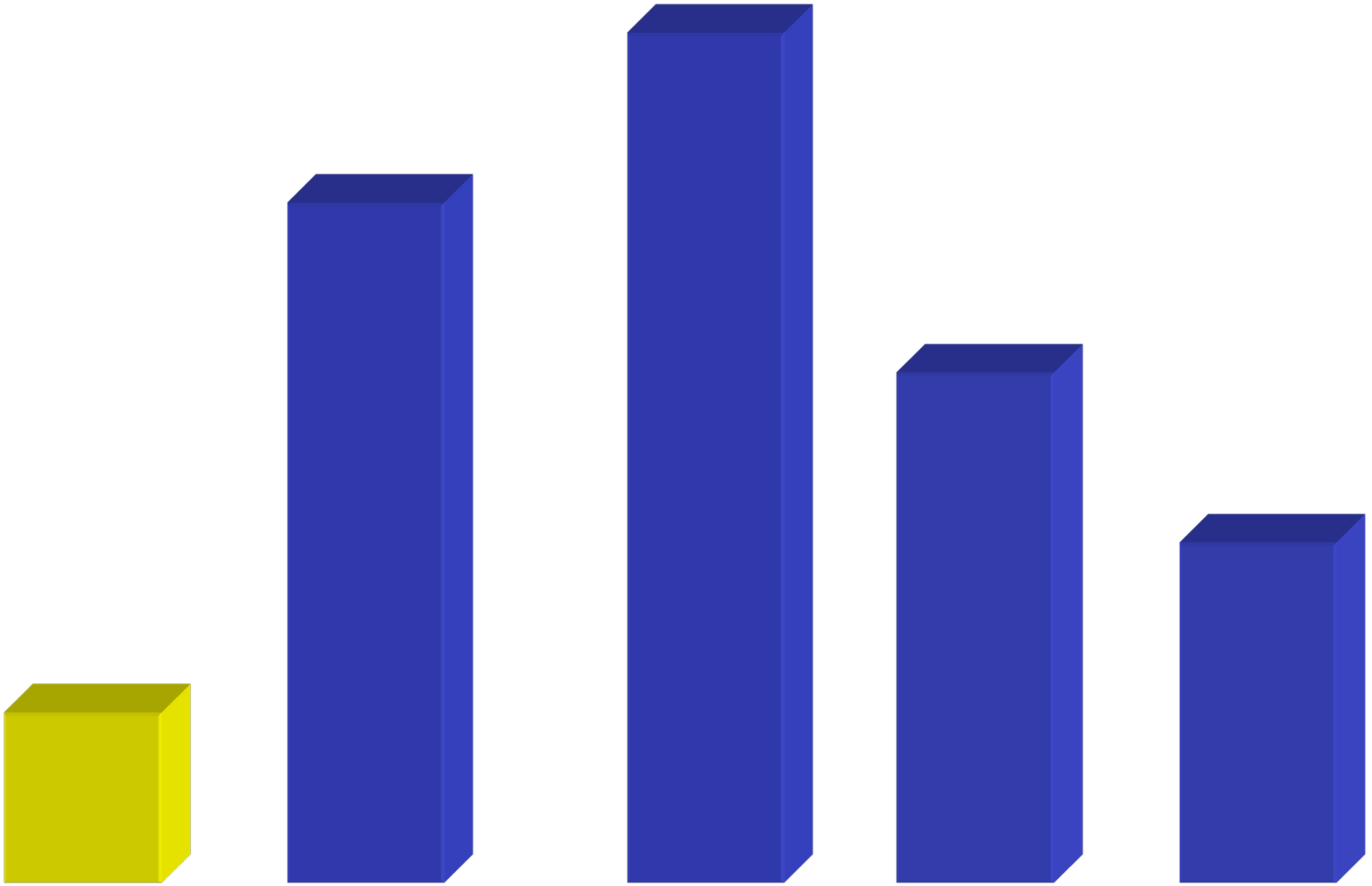
Selection Sort Animation



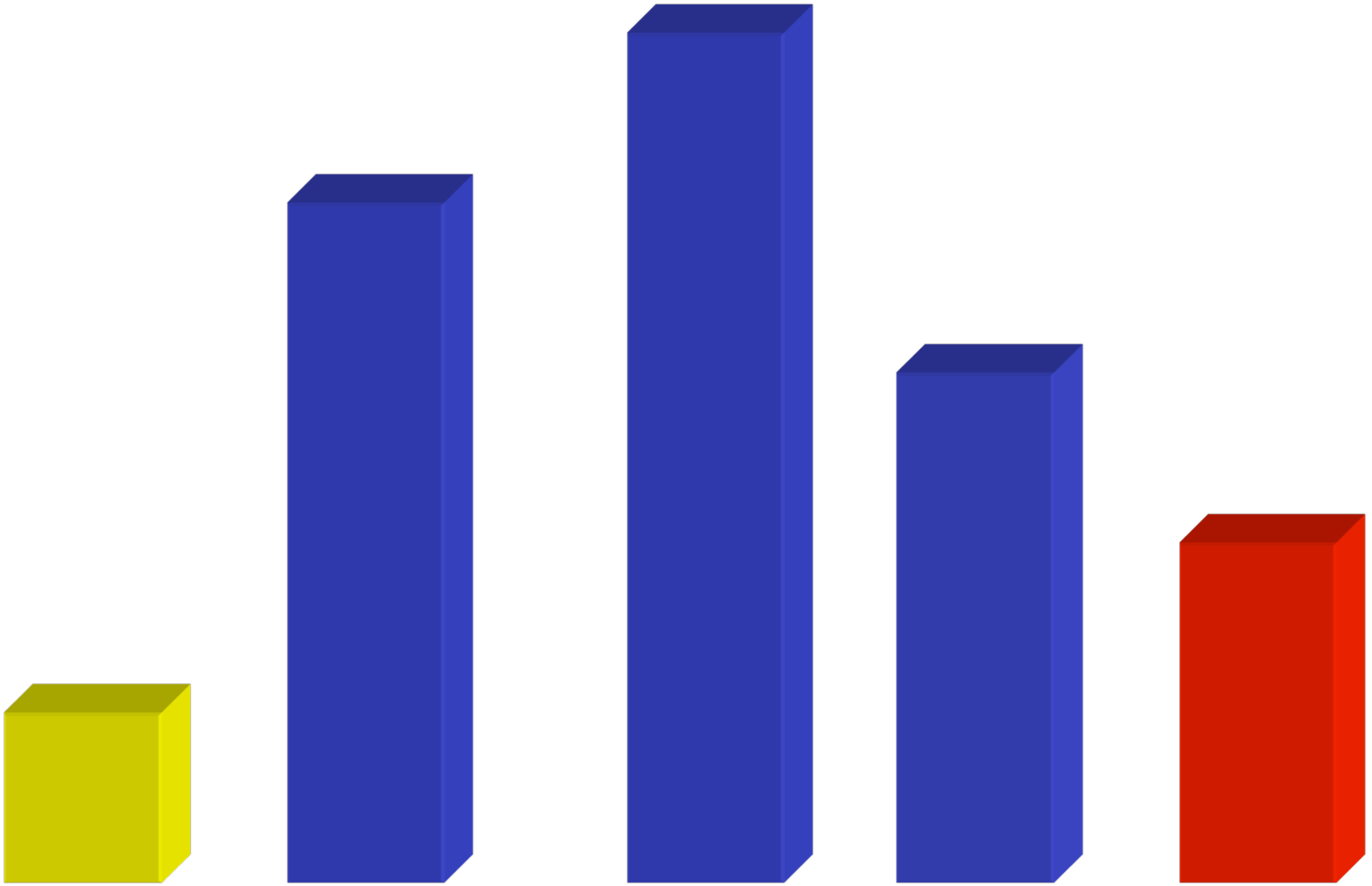
Selection Sort Animation



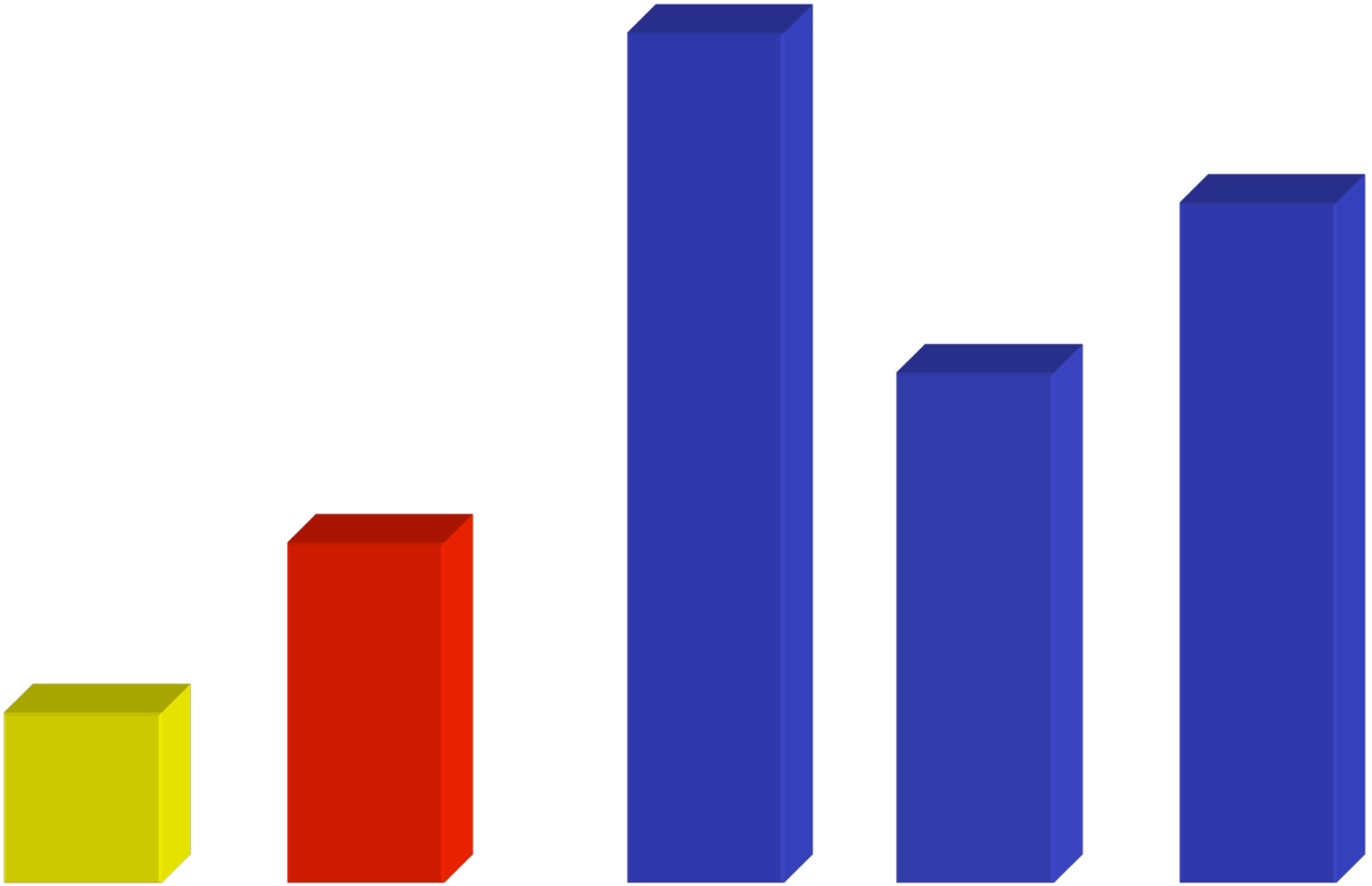
Selection Sort Animation



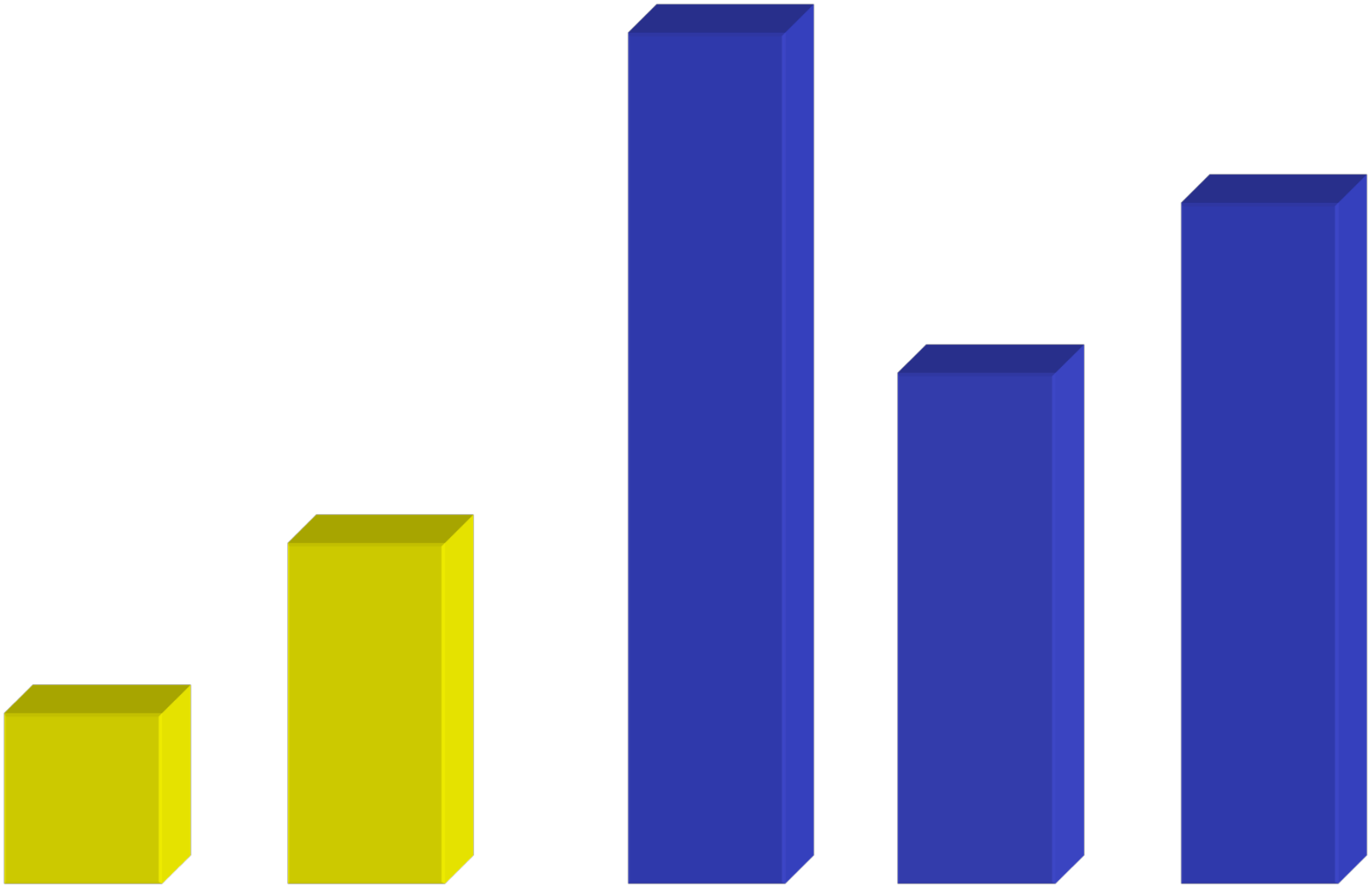
Selection Sort Animation



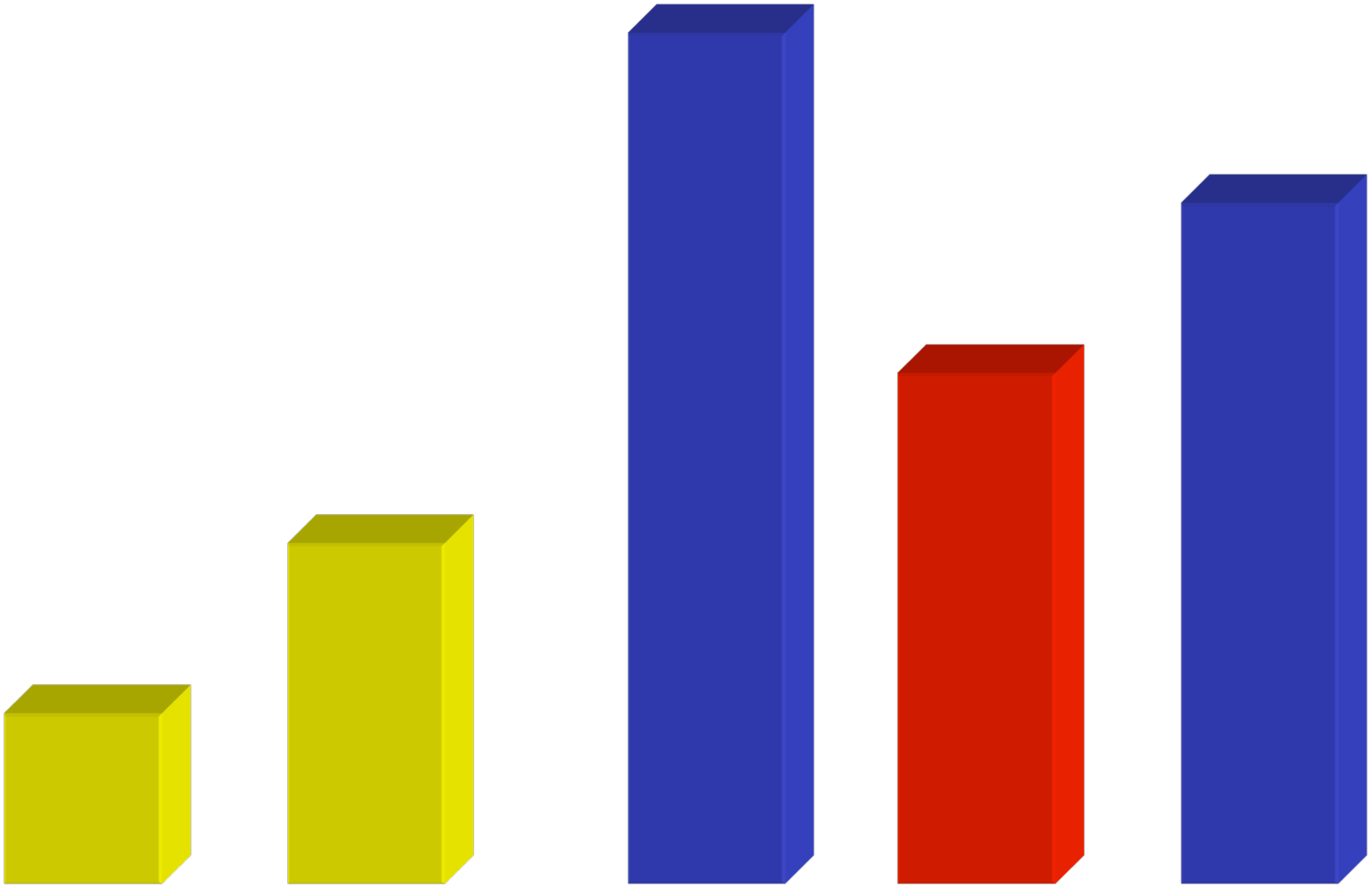
Selection Sort Animation



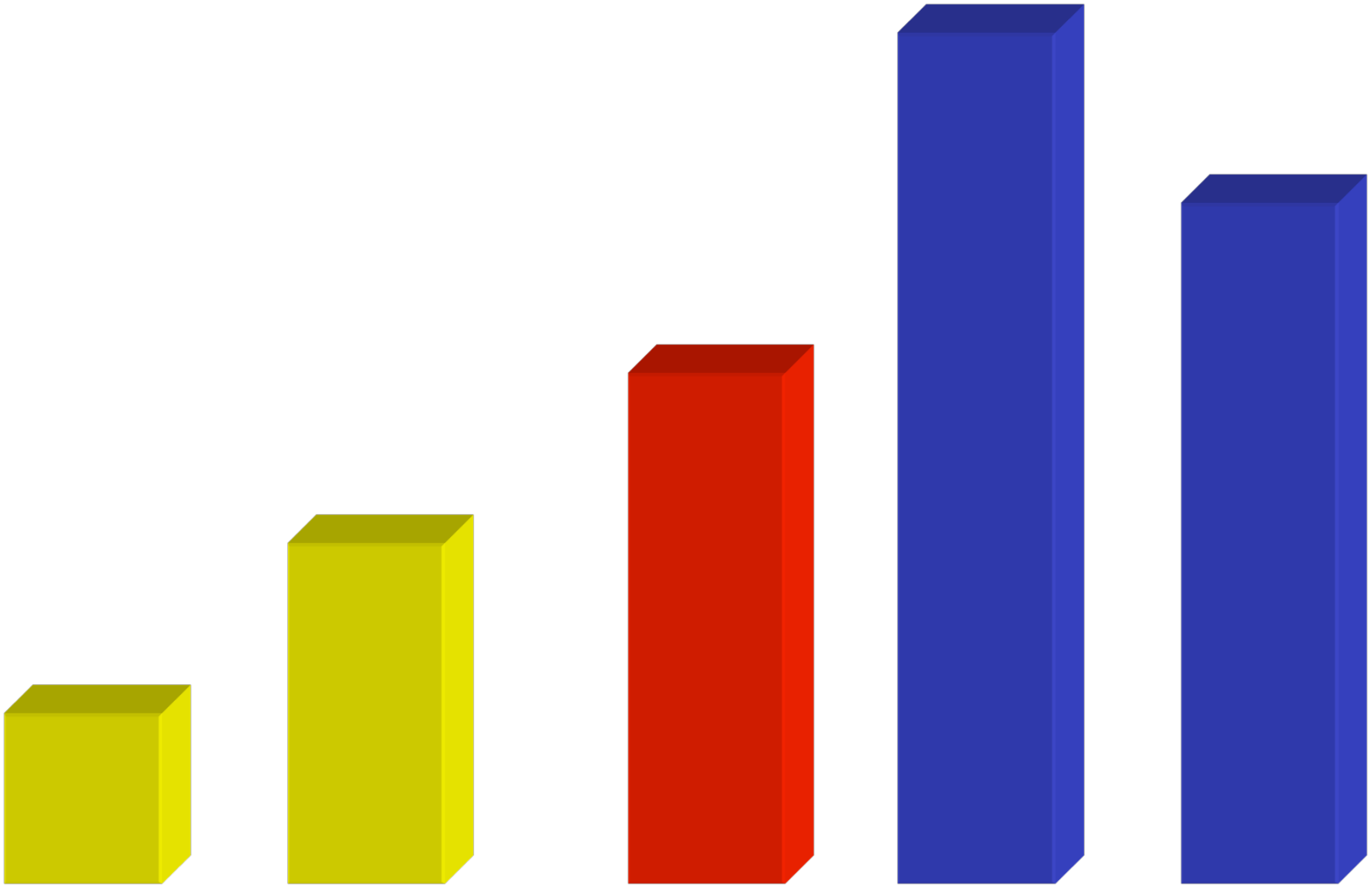
Selection Sort Animation



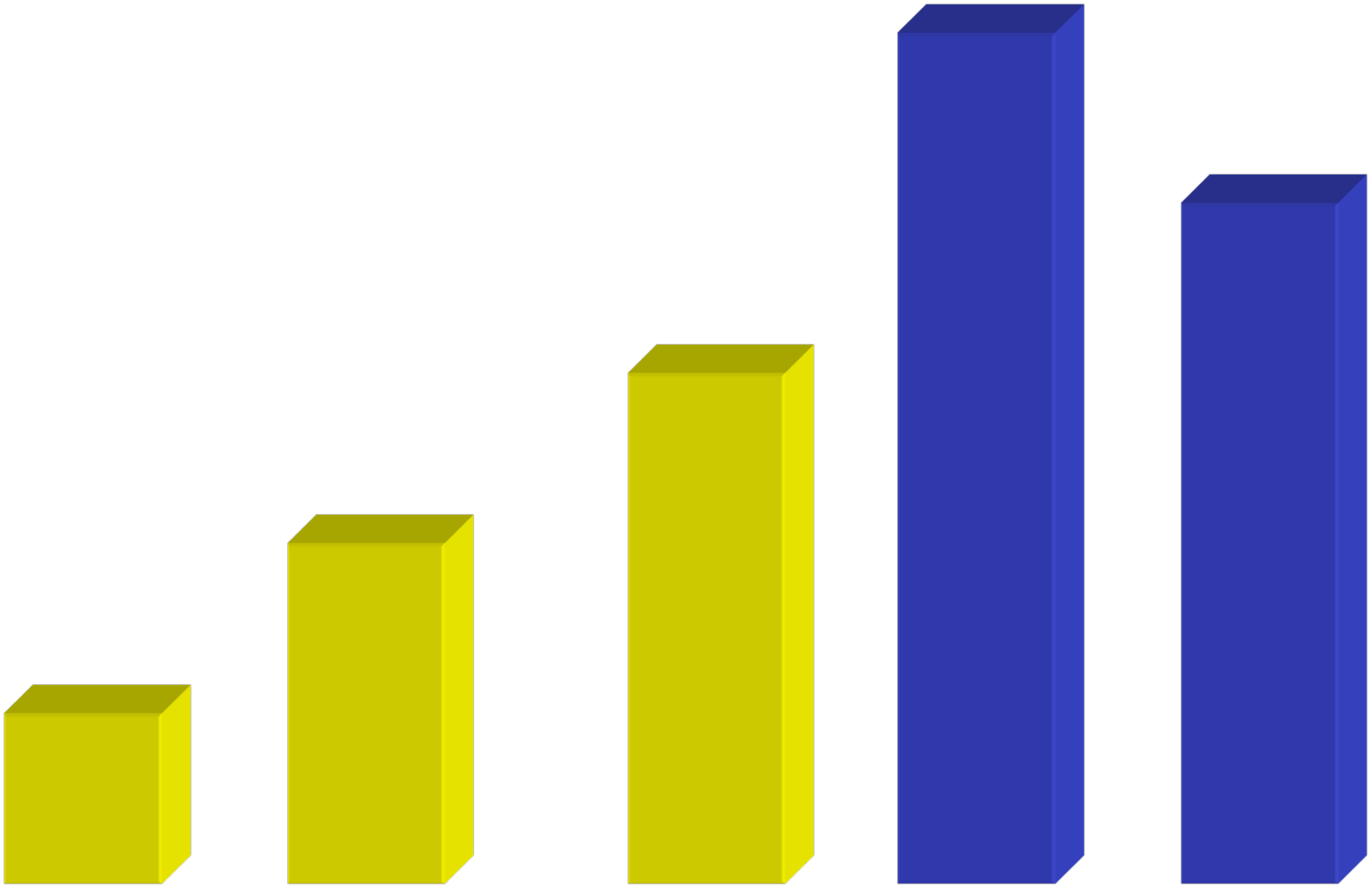
Selection Sort Animation



