ITMO UNIVERSITY

How to Win Coding Competitions: Secrets of Champions

Week 3: Sorting and Search Algorithms
Lecture 2: Insertion sort

Maxim Buzdalov
Saint Petersburg 2016

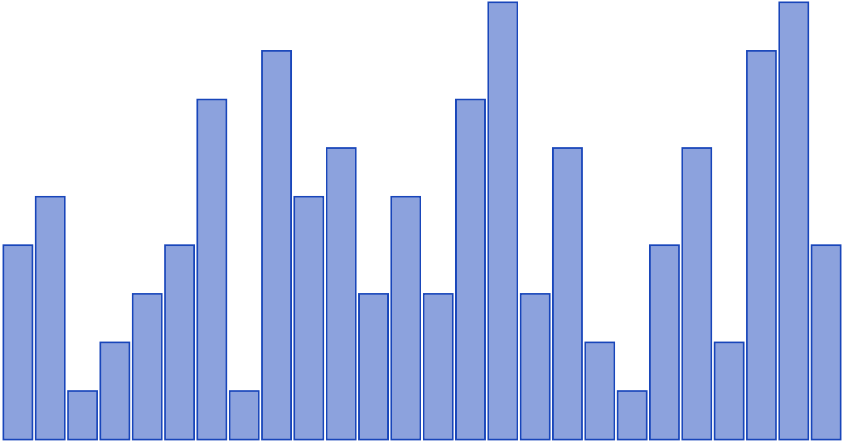
Idea of the algorithm:

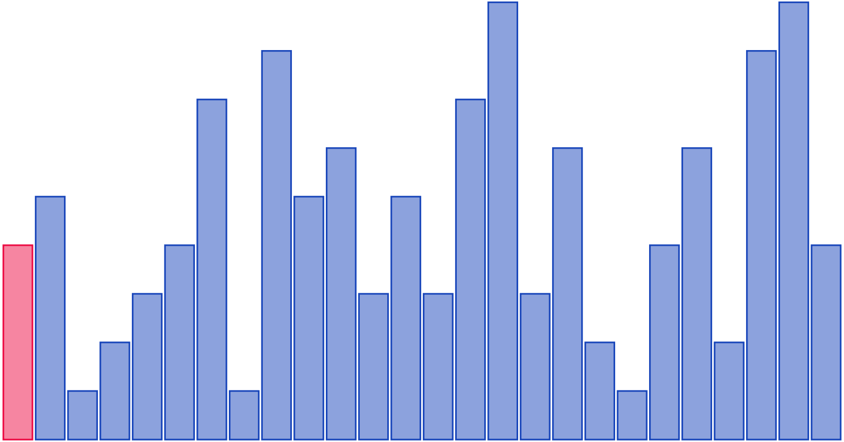
- A sequence of one element is sorted. Let's grow it!
- Increase the sorted part, step by step, until everything is sorted
  - Take the element adjacent to the sorted part
  - Push it backwards, step by step, while it is greater than the predecessor

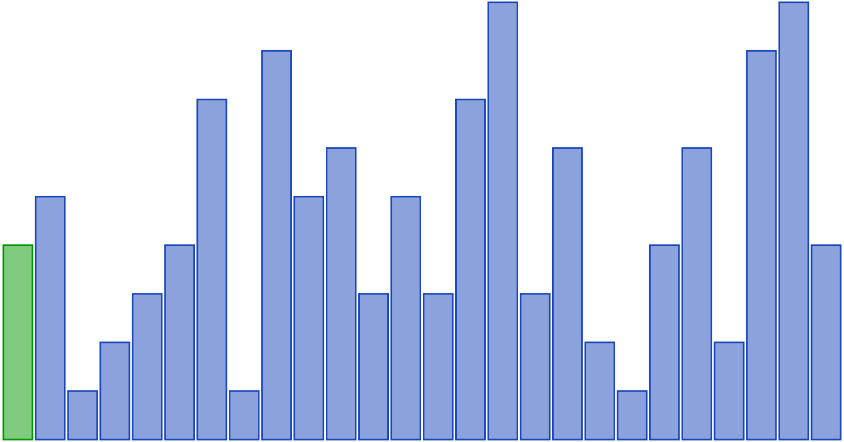
Idea of the algorithm:

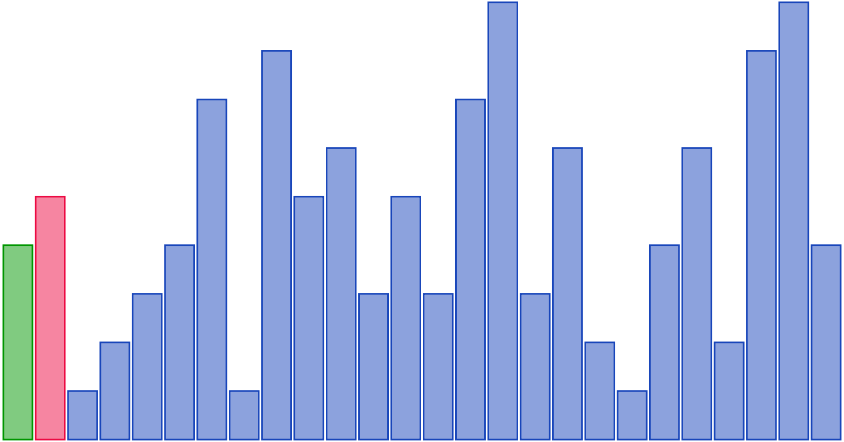
- A sequence of one element is sorted. Let's grow it!
- Increase the sorted part, step by step, until everything is sorted
  - Take the element adjacent to the sorted part
  - Push it backwards, step by step, while it is greater than the predecessor

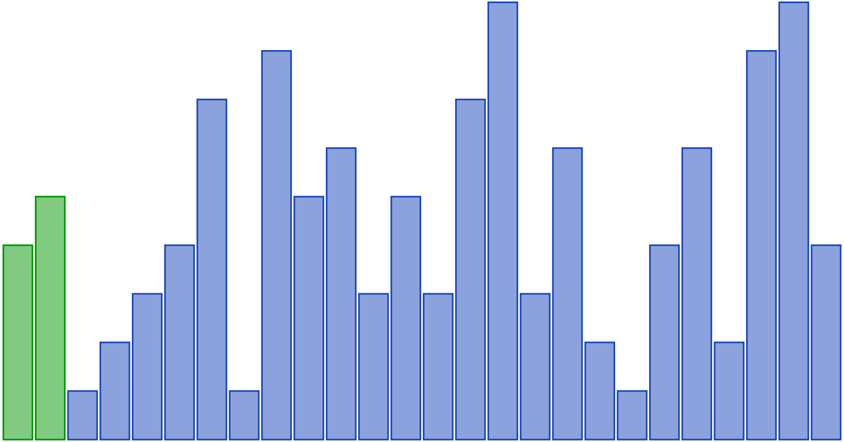
procedure $\textsc{InsertionSort}(A, \leq)$
  for i **from** 1 **to** $|A|$ **by** 1 **do**
    $k \leftarrow i$
    while $(k > 1)$ **and not** $(A[k-1] \leq A[k])$ **do**
      $A[k-1] \Leftrightarrow A[k]$
      $k \leftarrow k - 1$
    end while
  end for
end procedure