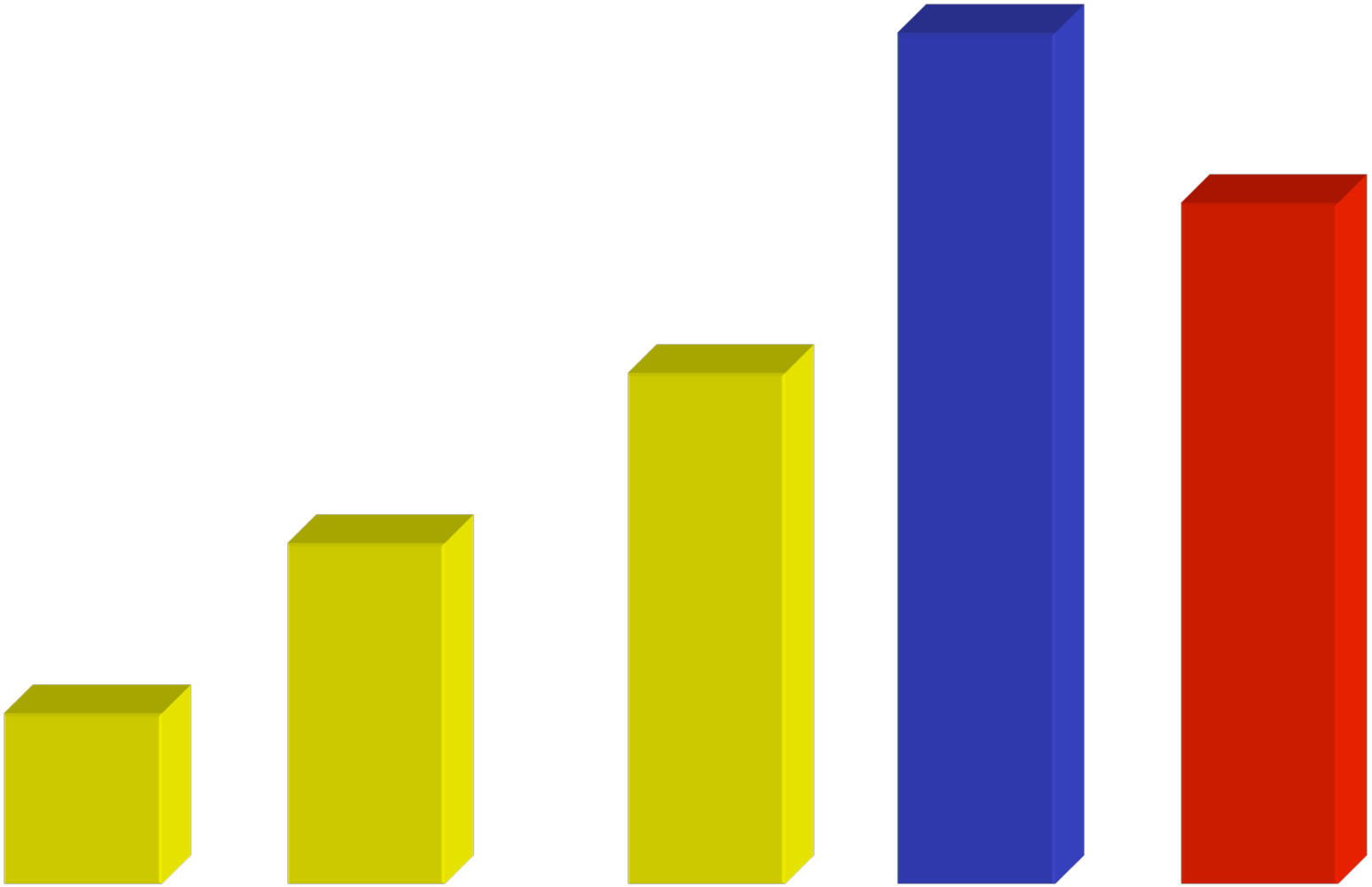
Selection Sort Animation



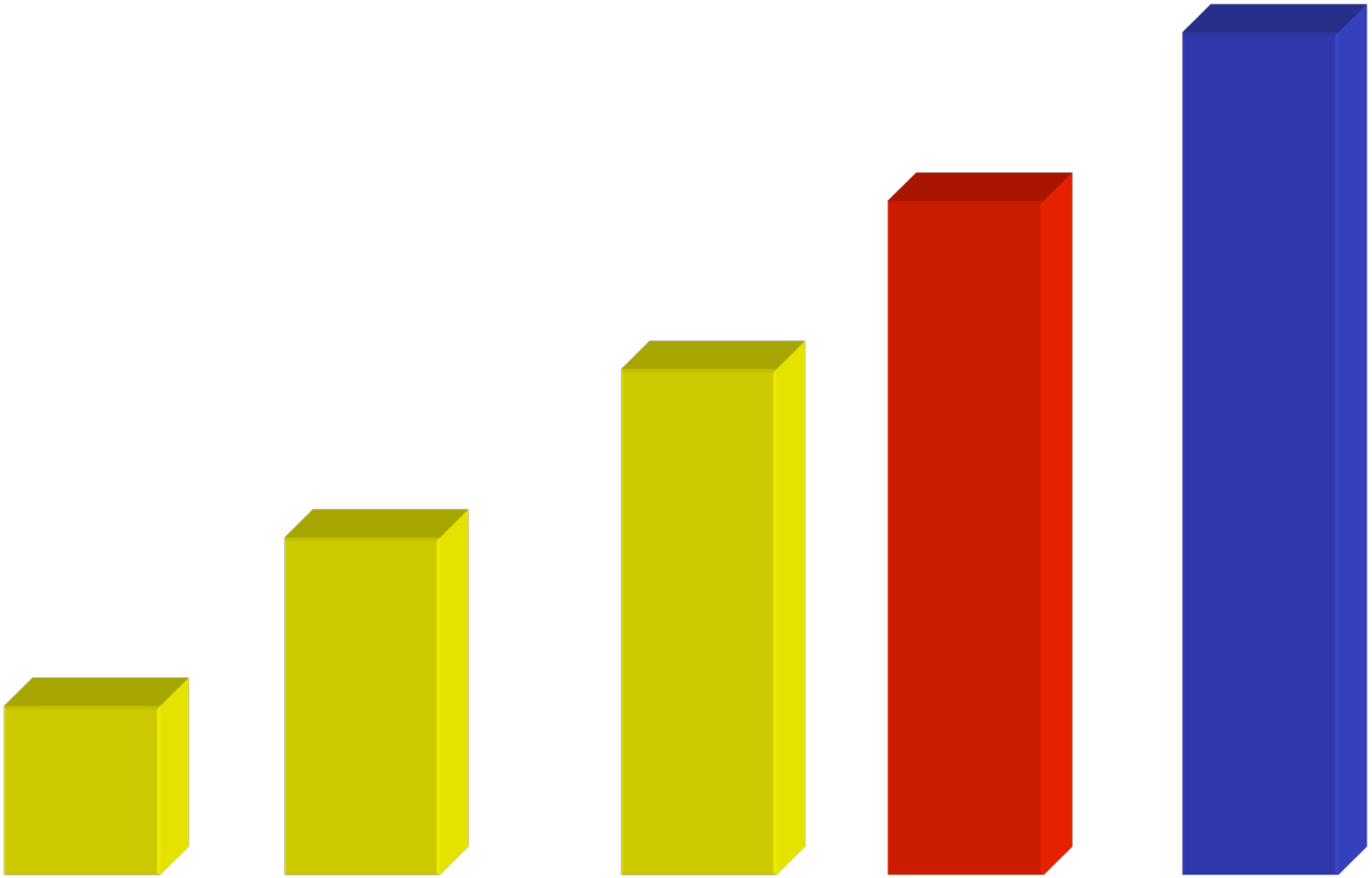
Selection Sort Animation



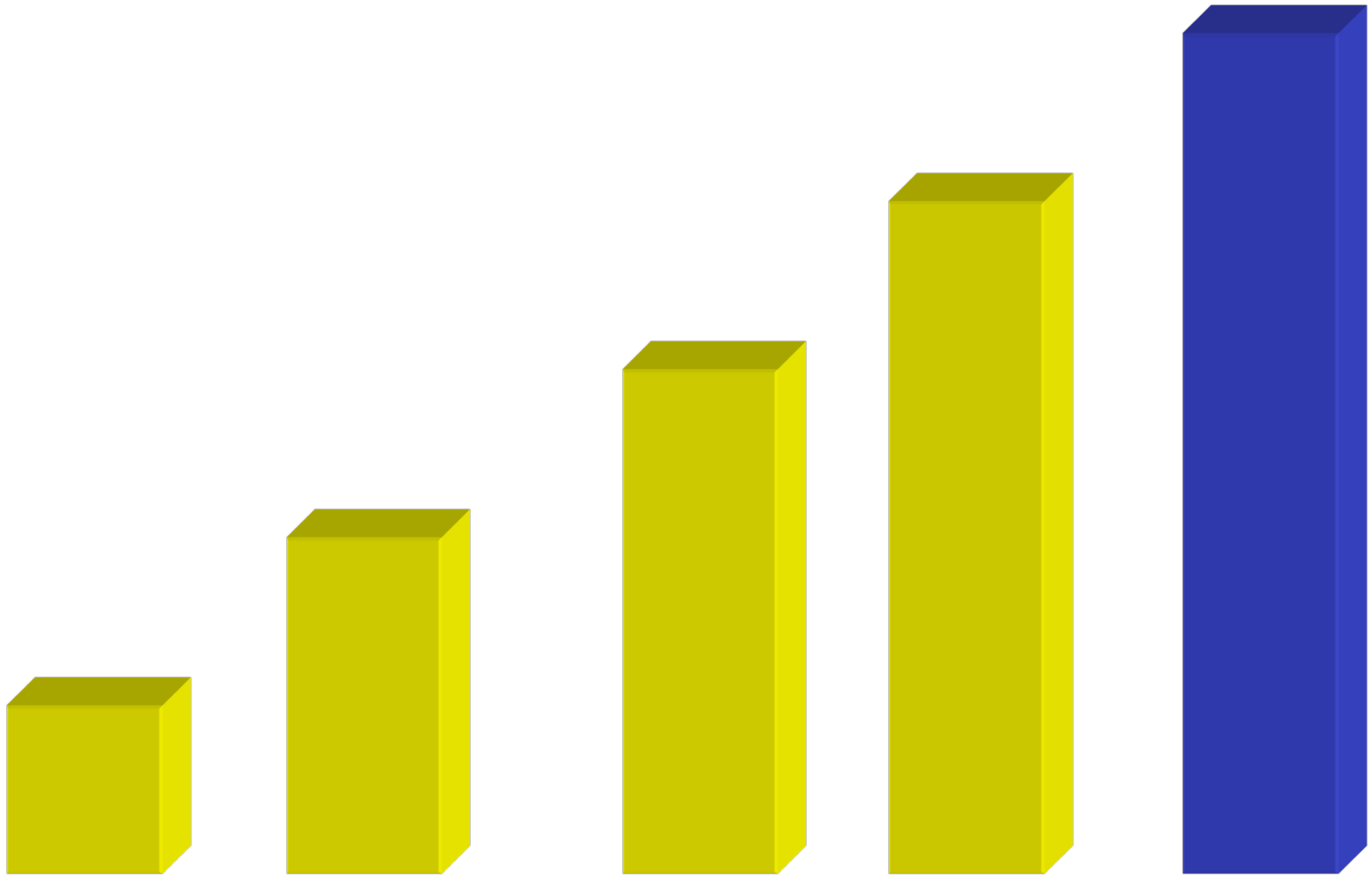
Selection Sort Animation



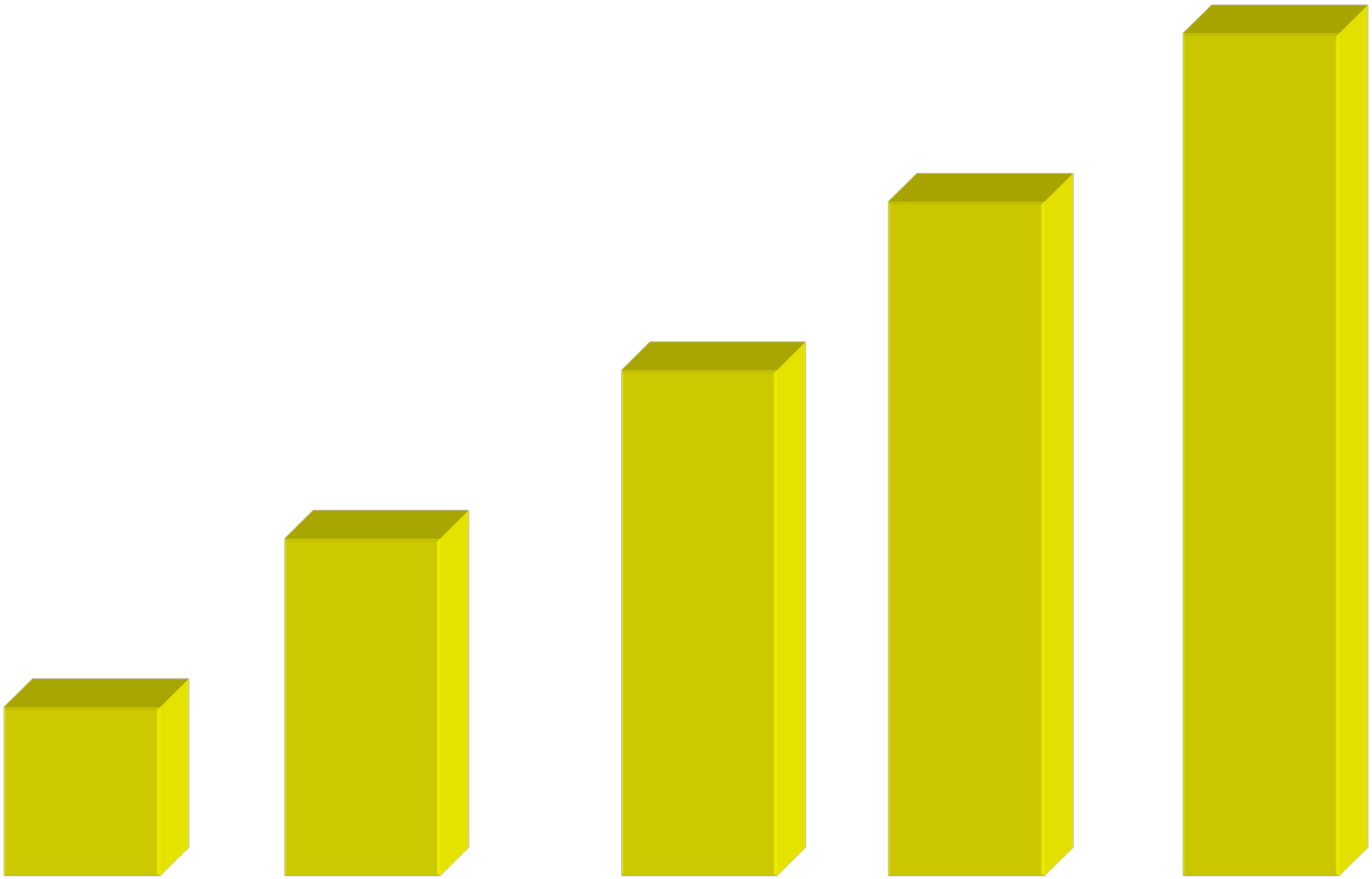
Selection Sort Animation



Selection Sort Animation



Selection Sort Animation



What is the Worst-case Running Time of Selection Sort?

for $k \leftarrow 0$ **to** $n-2$ **do**
    Find the smallest item $A[j]$ between $A[k]$ and $A[n-1]$
    swap $A[k]$ and $A[j]$
end

for $k \leftarrow 0$ **to** $n-2$ **do**
    min $\leftarrow k$
    for $j \leftarrow k+1$ **to** $n-1$ **do**
        if $A[j] < A[min]$ **then** $min \leftarrow j$ **end**
    end
    swap($A[k]$, $A[min]$)
end

What is the Worst-case Running Time of Selection Sort?

$$T(n) = \sum_{k=0}^{n-2} \sum_{j=k+1}^{n-1} \text{cmps} = \sum_{k=0}^{n-2} \sum_{j=k+1}^{n-1} 1 = \sum_{k=0}^{n-2} ((n-1)-(k+1)+1)$$