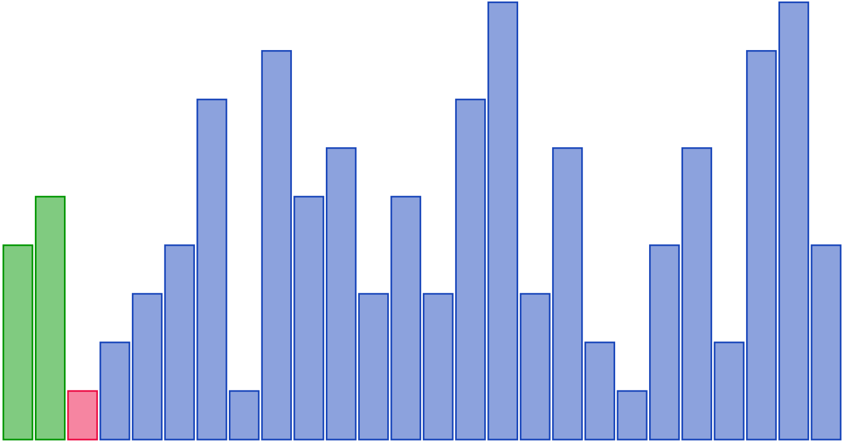


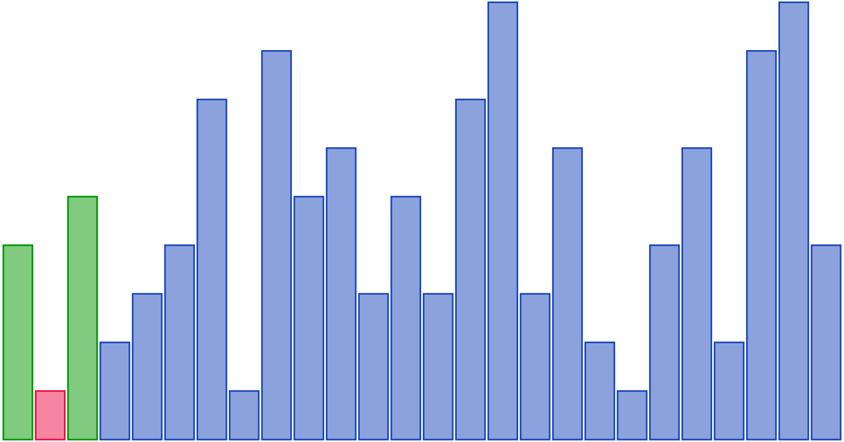


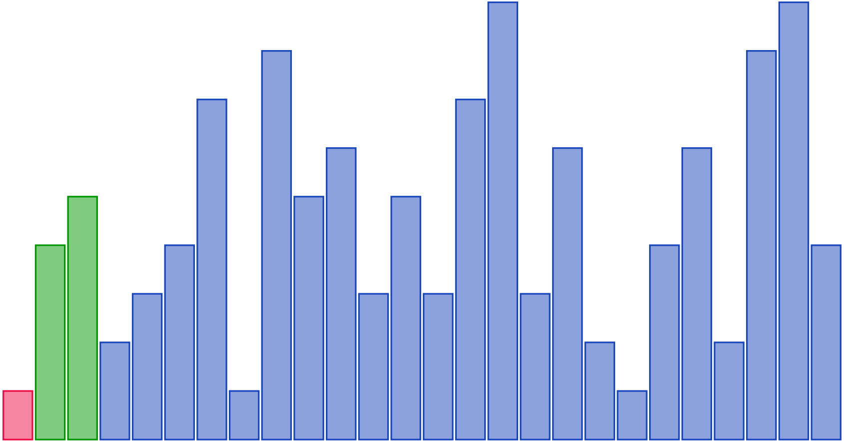


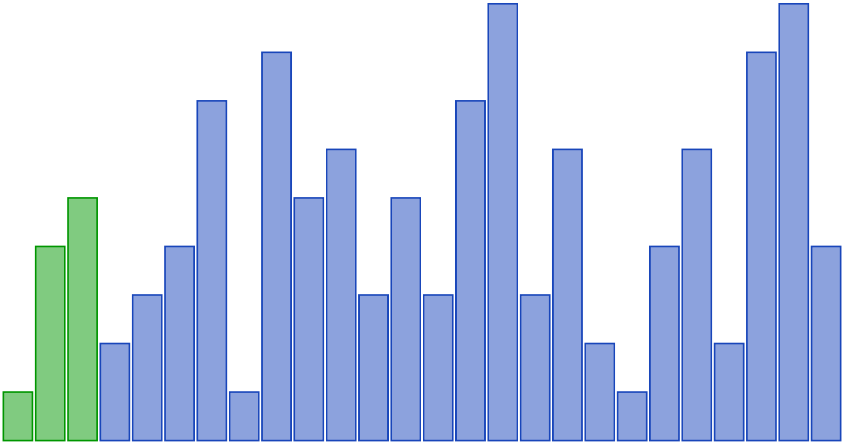


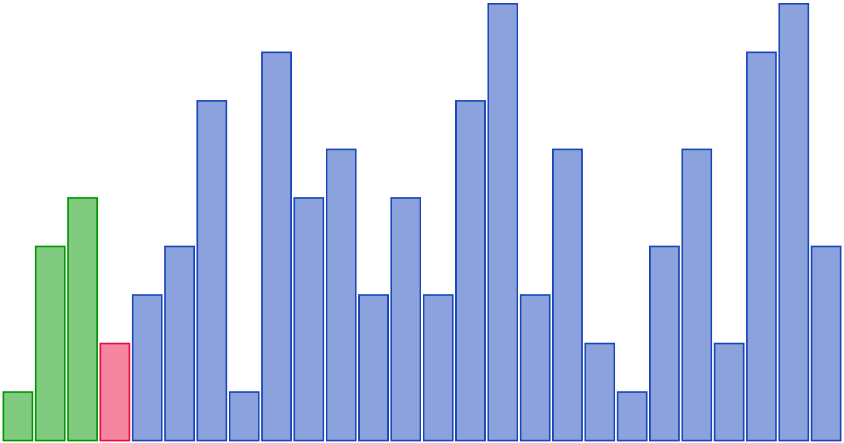


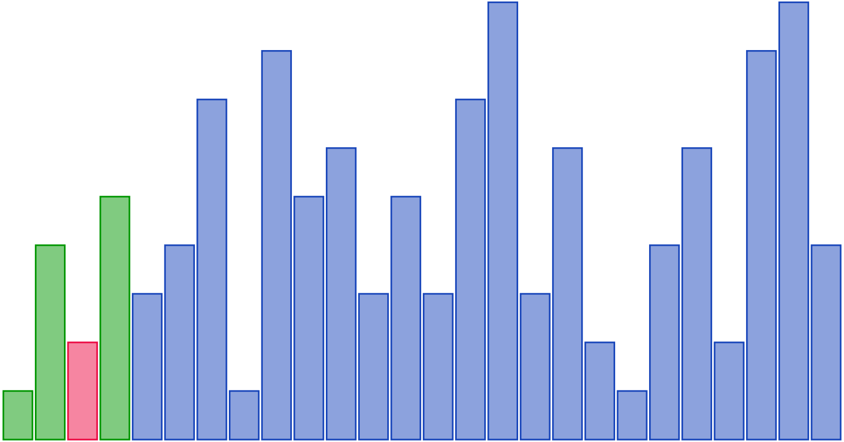


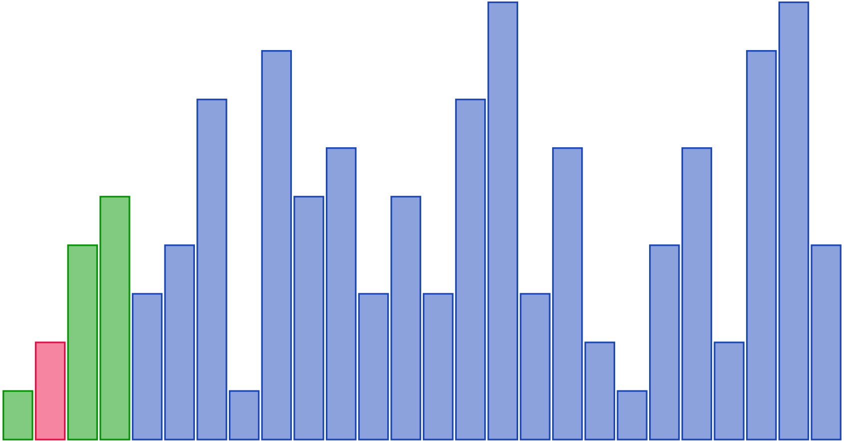


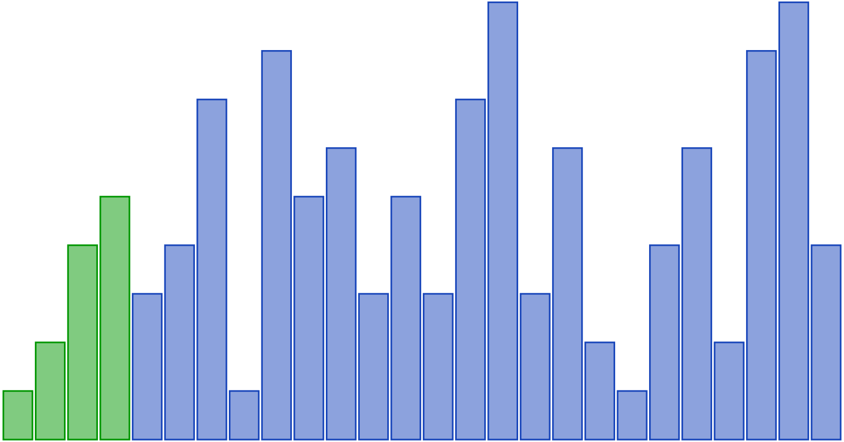


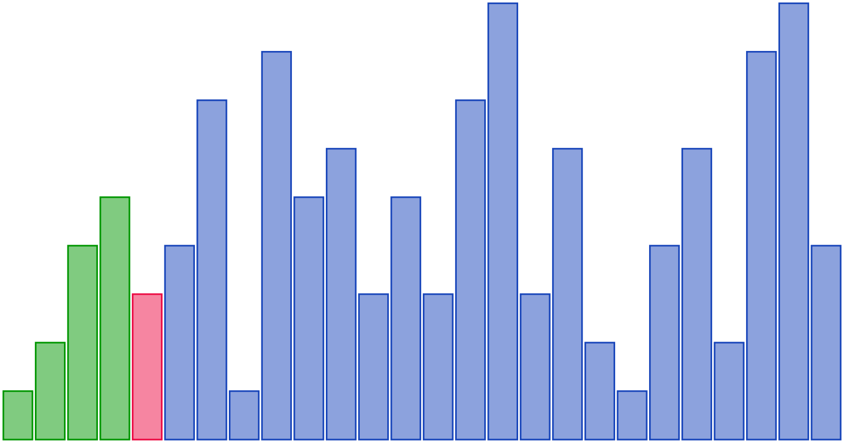


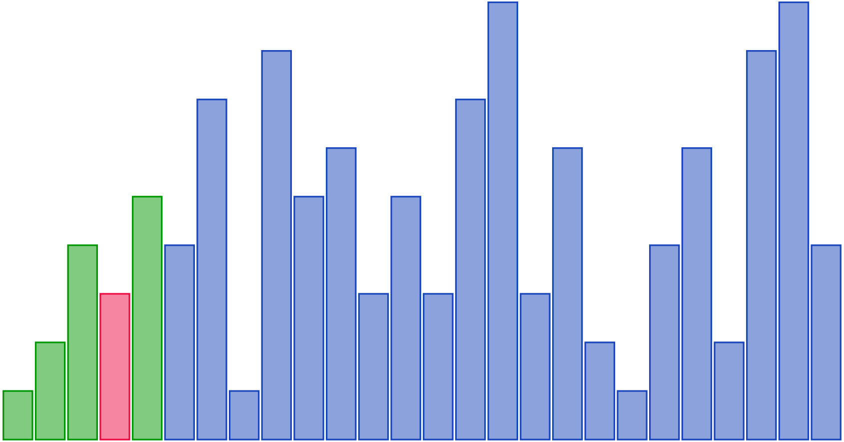


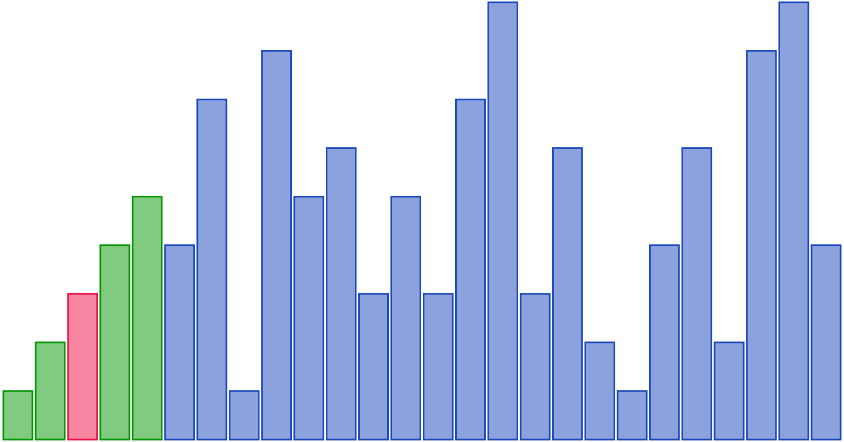


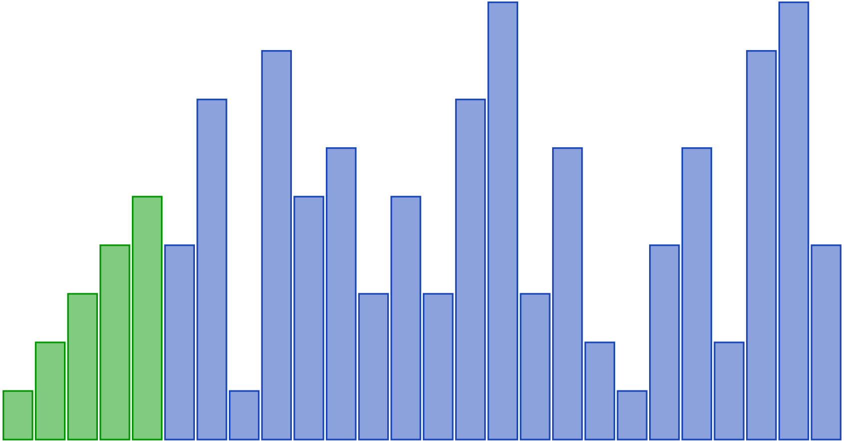


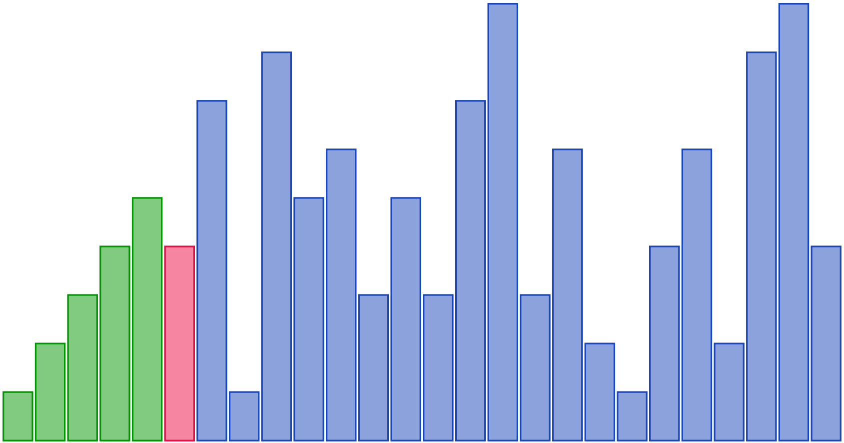


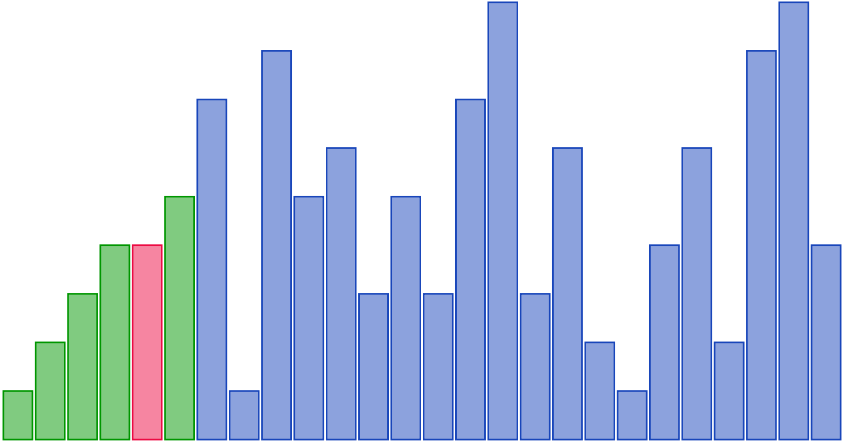


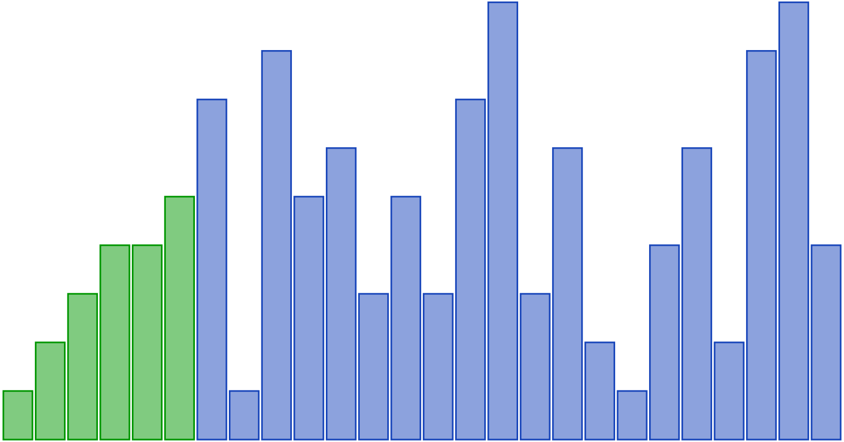


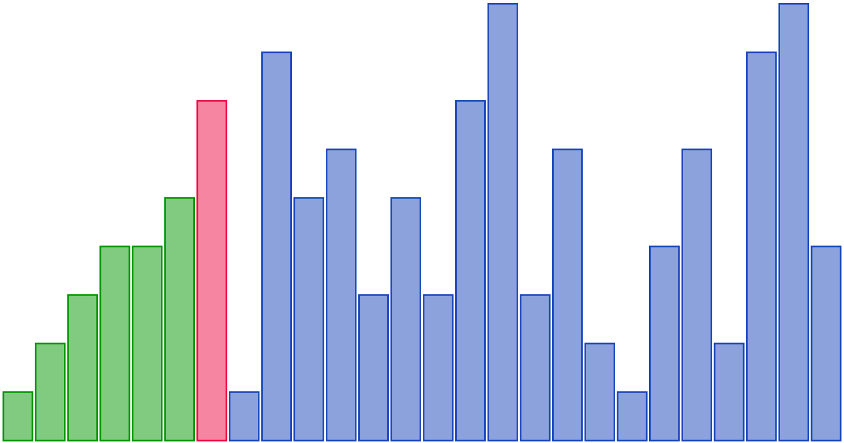


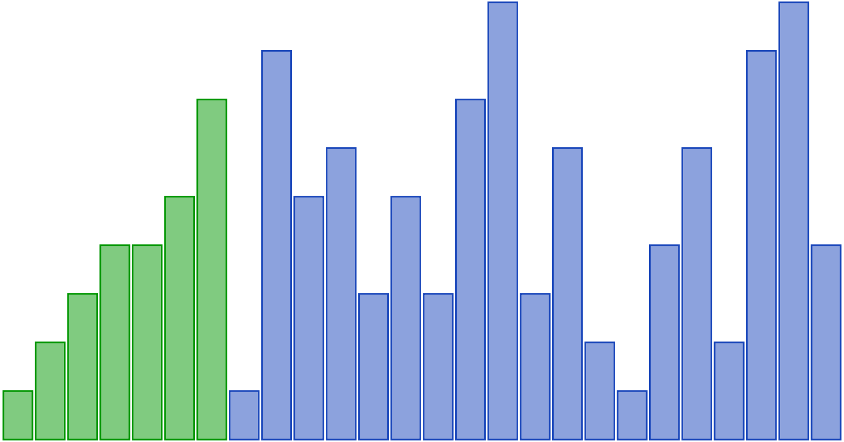


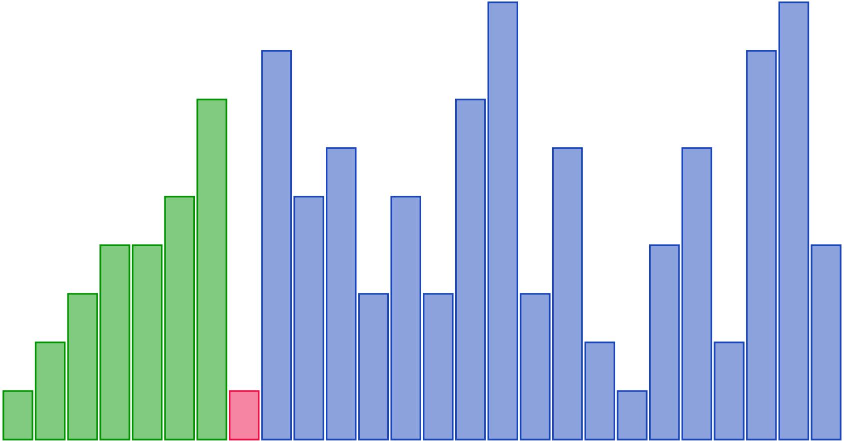


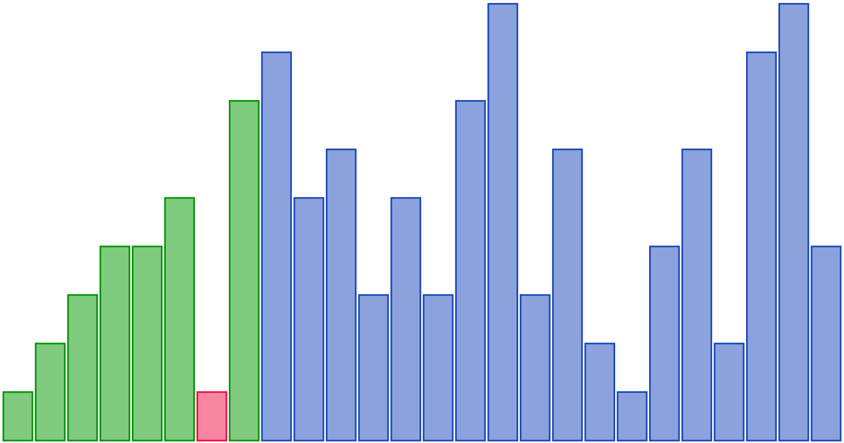


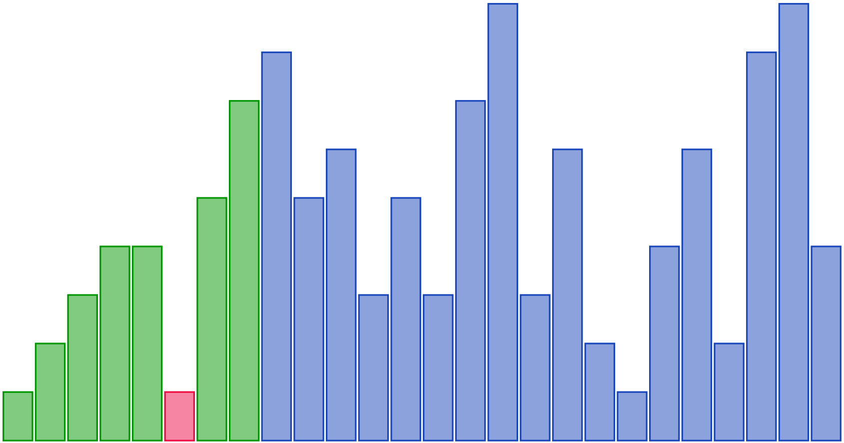


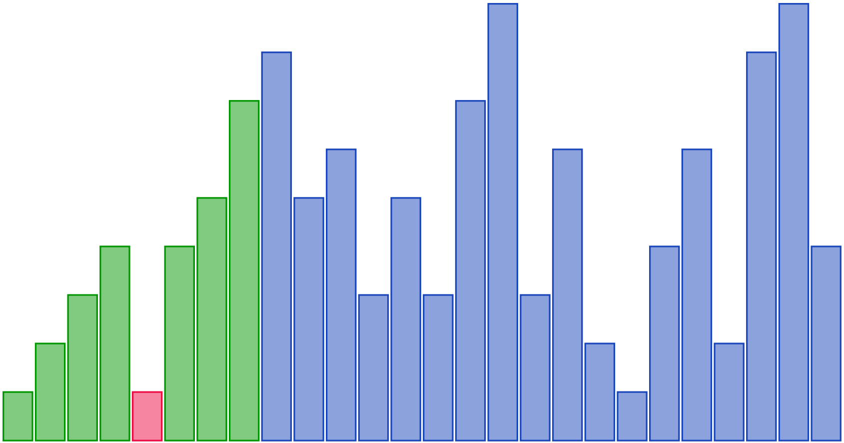