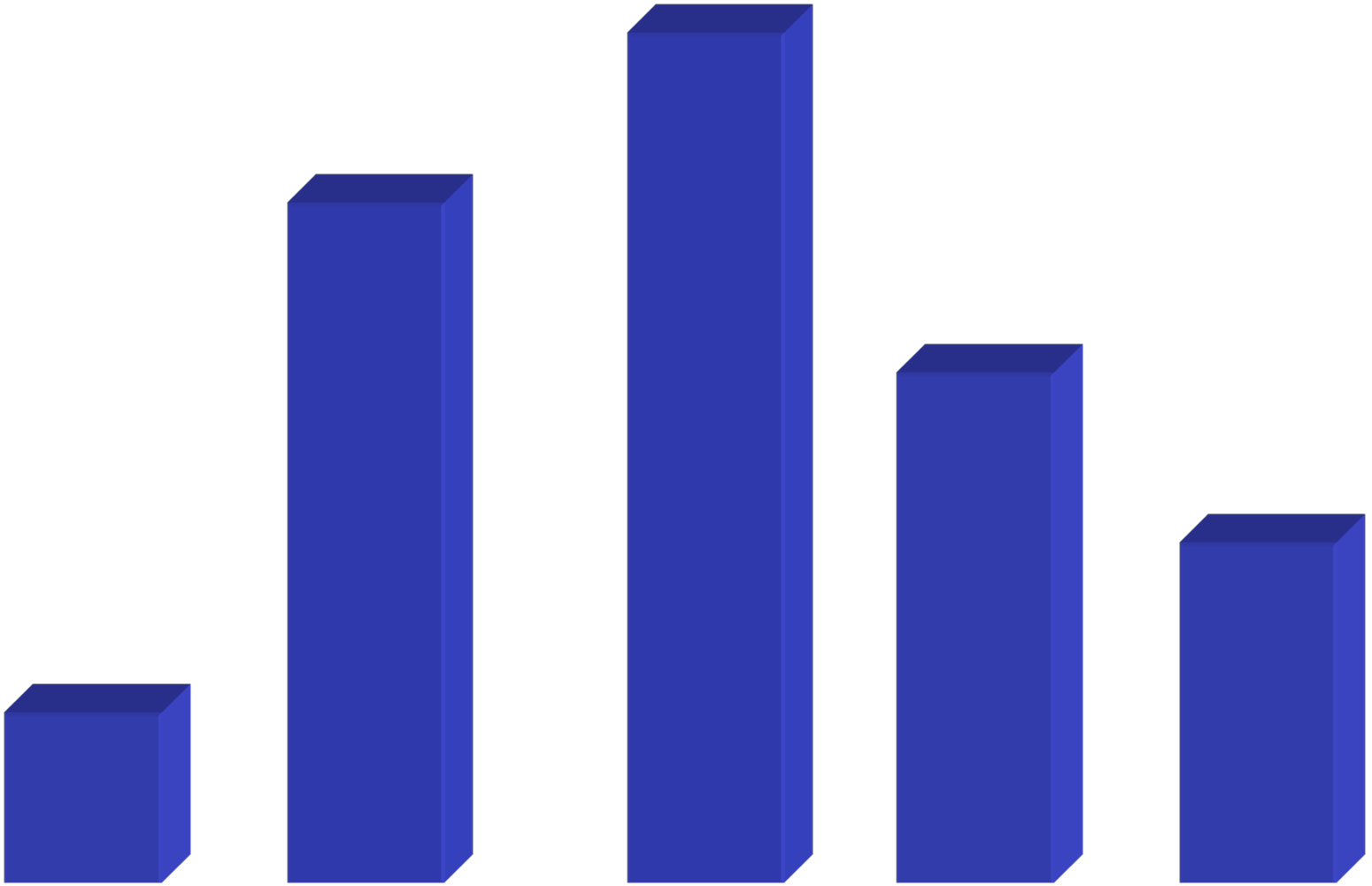
$$T(n) = \sum_{k=0}^{n-2} (n-1-k) = \sum_{k=0}^{n-2} n - \sum_{k=0}^{n-2} 1 - \sum_{k=0}^{n-2} k$$

$$T(n) = (n-1)n - (n-1) - \frac{(n-2)(n-1)}{2}$$

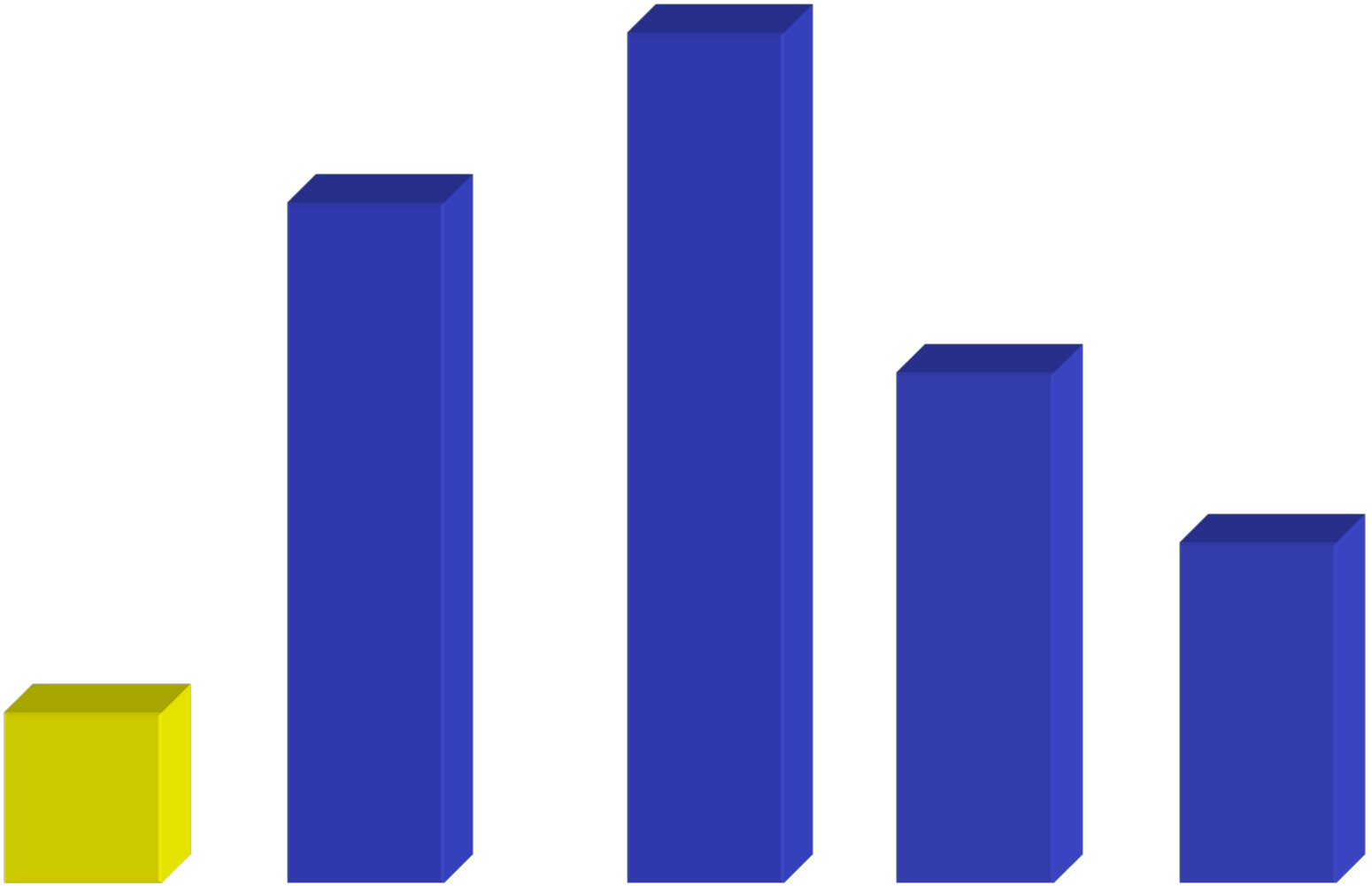
$$T(n) = \frac{2(n-1)n - 2(n-1) - (n-2)(n-1)}{2}$$