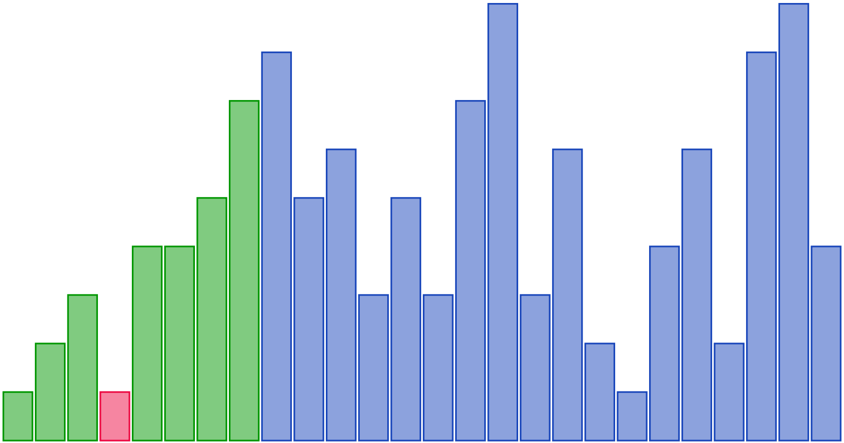


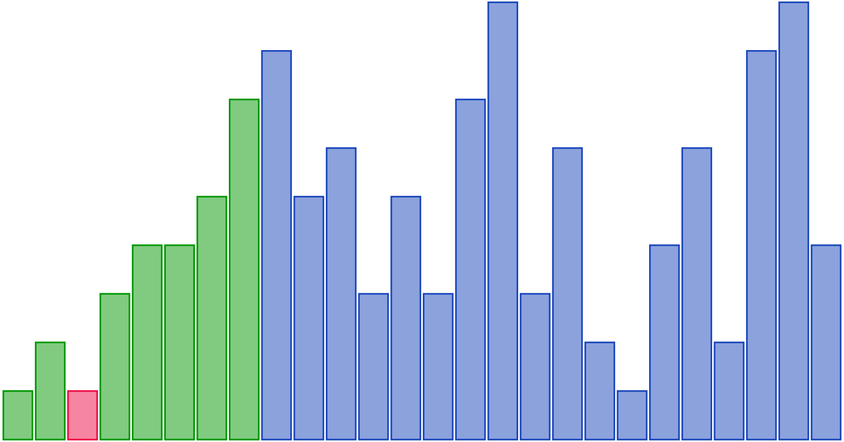


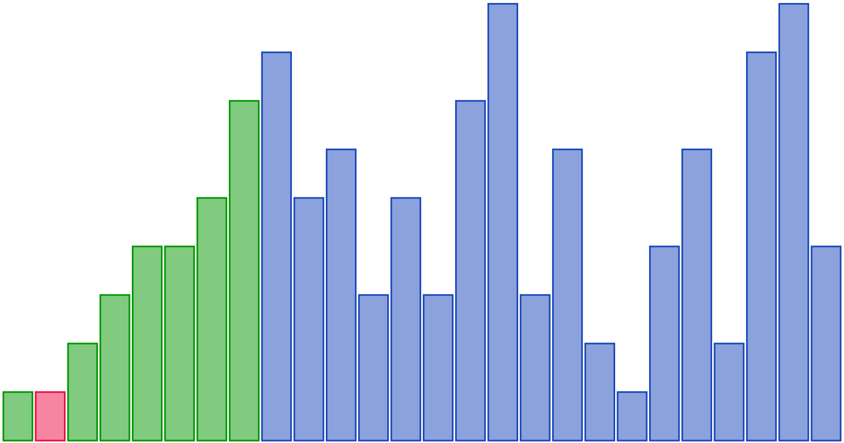


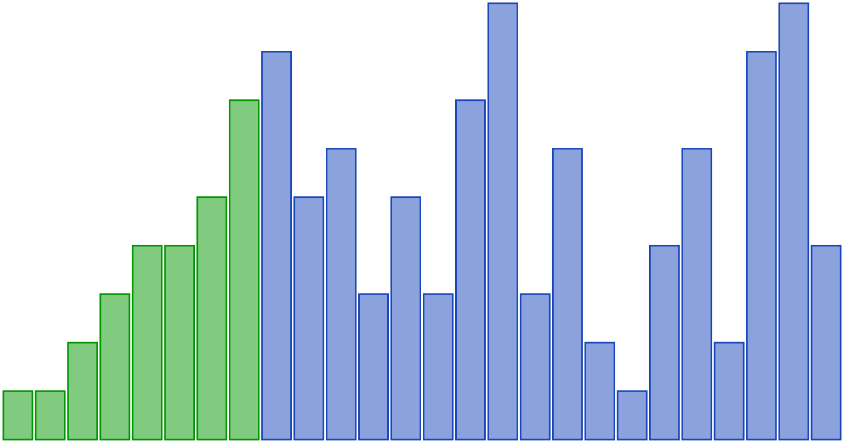


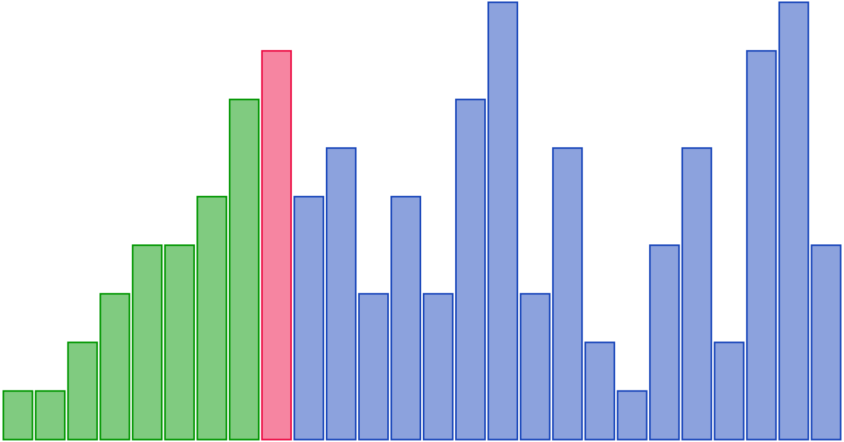


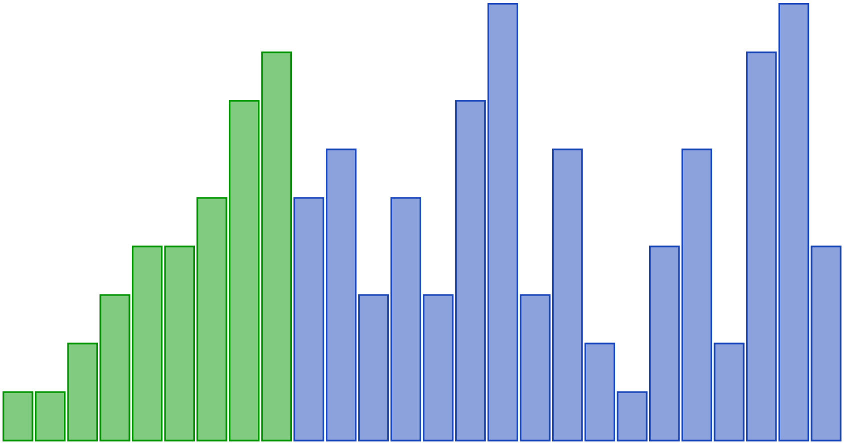


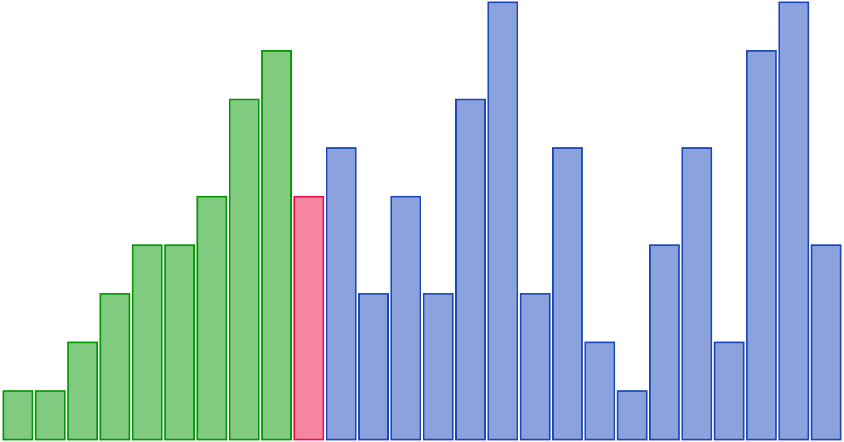


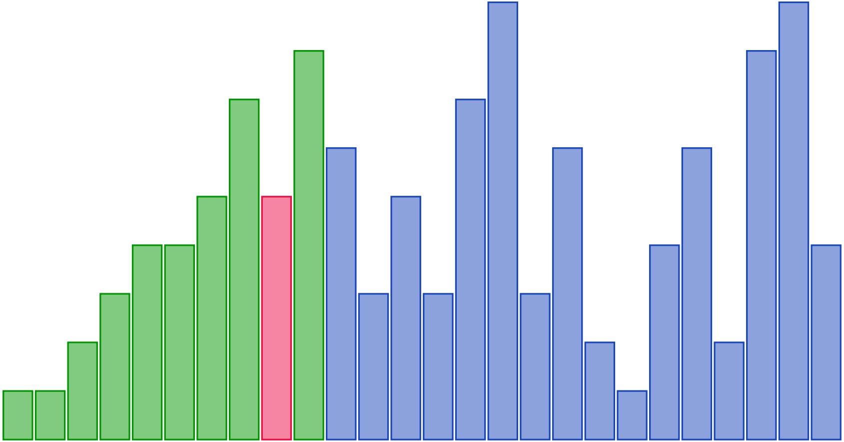


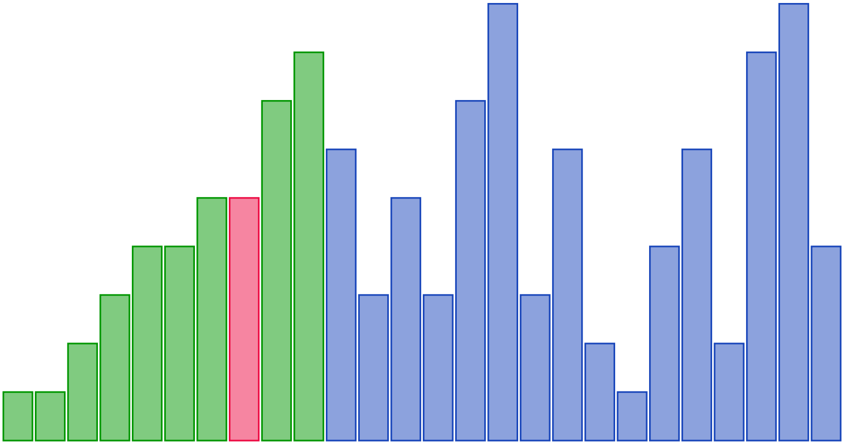


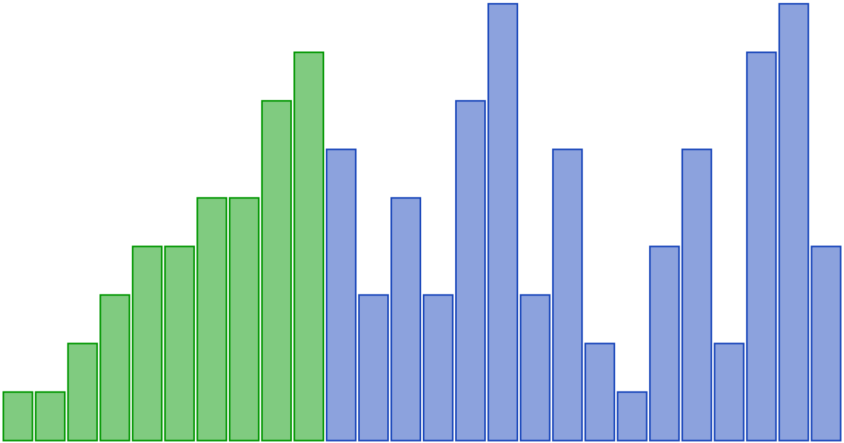


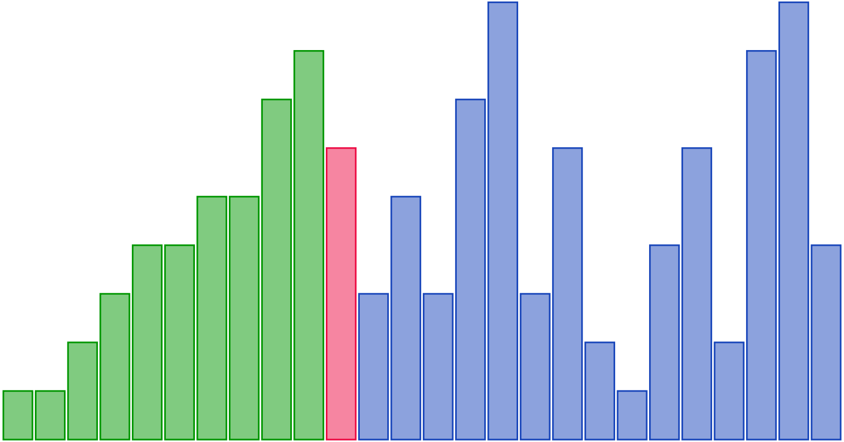


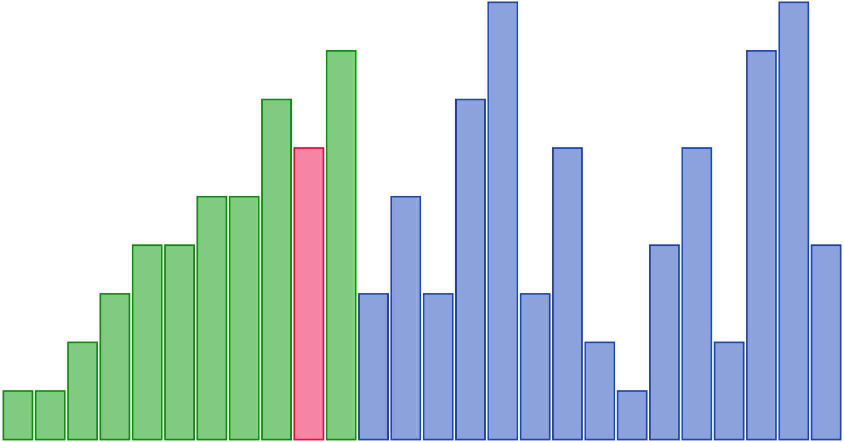


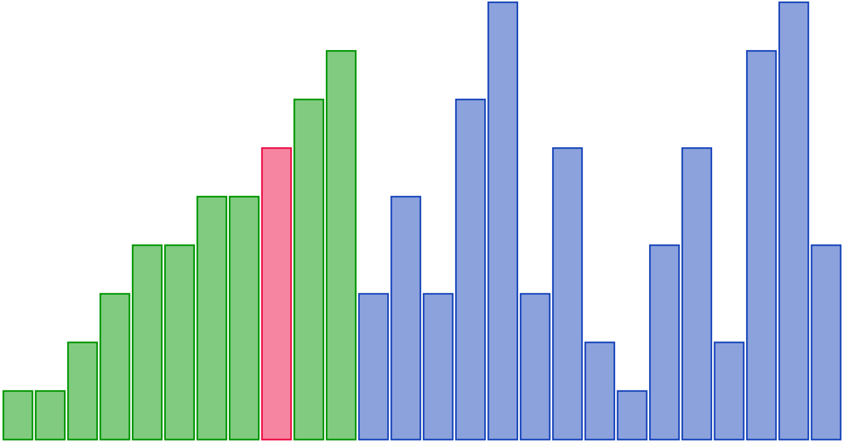


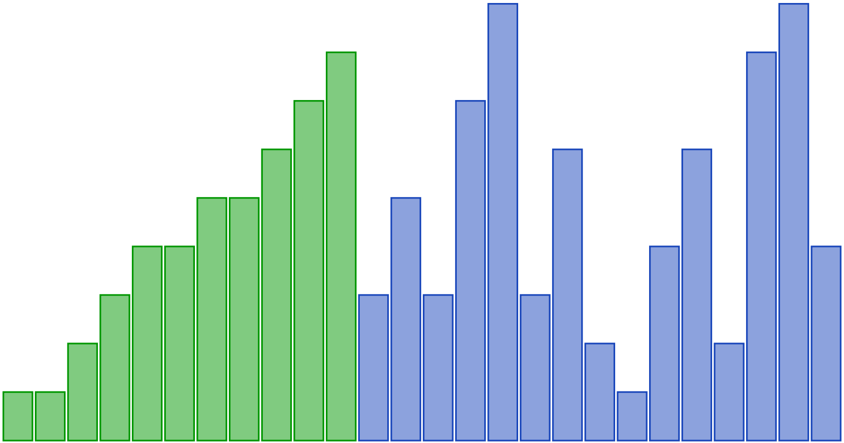


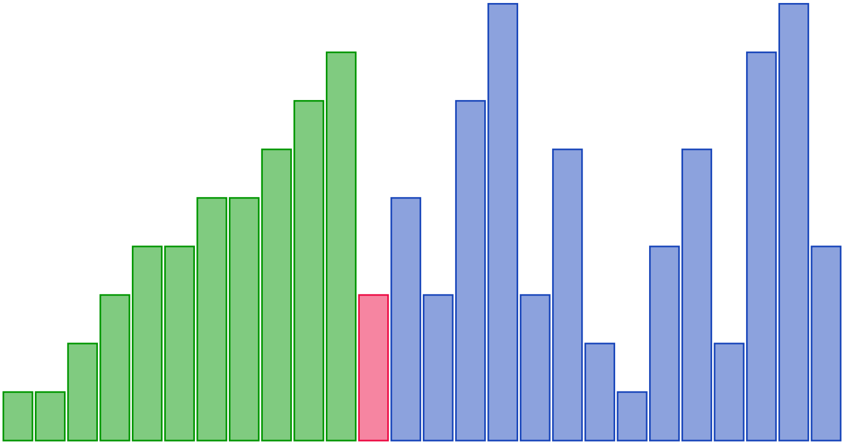


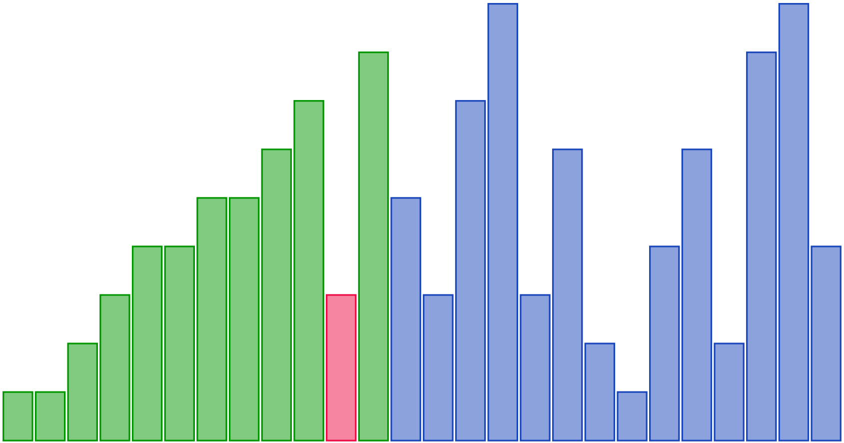


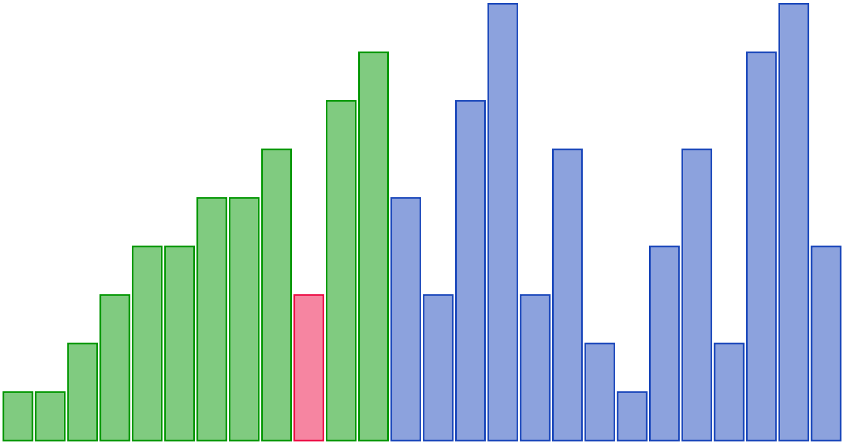


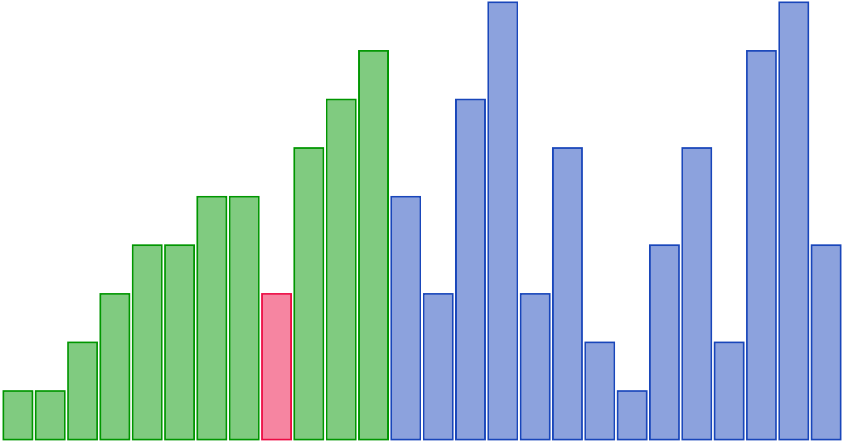


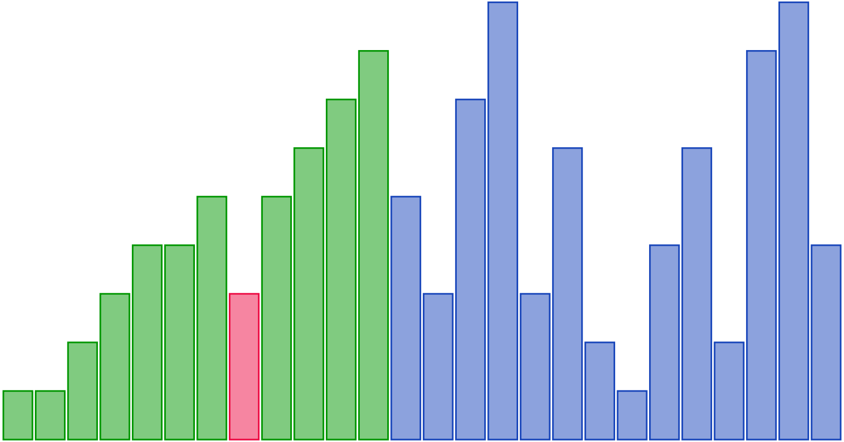


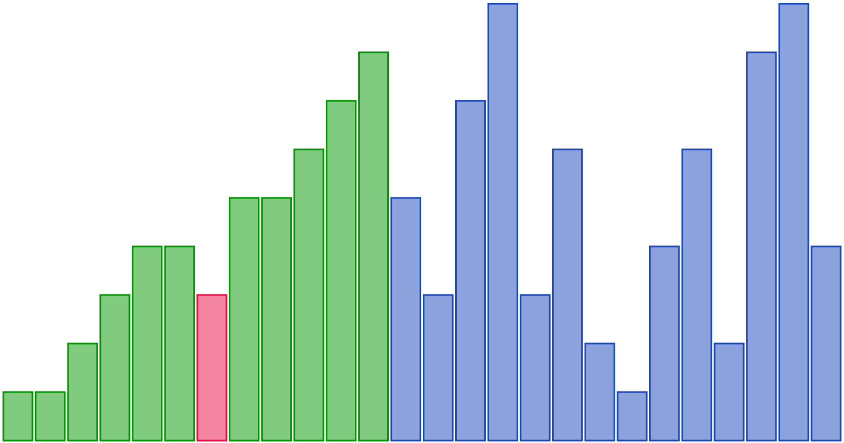


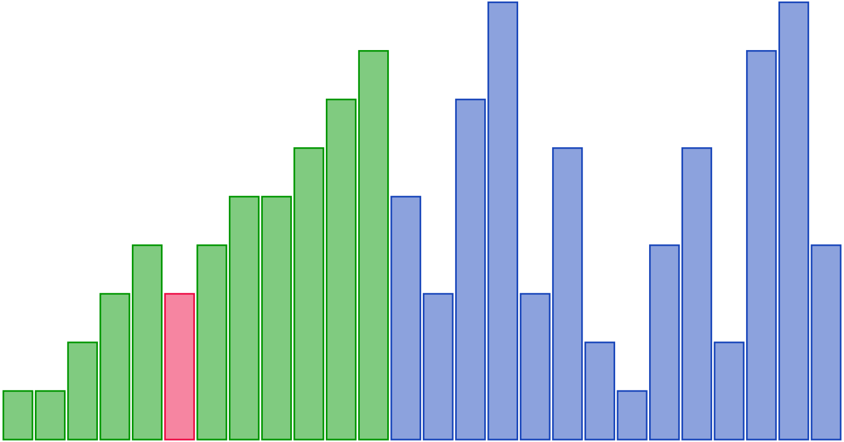


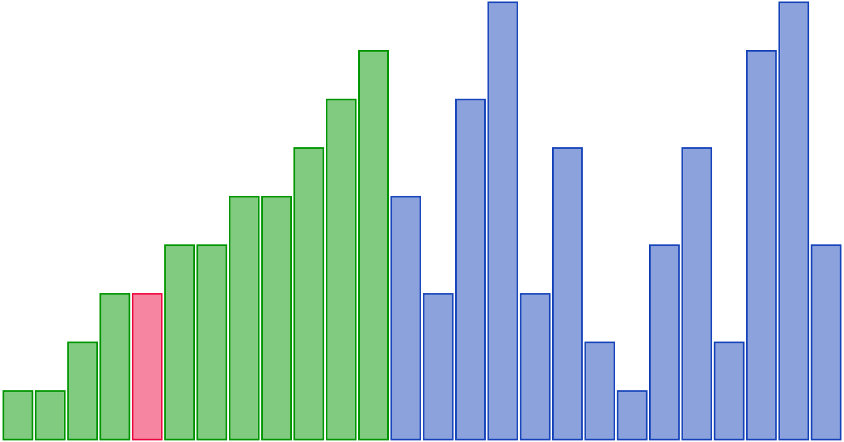