$$T(n) = \frac{(n-1)(2n-2-n+2)}{2} = \frac{(n-1)n}{2} \in O(n^2)$$

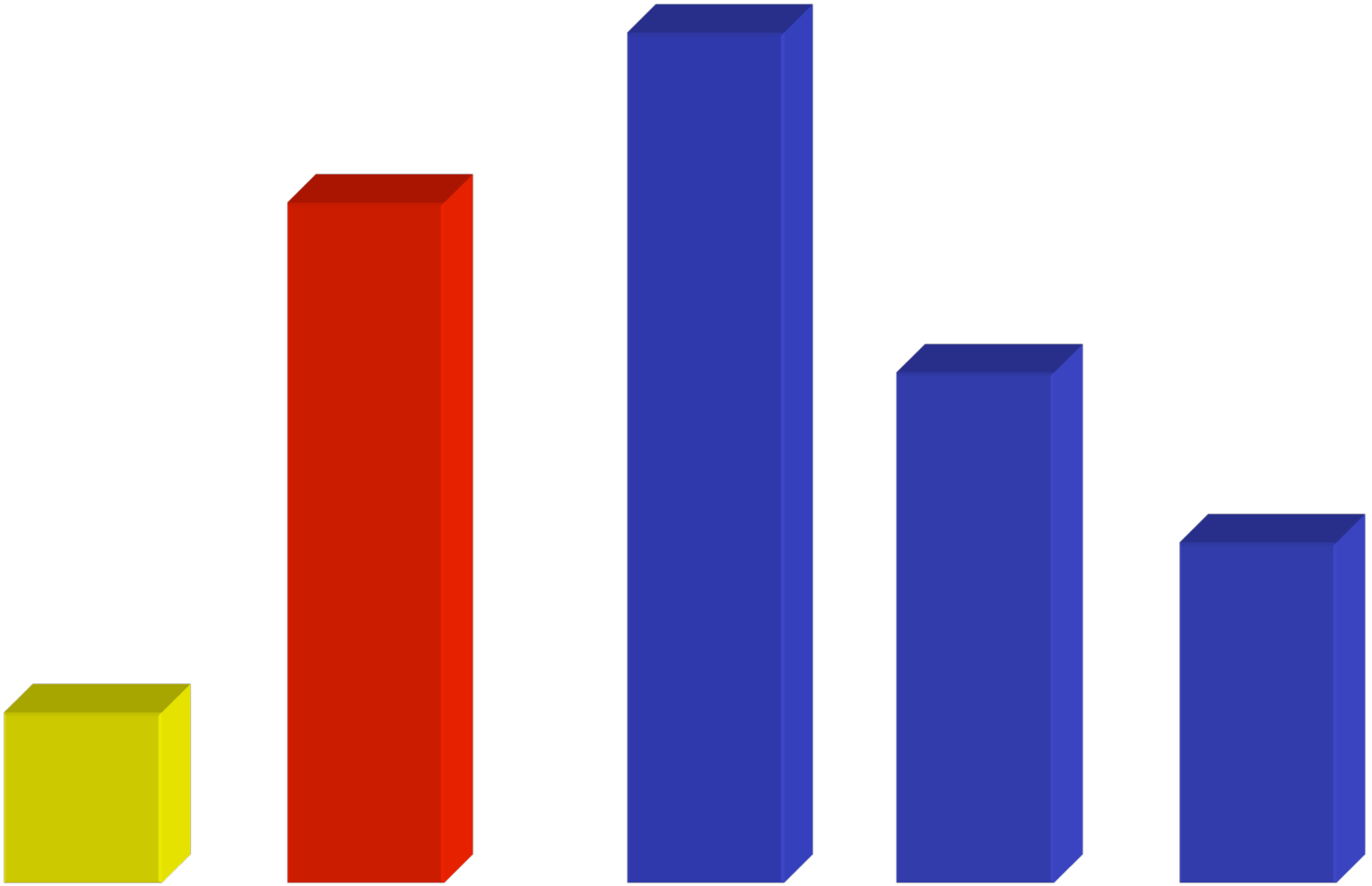
Insertion Sort



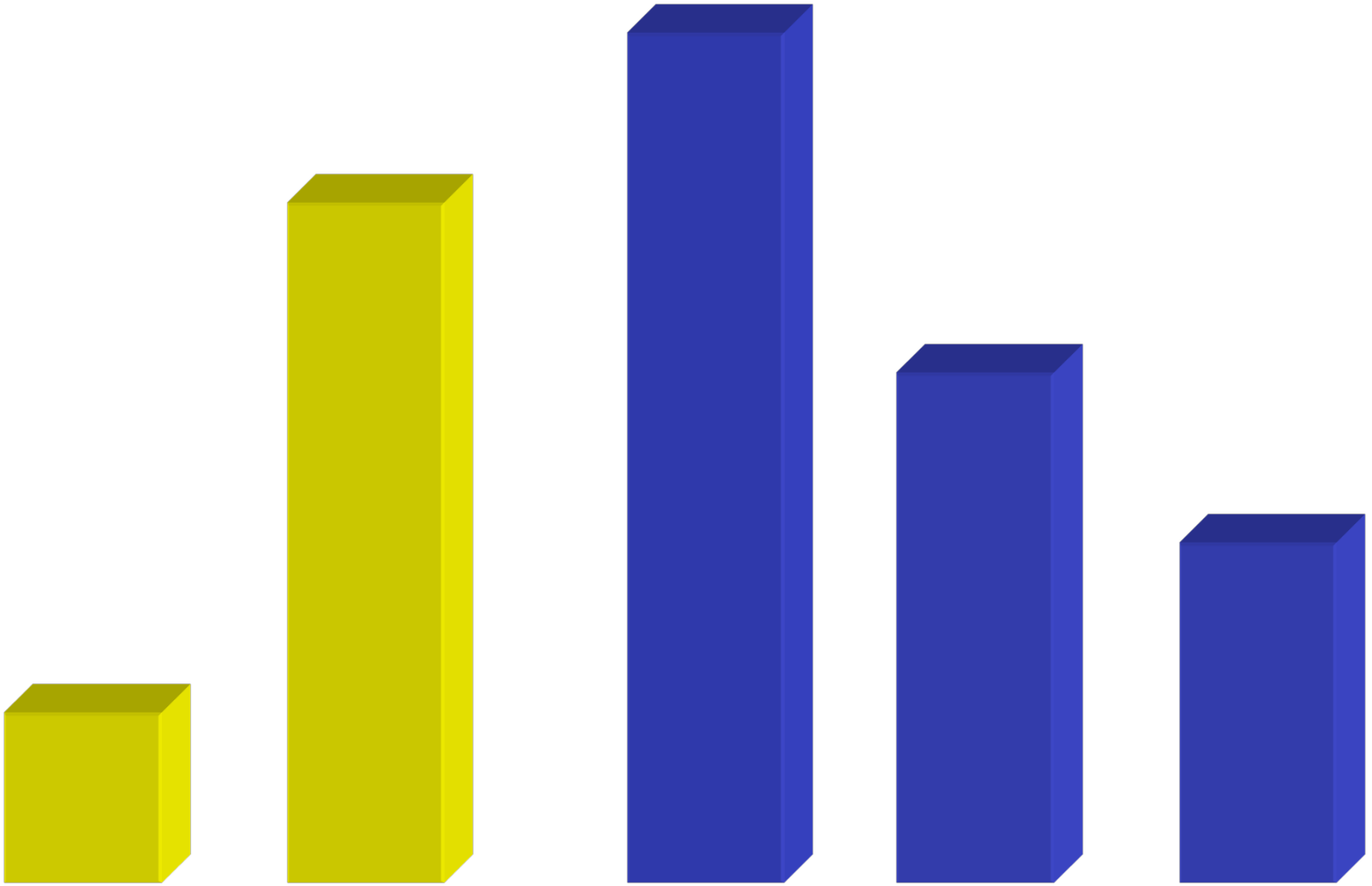
Insertion Sort



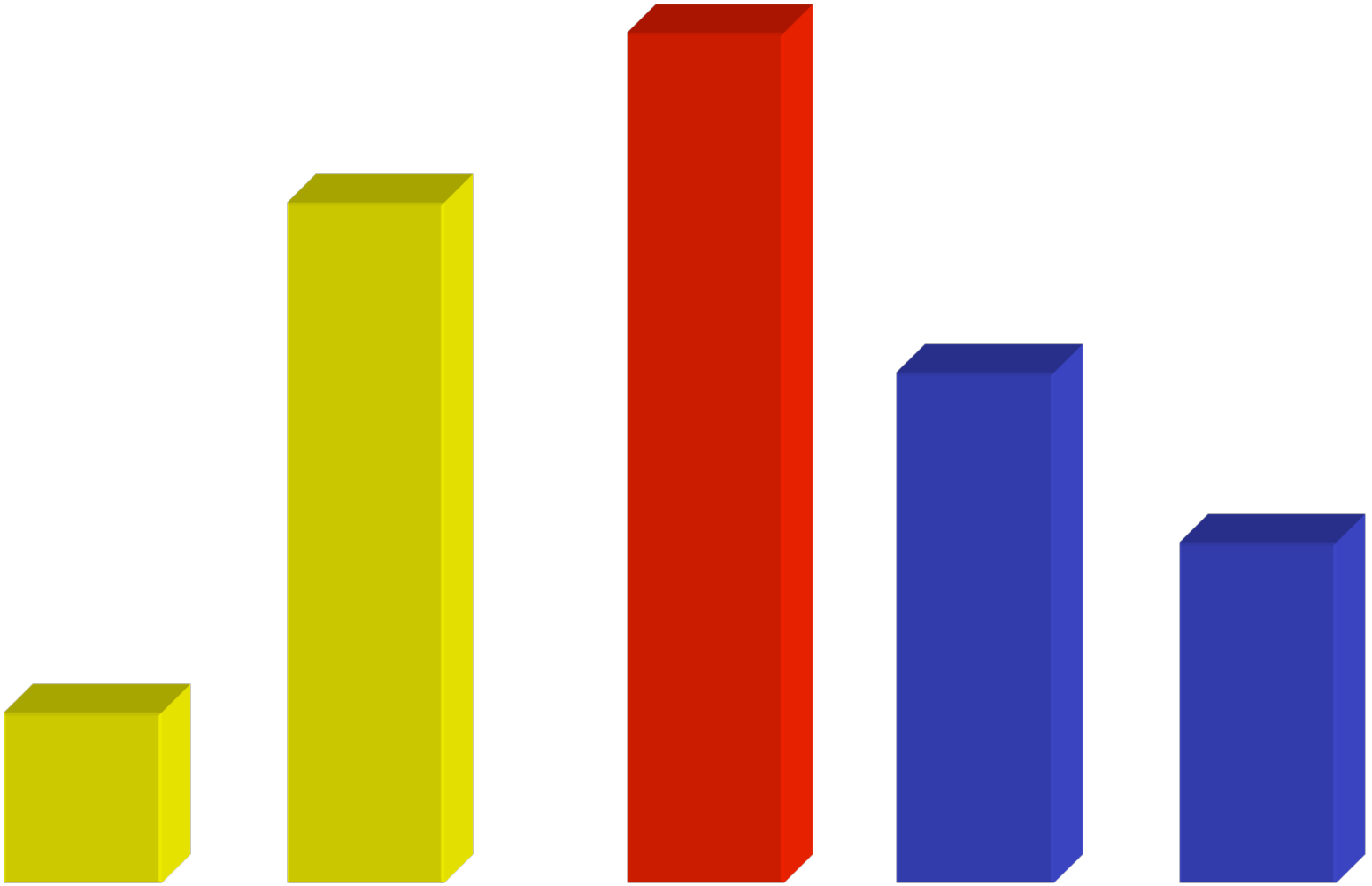
Insertion Sort



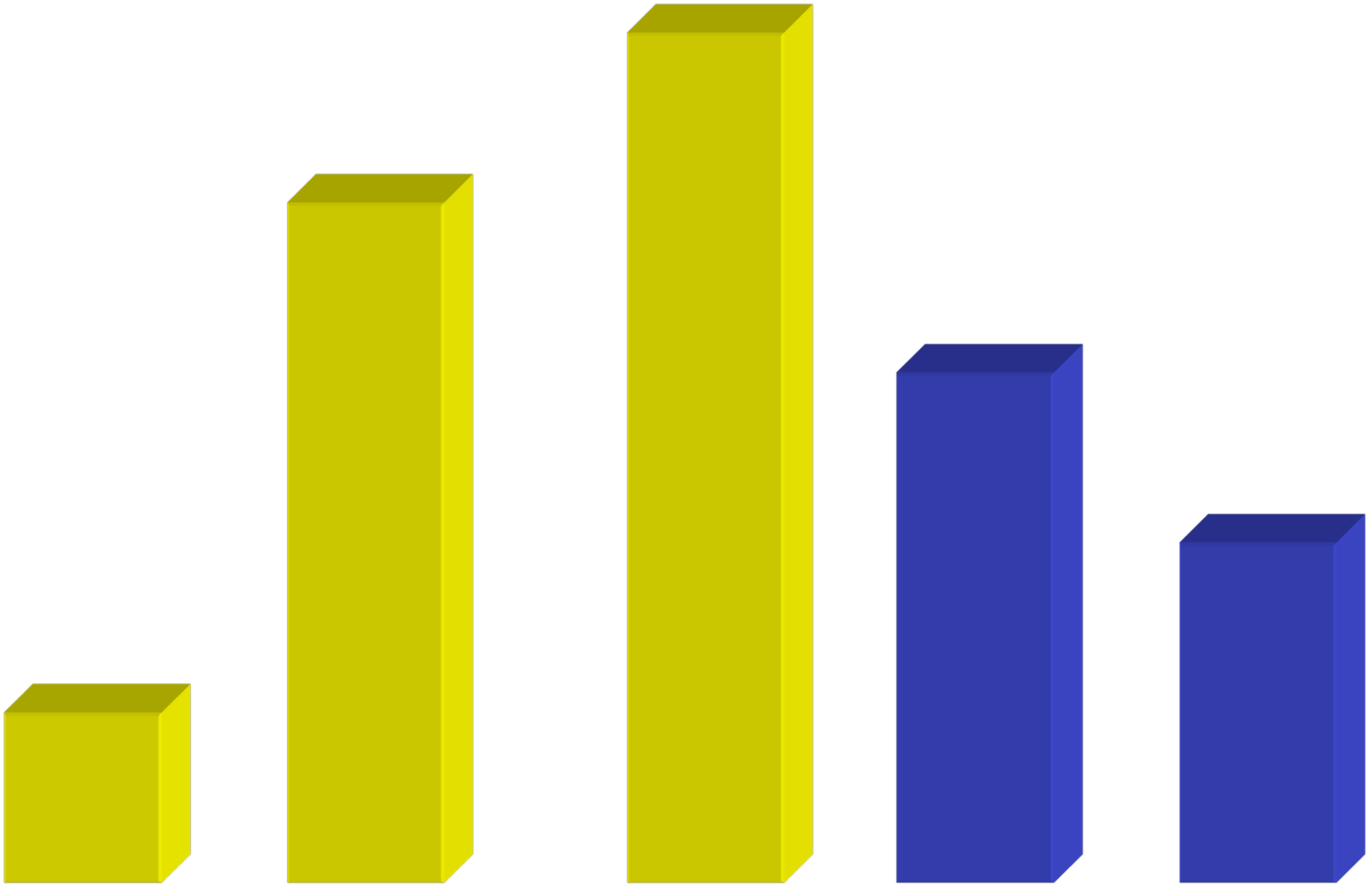
Insertion Sort



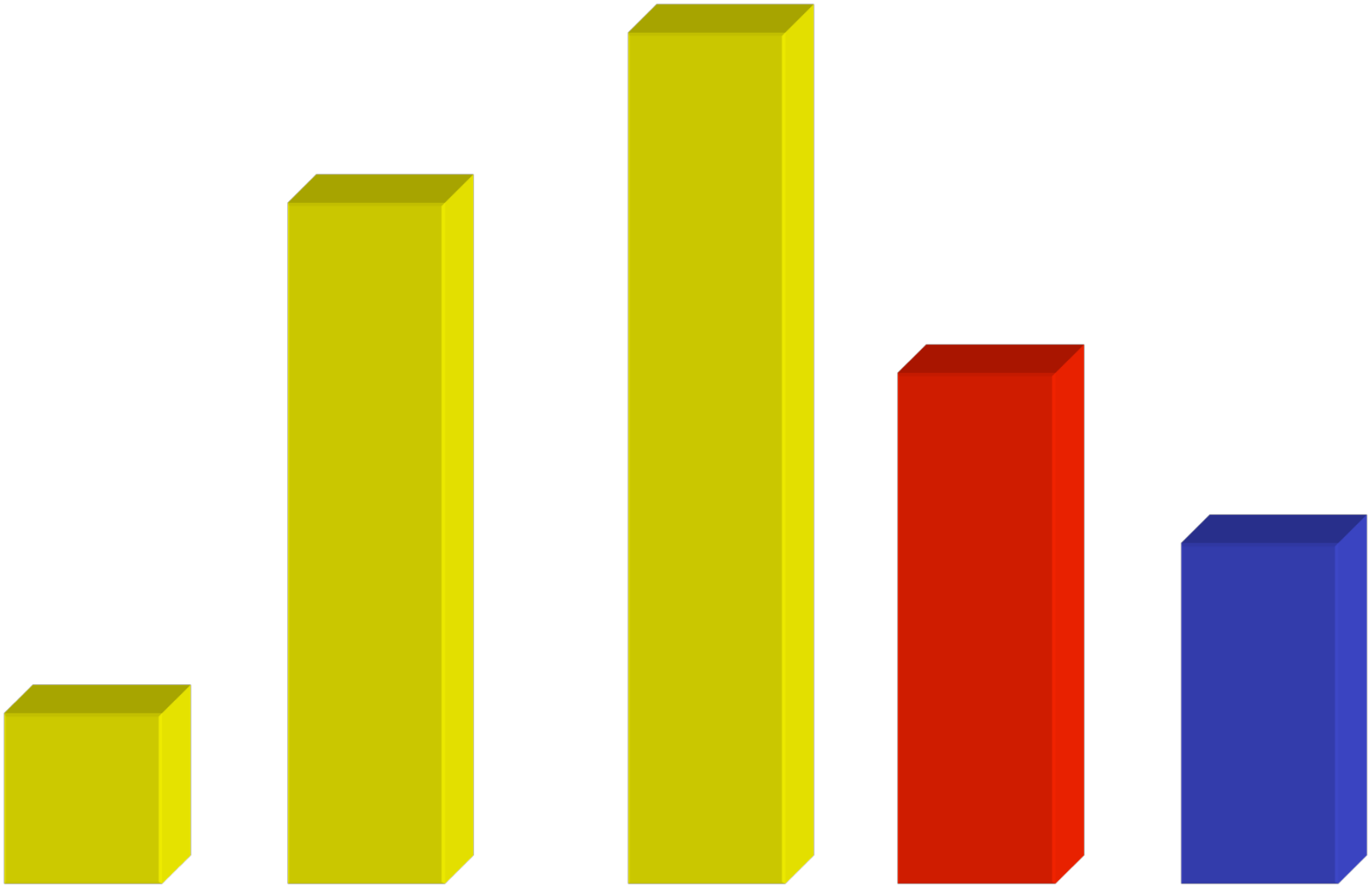
Insertion Sort