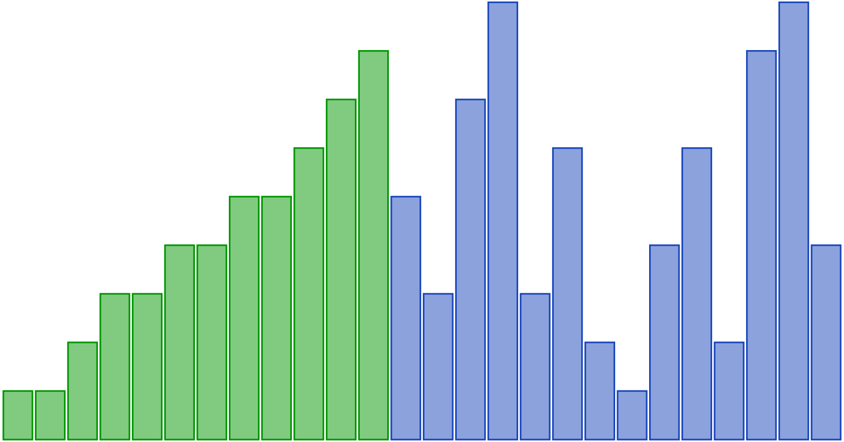


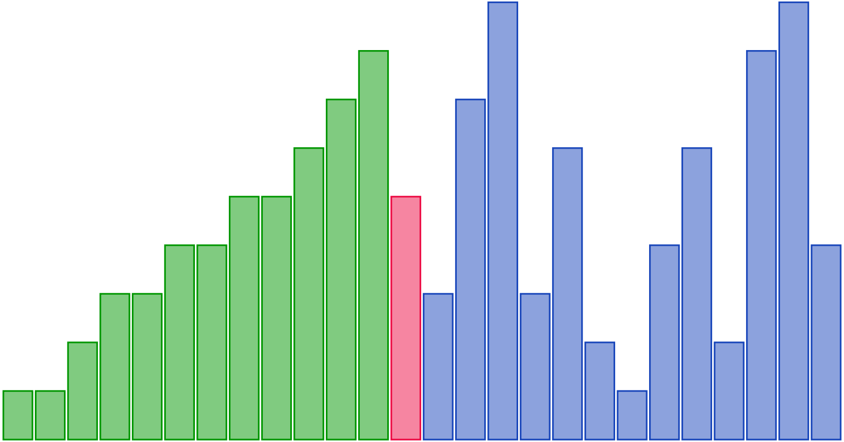


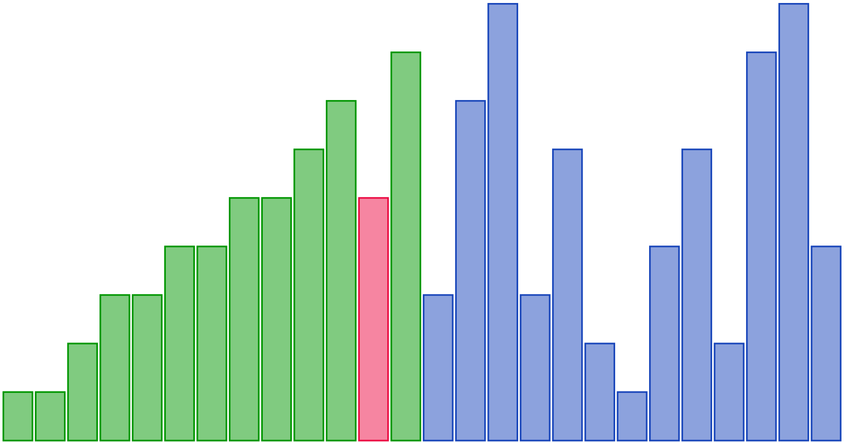


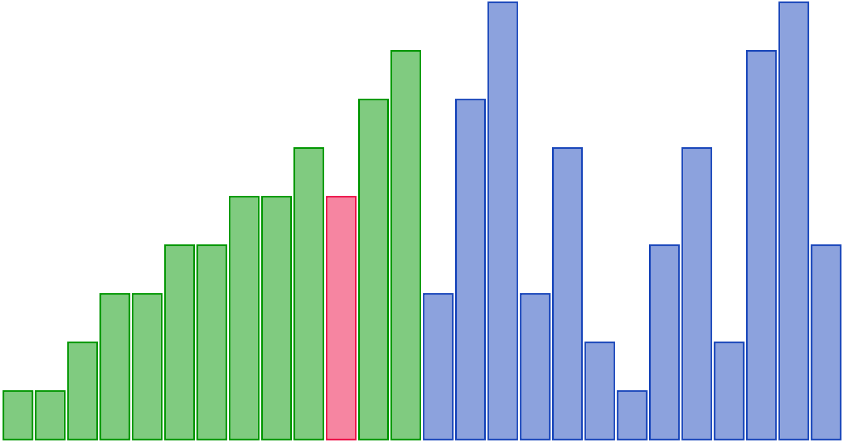


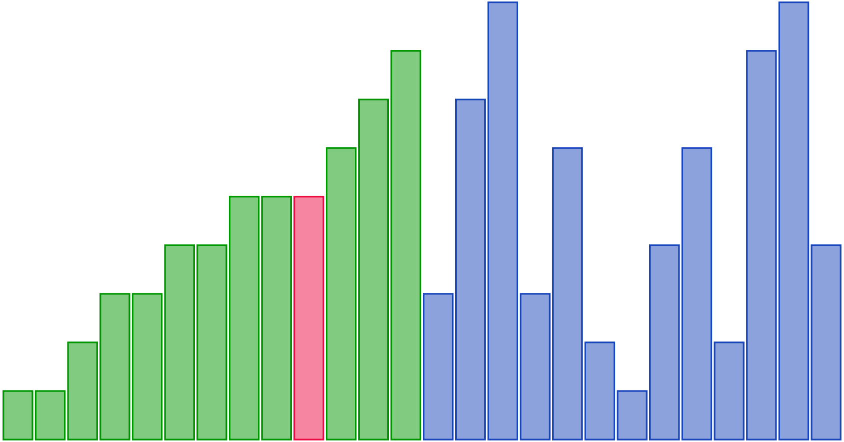


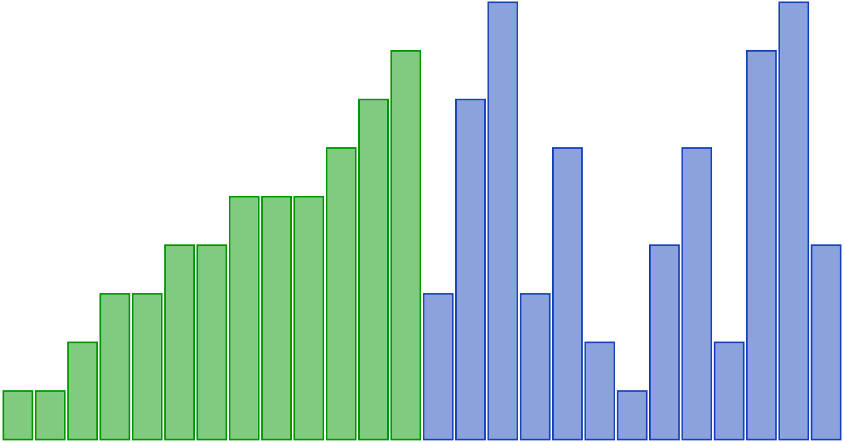


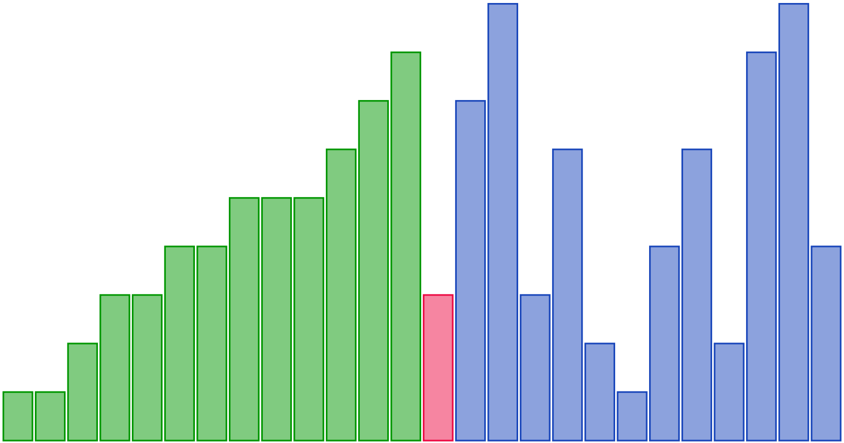


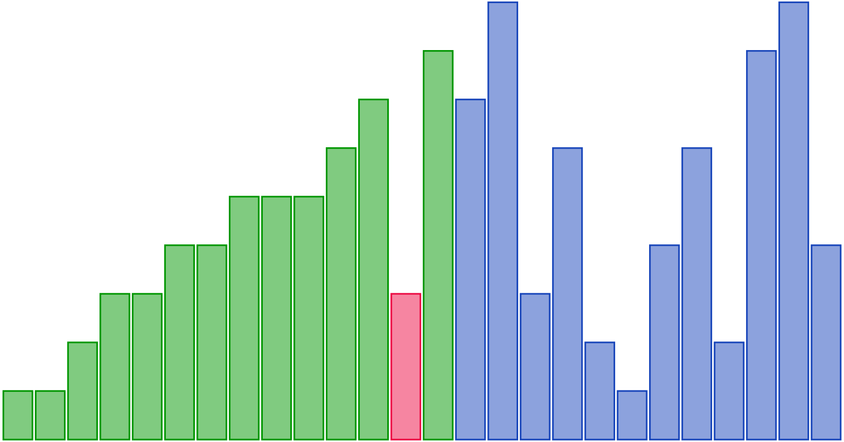


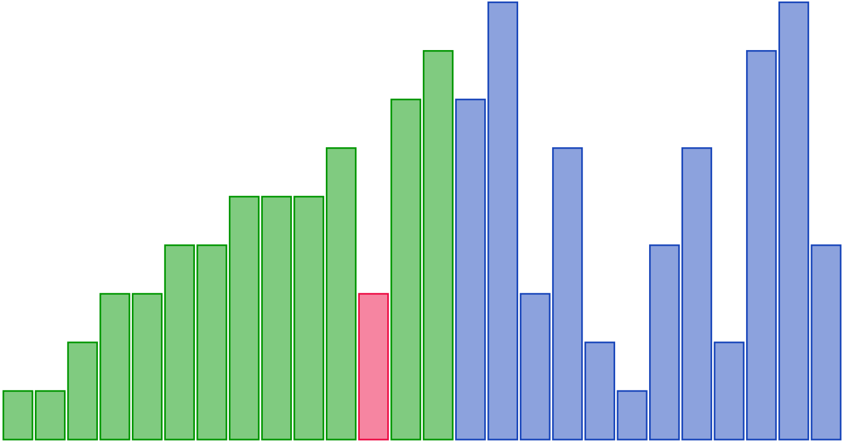


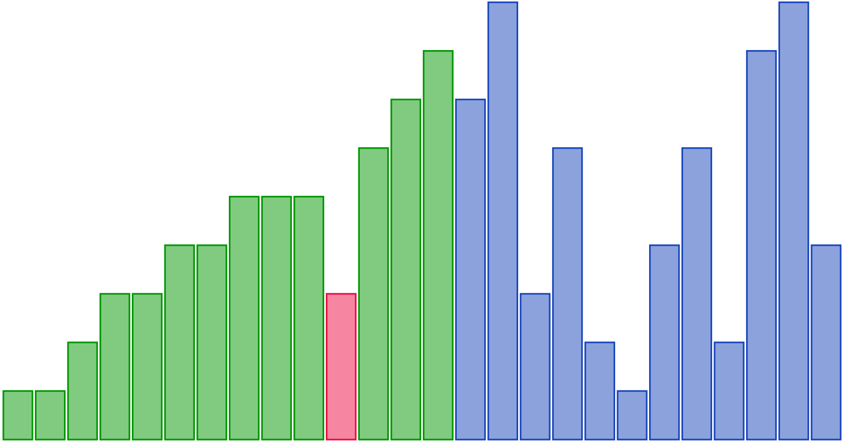


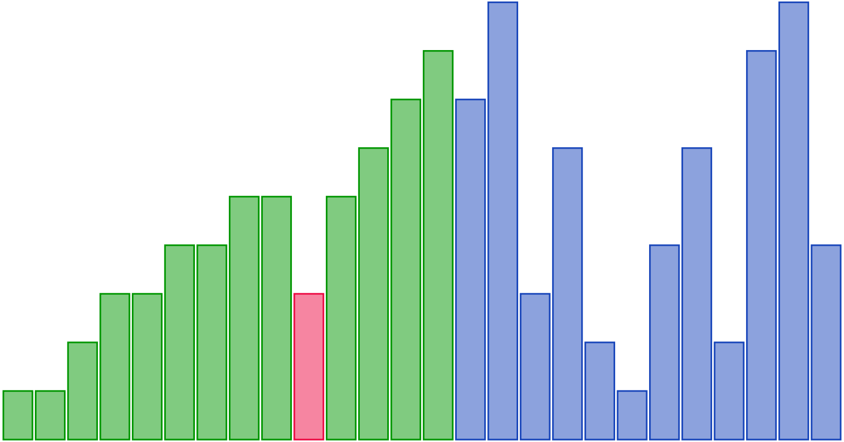