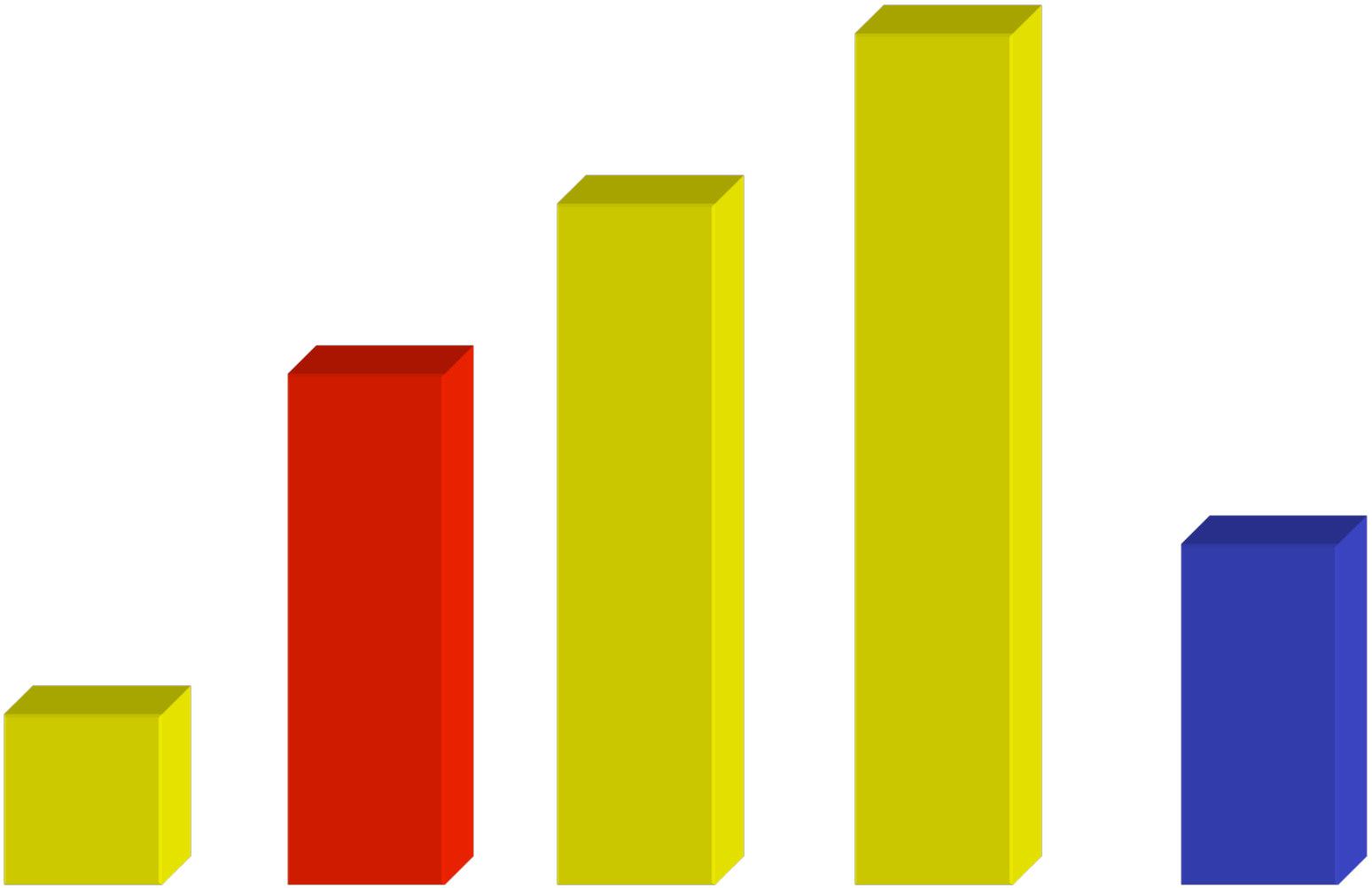
Insertion Sort



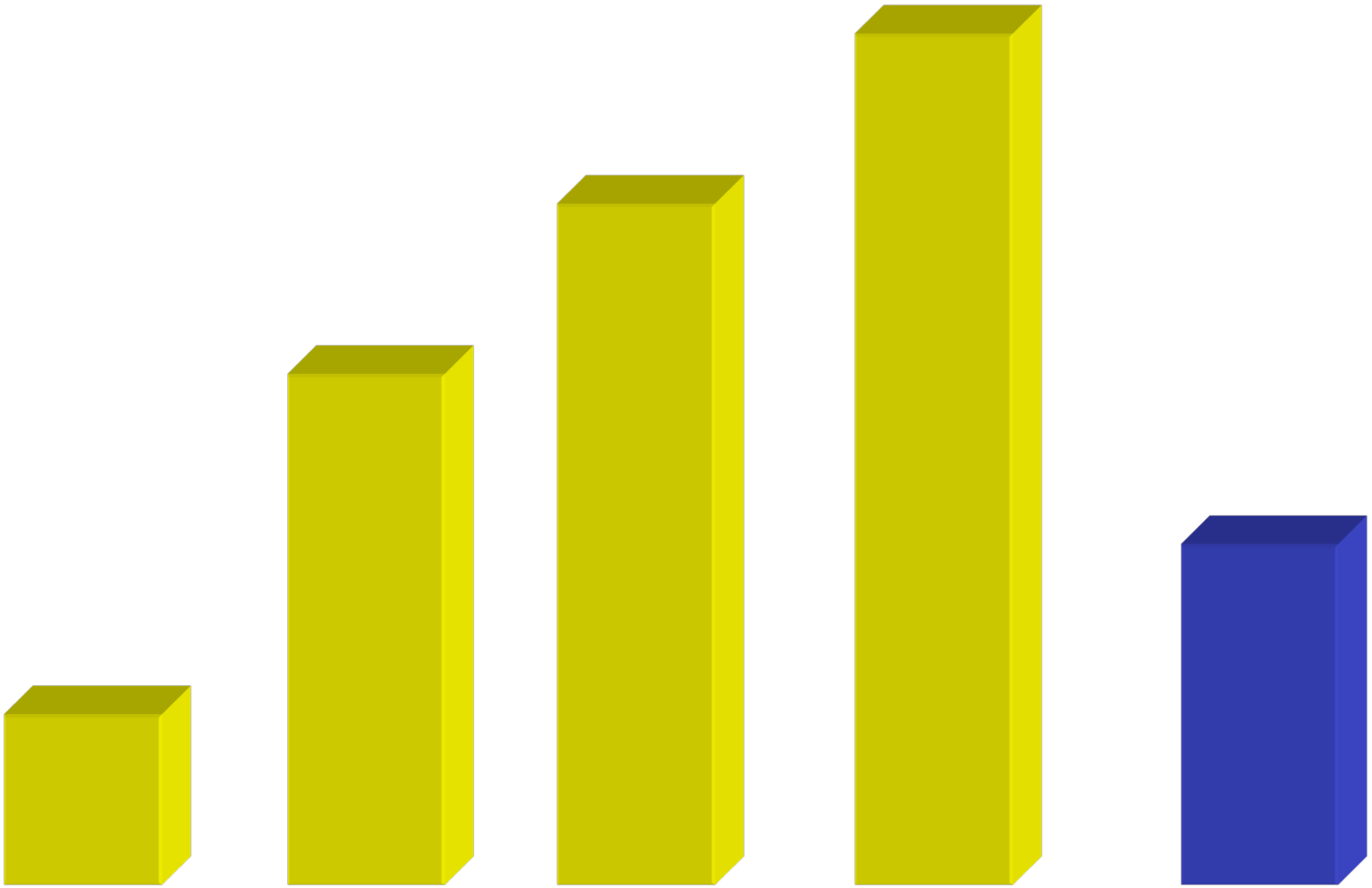
Insertion Sort



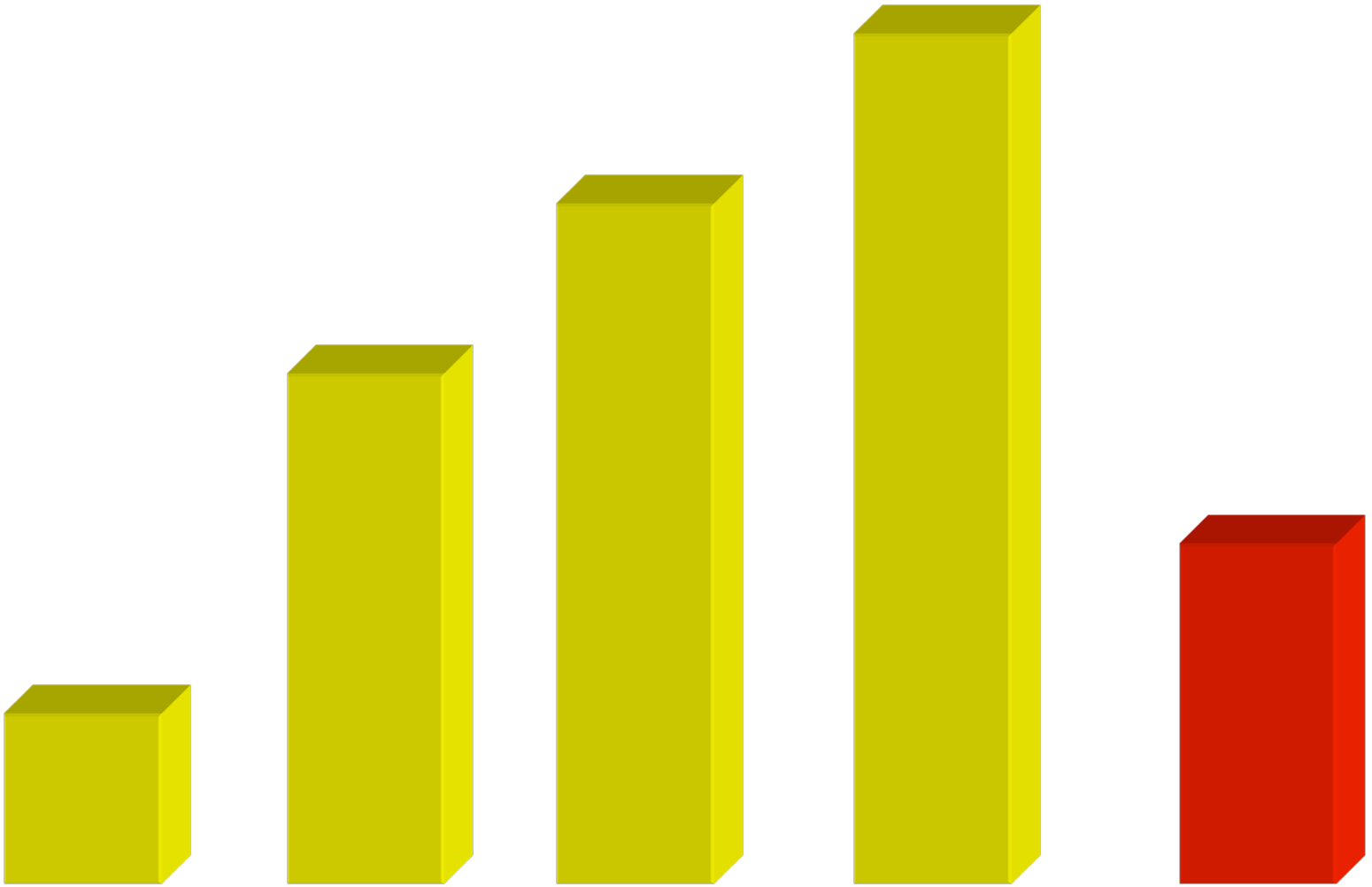
Insertion Sort



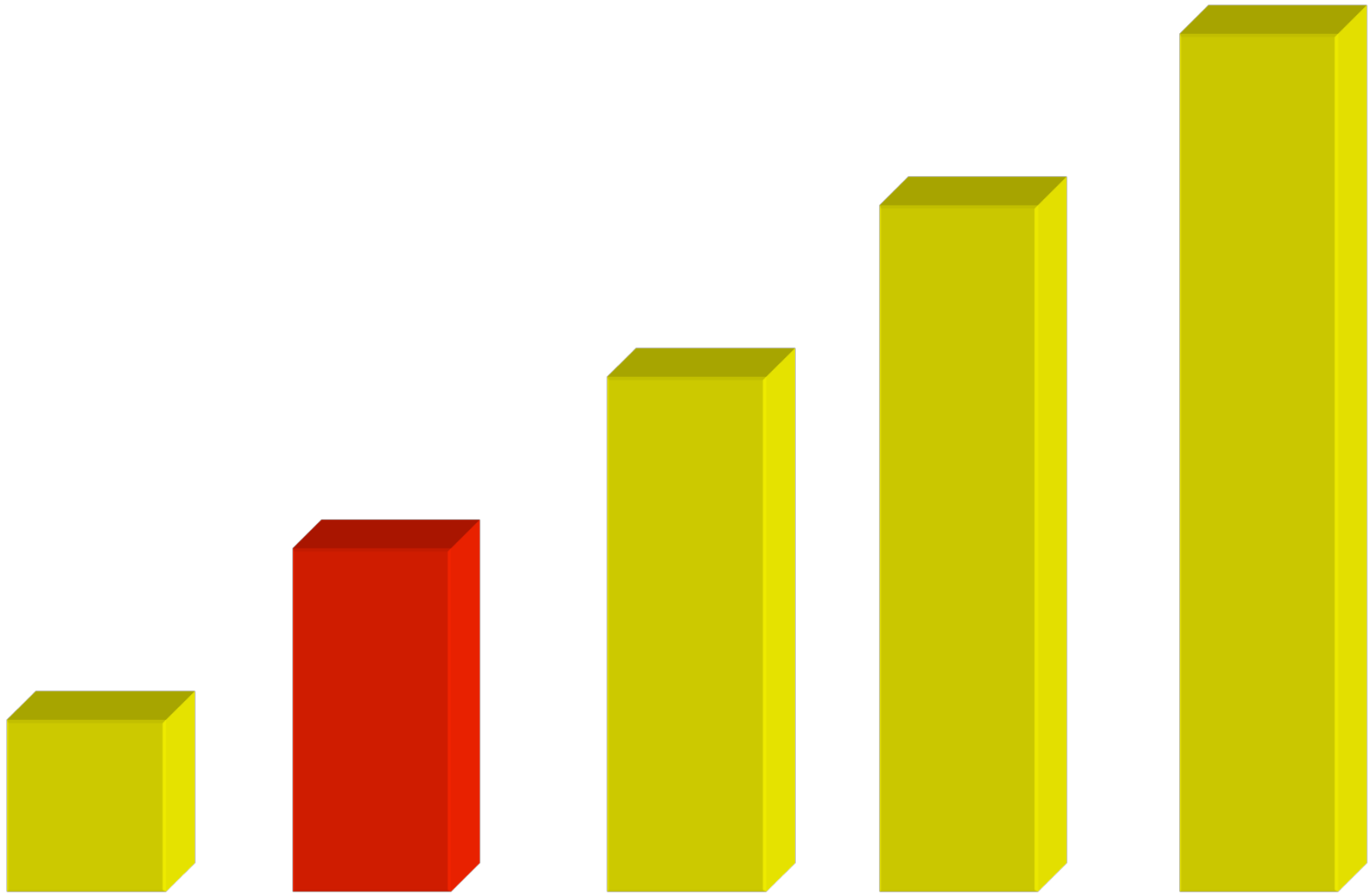
Insertion Sort



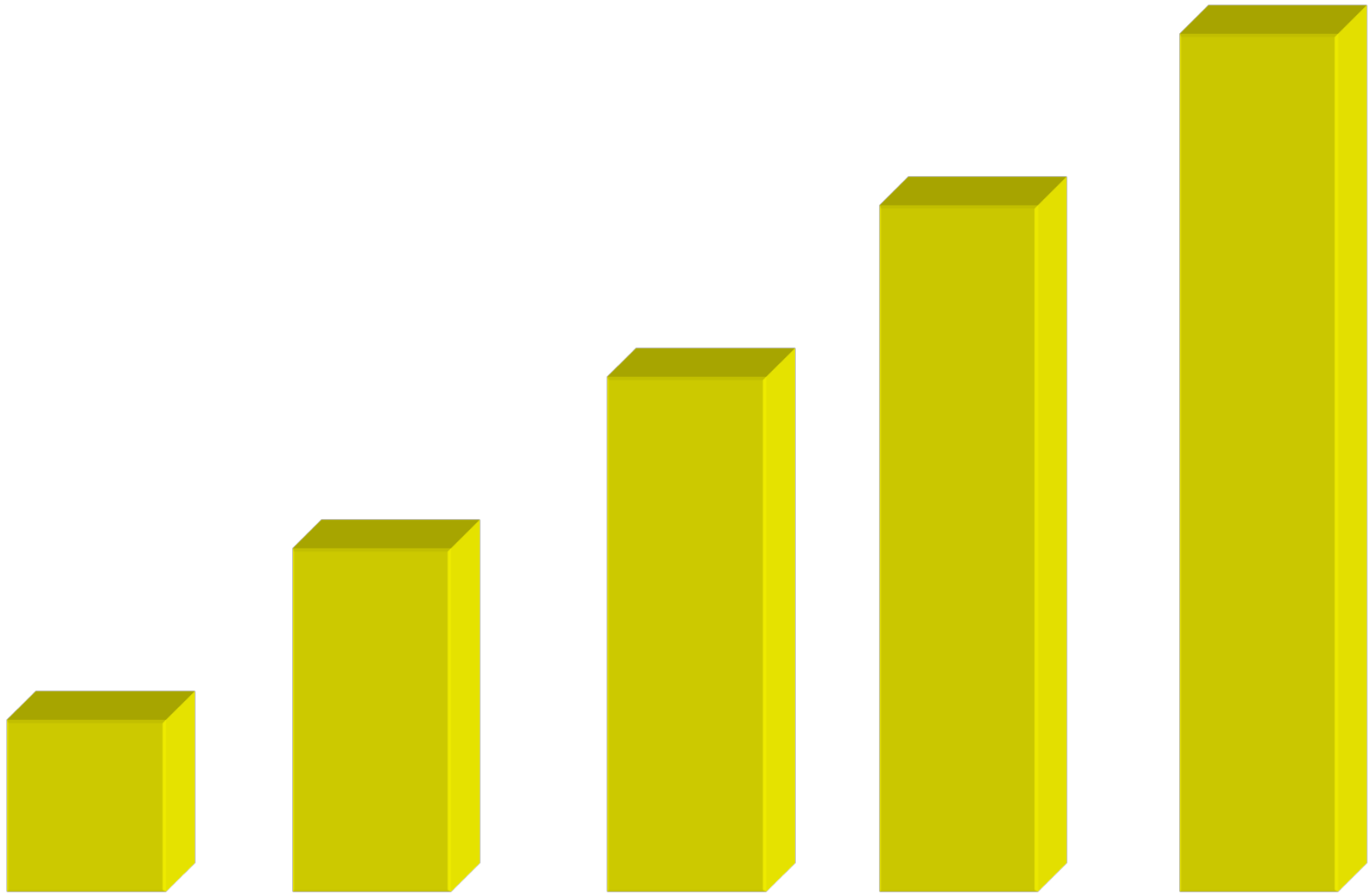
Insertion Sort



Insertion Sort



Insertion Sort



What is the Worst-case Running Time of Insertion Sort?

for $k \leftarrow 2$ **to** n **do**
    Move $A[k]$ forward to position j ≤ k such that
    $A[k] < A[p]$ for $j \leq t < k$ and
    either $A[k] \geq A[j-1]$ or $j = 1$
end

for $k \leftarrow 1$ **to** $n-1$ **do**
    val $\leftarrow A[k]$ $j = k-1$
    while $j \geq 0$ **and** $A[j] >$ val **do**
        $A[j+1] \leftarrow A[j]$ $j = j - 1$
    end
    $A[j+1]$ = val
end

What is the Worst-case Running Time of Insertion Sort?

$$T(n) = \sum_{k=1}^{n-1} \sum_{j=0}^{k-1} cmps = \sum_{k=1}^{n-1} \sum_{j=0}^{k-1} 1 =$$

$$T(n) = \sum_{k=1}^{n-1} k = \frac{(n-1)n}{2} \in O(n^2)$$

Algorithm Design Technique
Divide and Conquer

- Best-known general algorithm design technique
- Some very efficient algorithms are direct results of this technique
  - Mergesort
  - Quicksort
  - Linear selection/median

Algorithm Design Technique
Divide and Conquer

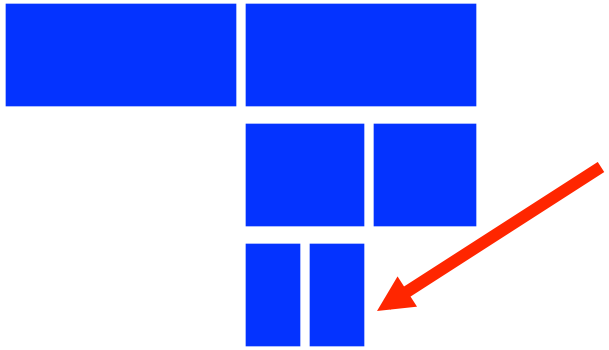
- The problem instance is divided into smaller instances of the same problem, ideally of about the same size (typically n/2)
- The smaller instances are solved (typically recursively, though sometimes a different algorithm is employed when instances become small enough)
- If necessary, the solution obtained for the smaller instances are combined to get a solution to the original instance

Divide and Conquer
Binary Search

- Assume array is sorted

Sorted array

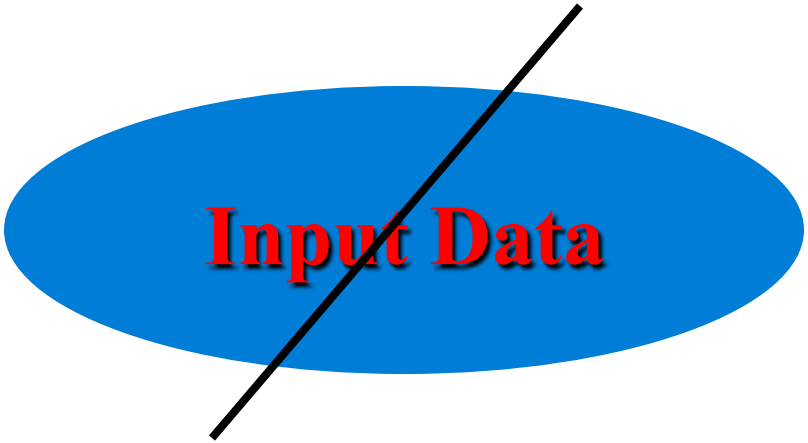
- Compare search value to an element m in the middle of the array; if the search value is less than m, continue searching in left subarray; otherwise right subarray



Subarray of one element; either search value is found or not in the array

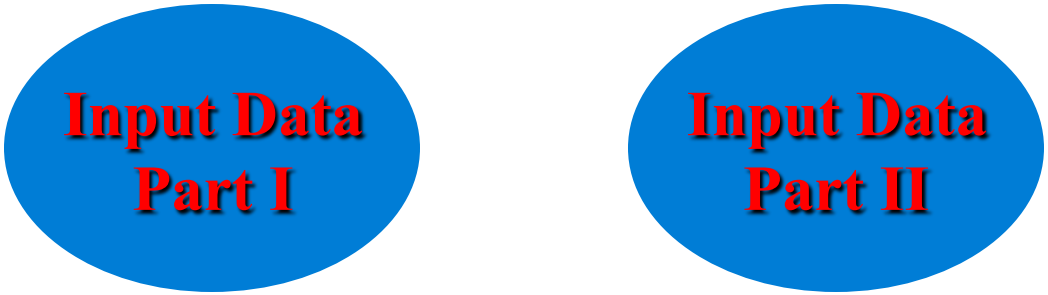
Algorithm Design Technique
Divide and Conquer

1. **Divide:** If the input size is smaller than a certain threshold, solve the problem directly. Otherwise, divide the input data into two or more disjoint subsets.

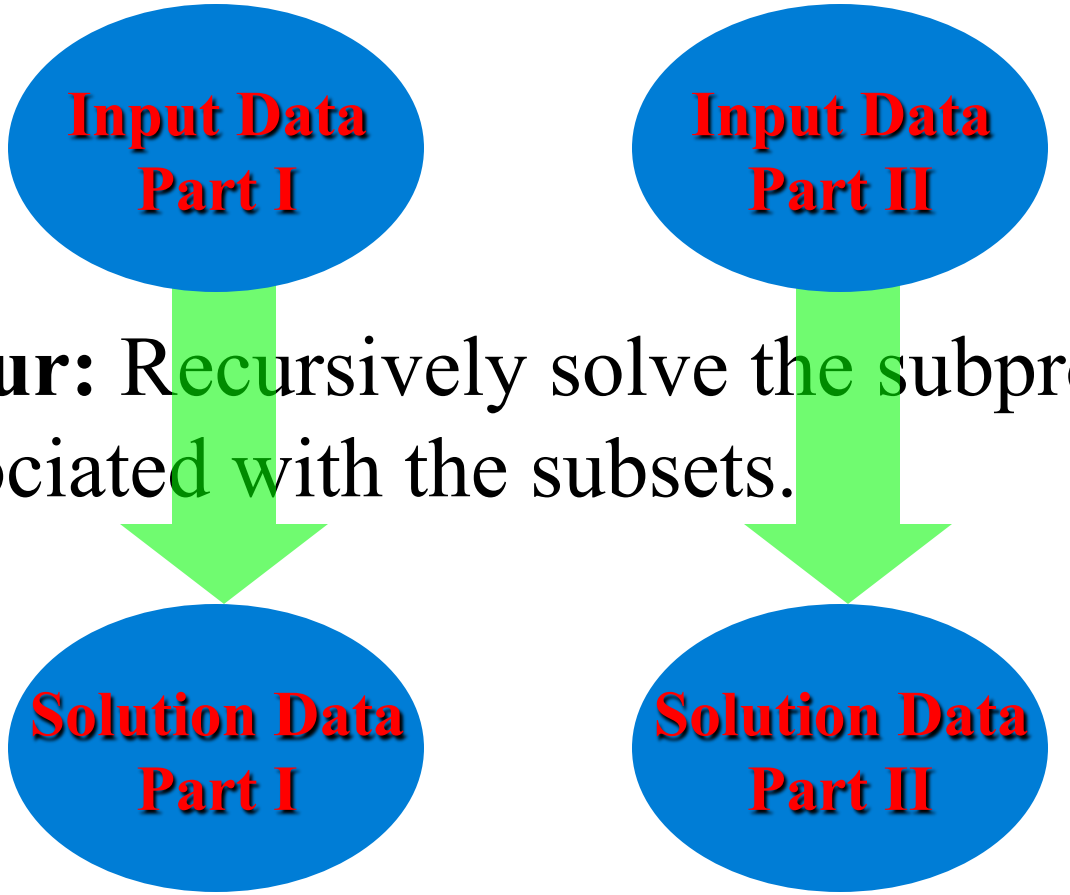


Divide and Conquer

1. **Divide:** If the input size is smaller than a certain threshold, solve the problem directly. Otherwise, divide the input data into two or more disjoint subsets.
2. **Recur:** Recursively solve the subproblems associated with the subsets.



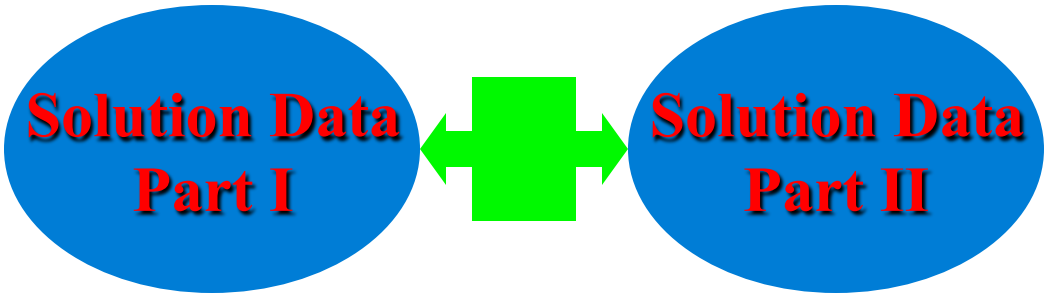
Divide and Conquer



2. **Recur:** Recursively solve the subproblems associated with the subsets.

Divide and Conquer

3. **Conquer:** Take the solutions to the subproblems and merge them into a solution to the original problem.



Divide and Conquer

3. **Conquer:** Take the solutions to the subproblems and merge them into a solution to the original problem.

Merge Sort

Input: A collection of *n* objects (stored in a list, vector, array or sequence) and a comparator defining a total order on these objects

Output: An ordered representation of these objects

➔Apply the Divide-and-Conquer technique to the Sorting problem.

Merge Sort

Let S be a sequence with n elements

1. Divide

- ✓ If S has zero or one element, return S since S is sorted.
- ✓ Otherwise, remove all the elements from S and put them into two sequences S_1 and S_2 such that S_1 and S_2 each contain about half of the elements of S.

Algorithm divide(S_1, S_2, S)

- ✓ S is a sequence containing n elements
- ✓ Let S_1 and S_2 be empty sequences

```
for i ← 0 to ⌊n/2⌋ do
  S1.insertLast(S.removeFirst())
end
for i ← ⌊n/2⌋ + 1 to n-1 do
  S2.insertLast(S.removeFirst())
end
```

Merge Sort

2. **Recur**
   - ✓ Recursively sort sequences S_1 and S_2
3. **Conquer**
   - ✓ Put the elements back into S by ***merging*** the sorted sequences S_1 and S_2 into a sorted sequence.

Merging Two Sorted Sequences

- Assume two sorted sequences S_1 and S_2
- Look up the smallest element of each sequence and compare the two elements
- Remove a smallest element e from these two elements from its sequence and add it to the output sequence S
- Repeat the previous two steps until one of the two sequences is empty
- Copy the remainder of the non-empty sequence to the output sequence

Algorithm merge(S_1, S_2, S)

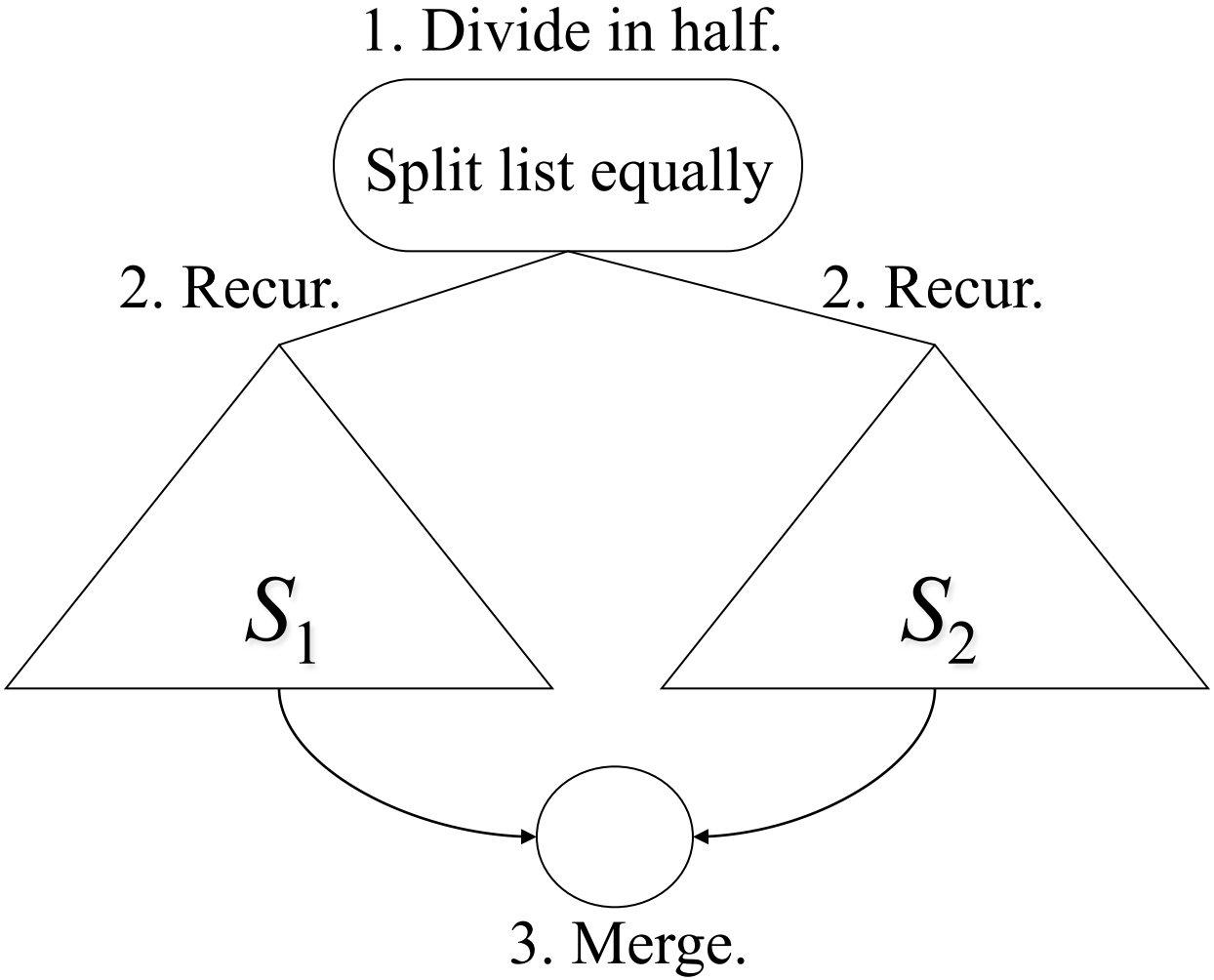
Input: Sequences S_1 and S_2 sorted in non-decreasing order; an empty output sequence S.