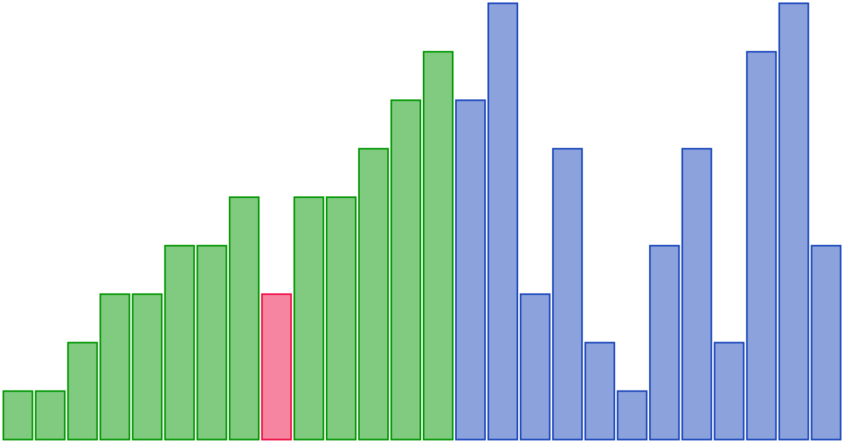


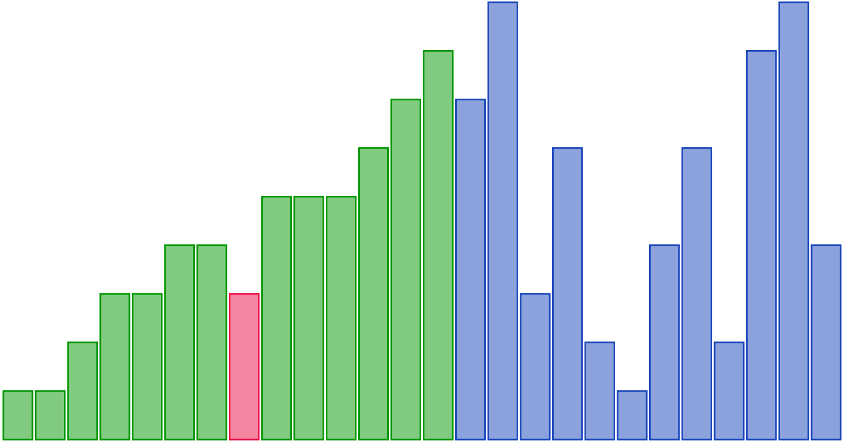


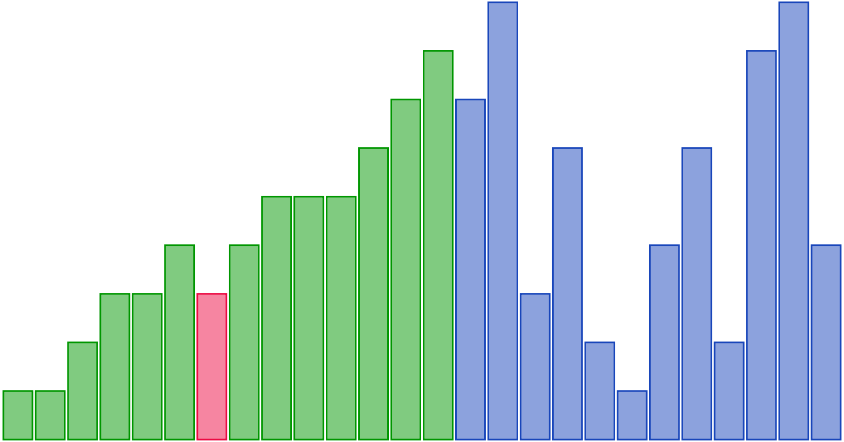


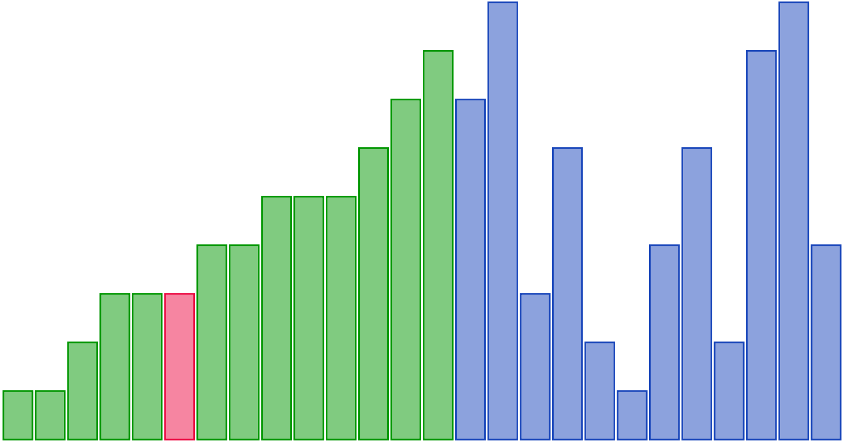


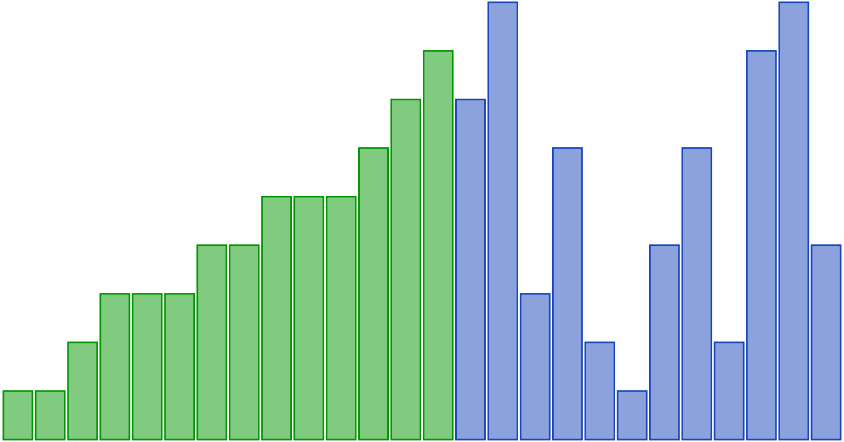


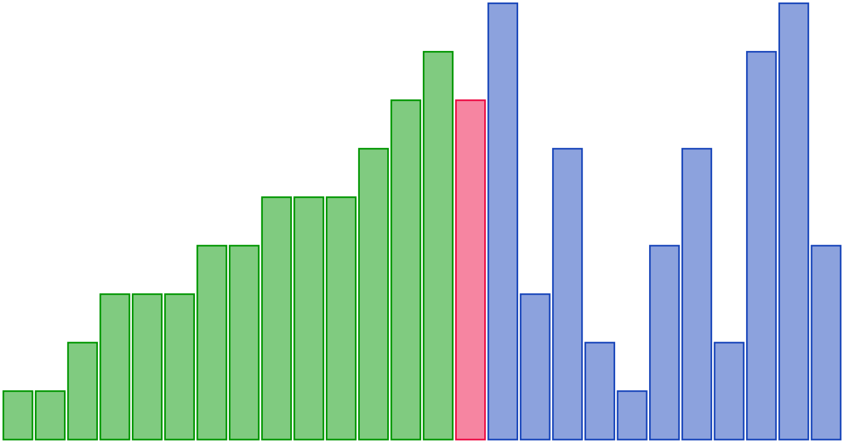


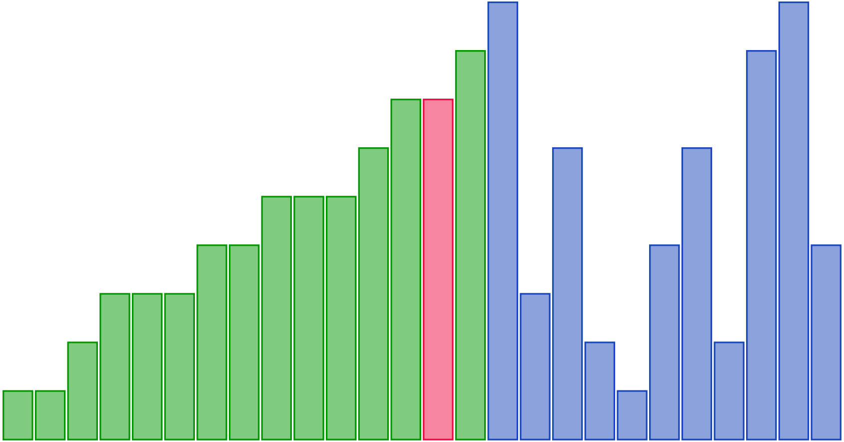


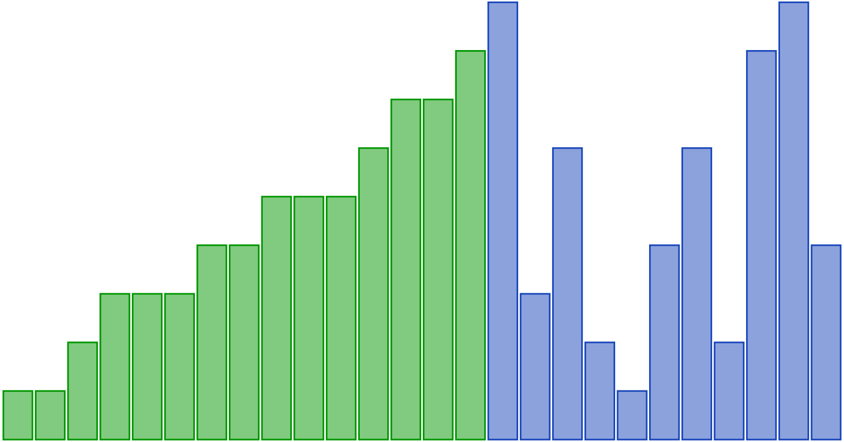


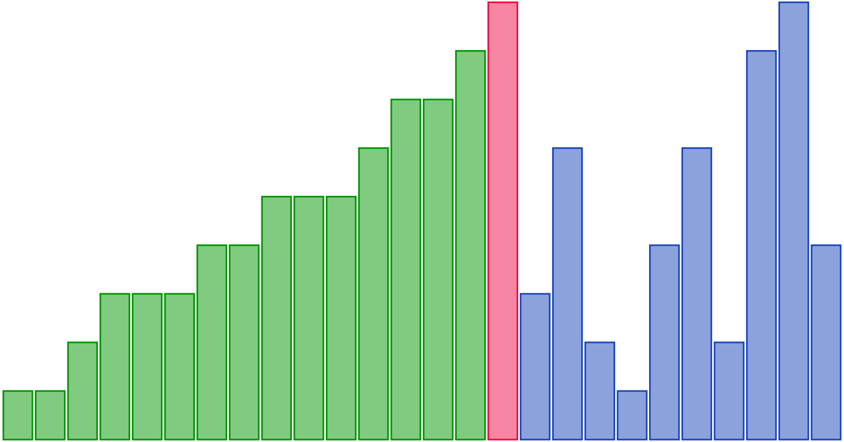


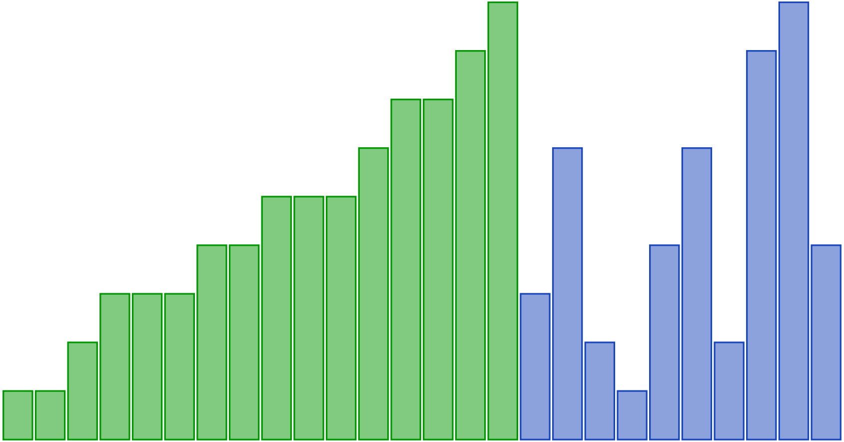