Output: Sequence S containing the elements from S_1 and S_2 sorted in non-decreasing order

Algorithm merge(S_1, S_2)

```
while not(S1.isEmpty() or S2.isEmpty()) do
  if S1.first().key() < S2.first().key() then
     S.insertLast(S1.removeFirst())
  else
     S.insertLast(S2.removeFirst()))
  end
end
while not(S1.isEmpty()) do
  S.insertLast(S1.removeFirst())
end
while not(S2.isEmpty()) do
  S.insertLast(S2.removeFirst())
end
return S
```

Merge Sort Algorithm



Algorithm mergeSort(S)

```
if S.size() < 2 then return S
S1, S2 ← divide(S)
S1 ← mergeSort(S1)
S2 ← mergeSort(S2)
S ← merge(S1, S2)
return S
```

Worst-case Running Time of Merge Sort

```
if S.size() < 2 then return S          b
S1, S2 ← divide(S)                     n
S1 ← mergeSort(S1)                     T(n/2)
S2 ← mergeSort(S2)                     T(n/2)
S ← merge(S1, S2)                      n
return S
```

$$T(n) = \begin{cases} b & \text{if } n = 1 \\ 2T(\frac{n}{2}) + cn & \text{otherwise} \end{cases}$$

Solve Recurrence Equation by Repeated Substitution

For $n \geq 3$

$$T(n) = 2T\left(\frac{n}{2}\right) + cn$$

$$= 2\left(2T\left(\frac{1}{2}\frac{n}{2}\right) + c\frac{n}{2}\right) + cn$$

$$= 2\left(2T\left(\frac{n}{2^2}\right) + c\frac{n}{2}\right) + cn$$

$$= 2^2 T\left(n/2^2\right) + cn + cn$$

$$= 2^2 T\left(n/2\right) + 2cn$$

Another Substitution

$$\text{For } n \geq 4: \; T(n) = 2T\left(\frac{n}{2}\right) + cn$$

$$= 2\left(2T\left(\frac{n}{2^2}\right) + c\frac{n}{2}\right) + cn$$

$$= 2\left(2\left(T\left(\frac{1}{2}\frac{n}{2^2}\right) + c\frac{n}{2}\right) + c\frac{n}{2}\right) + cn$$

$$= 2\left(2\left(T\left(\frac{n}{2^3}\right) + c\frac{n}{2}\right) + c\frac{n}{2}\right) + cn$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 2cn + cn$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 3cn$$

Repeated Substitution

For $n \geq i + 1$

$$T(n) = 2^i T\left(\frac{n}{2^i}\right) + icn$$

The Substitution Terminates with T(1)

$$T(n) = b \qquad \text{if } n = 1$$

After how many recursion calls is $n = 1$?

When $2^i = n$ or after $i = \log n$ times.

The Final Steps

Since $T(n) = 2^i T\left(\frac{n}{2^i}\right) + icn$ (for $n \geq i + 1$)

For $i = \log n$

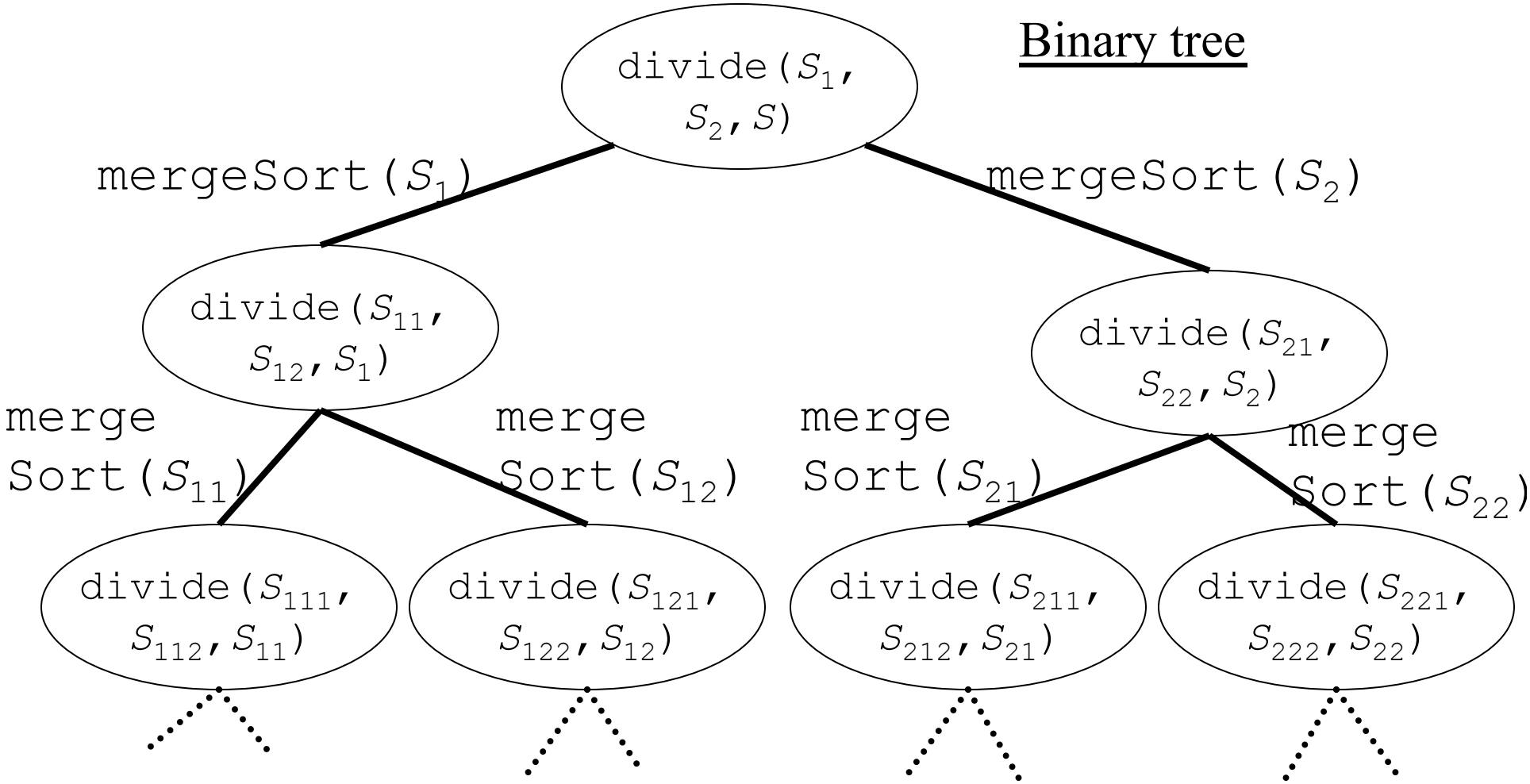
$$T(n) = 2^{\log n} T\left(\frac{n}{2^{\log n}}\right) + cn \log n$$

$$= nT\left(\frac{n}{n}\right) + cn \log n$$

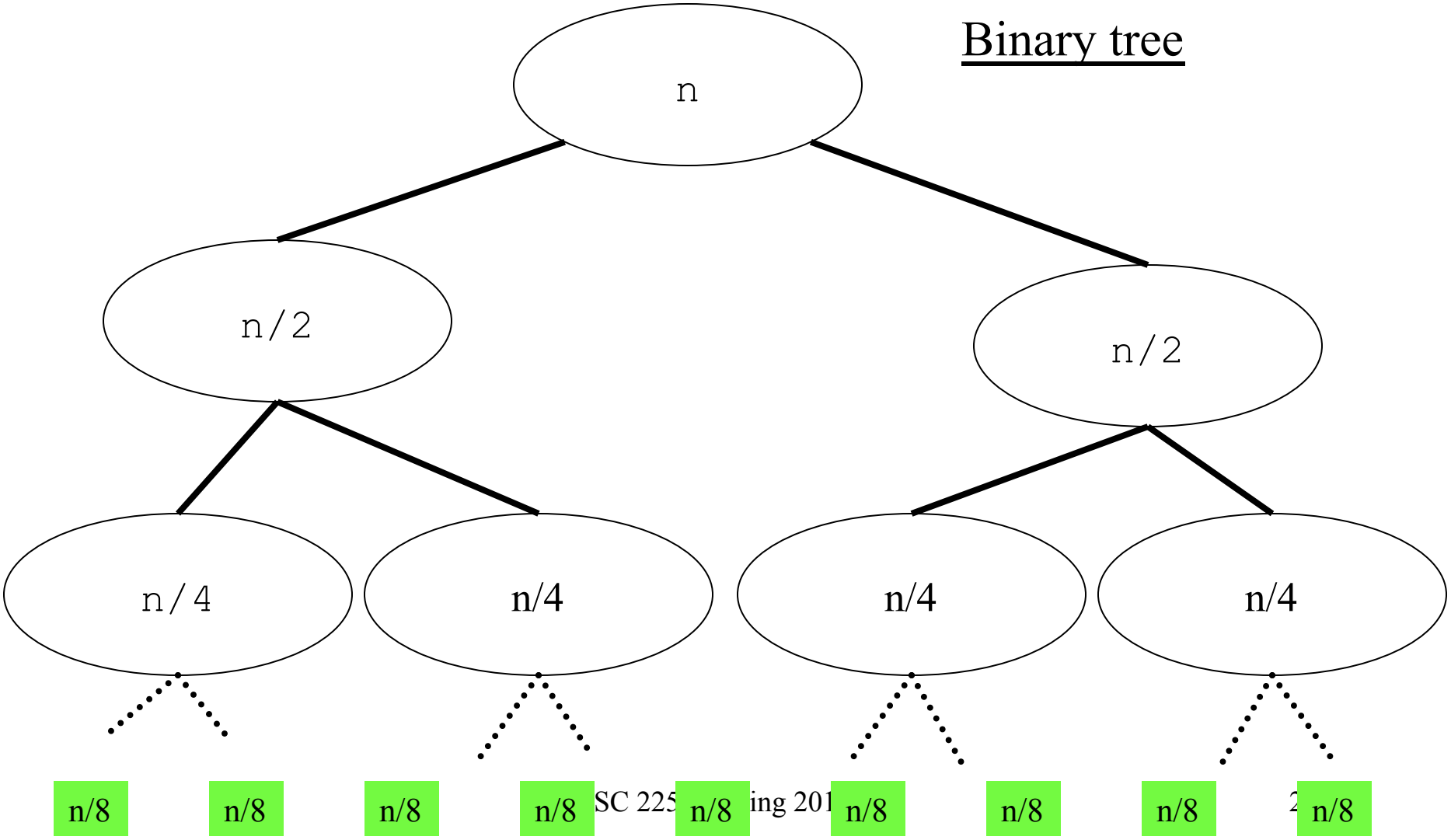
$$= nT(1) + cn \log n$$

$$= nb + cn \log n \quad \text{is } O(n \log n)$$

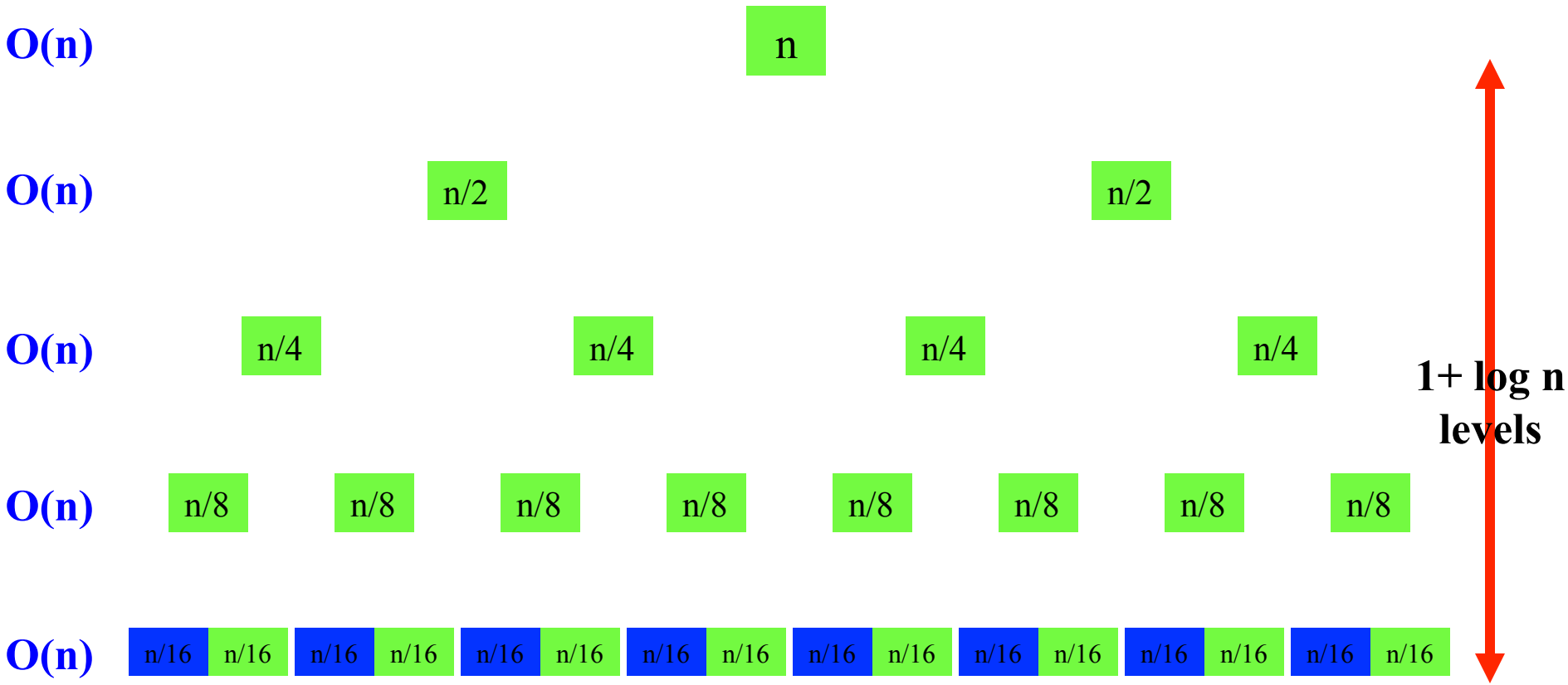
Depth of Recursion of Merge Sort



Depth of Recursion of Merge Sort



Depth of Recursion of Merge Sort



n leaves nodes
n-1 interior nodes

$$T(n) = n \log n \in O(n \log n)$$

What is the Space Complexity of Merge Sort?