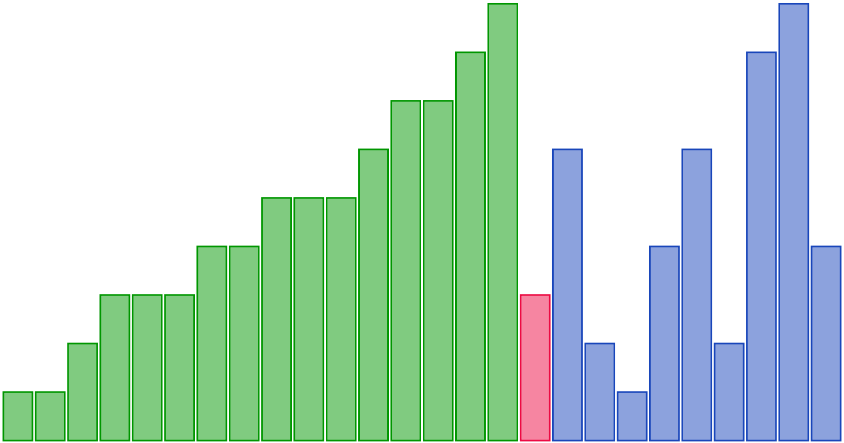


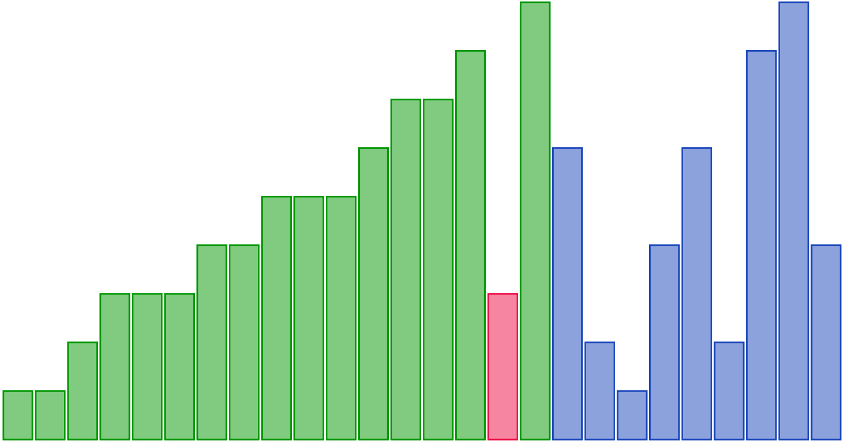


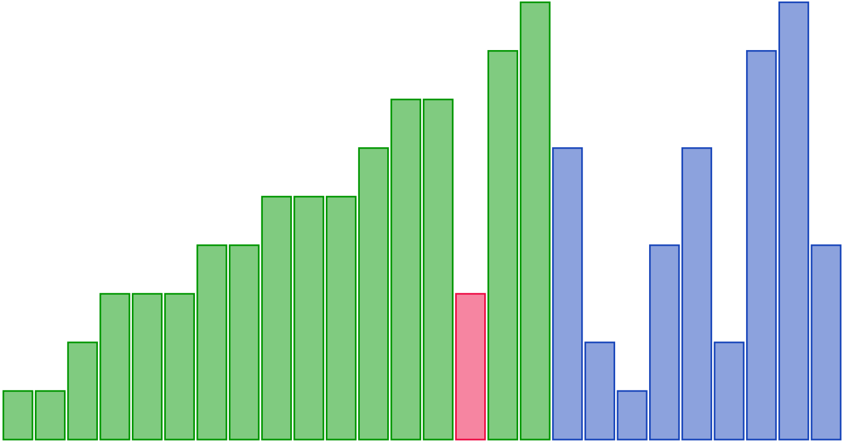


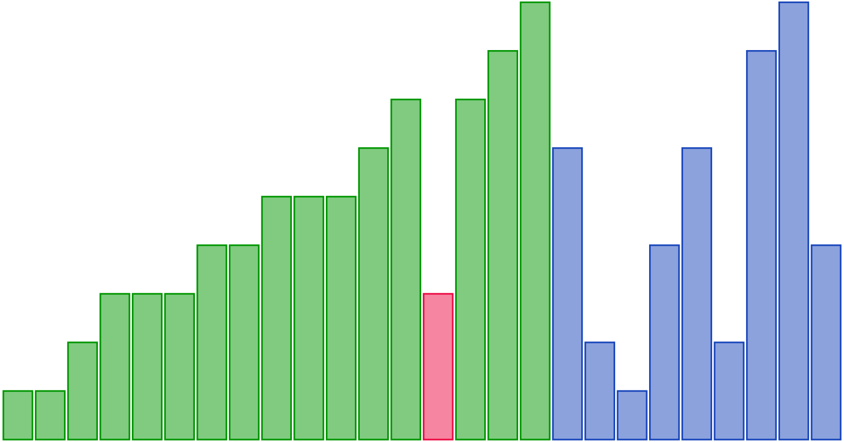


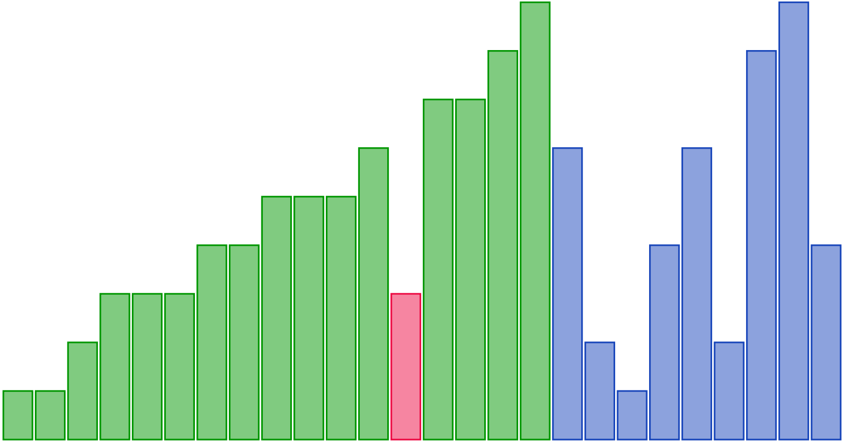


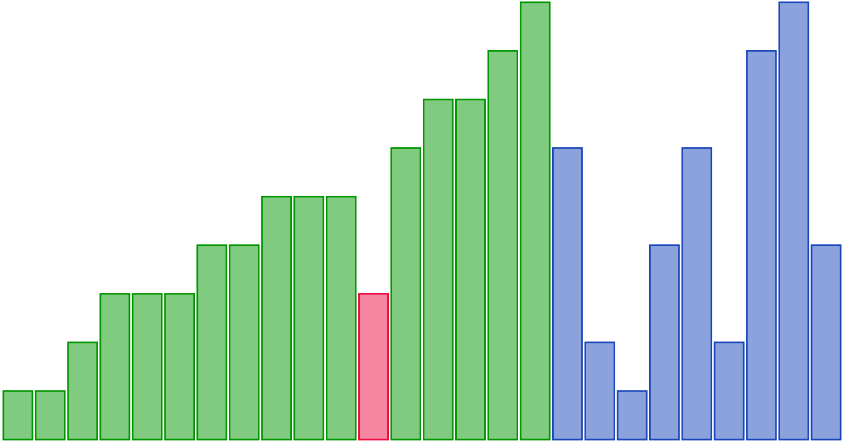


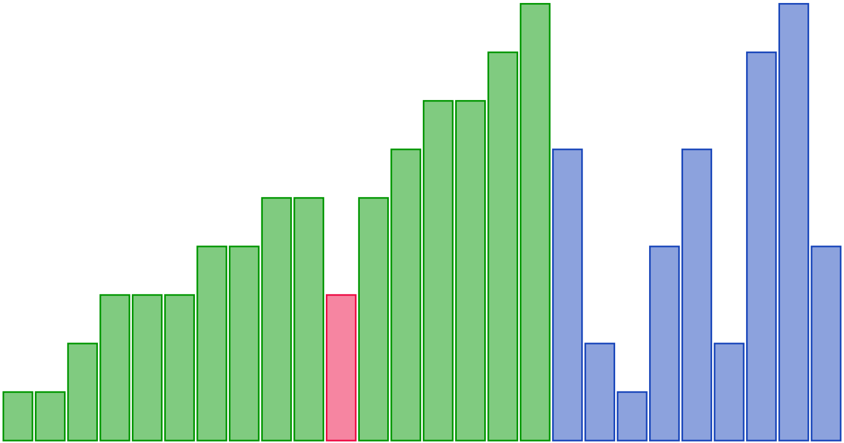


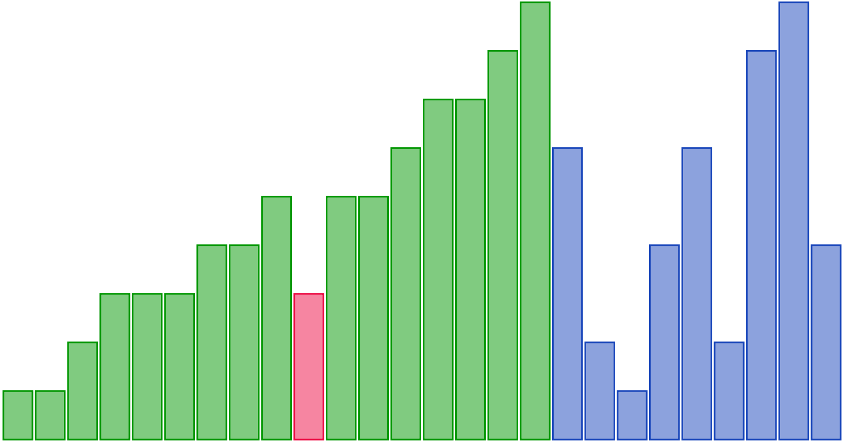


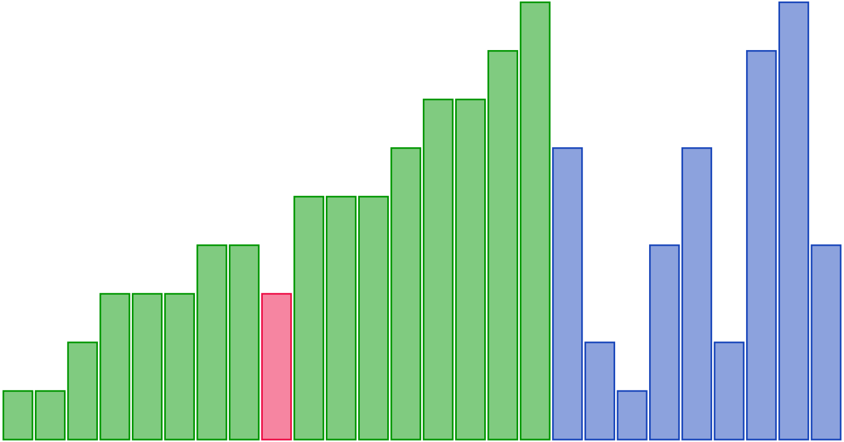


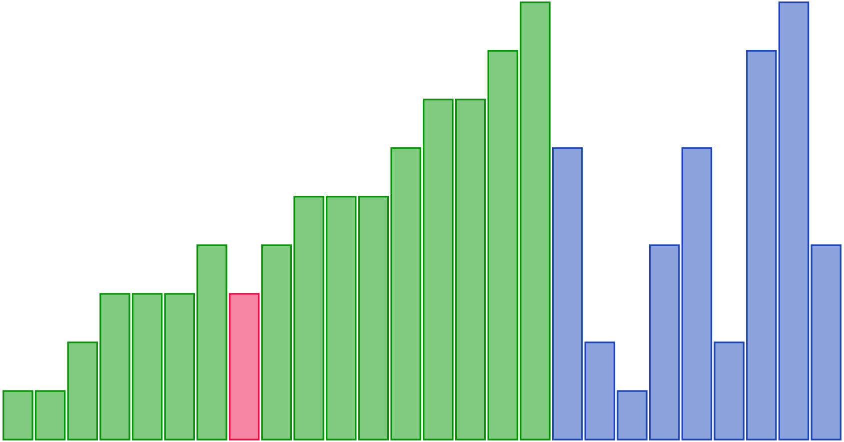


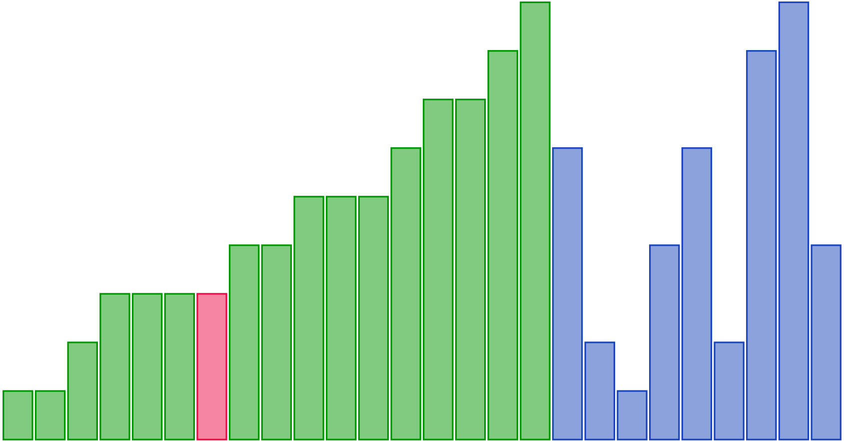


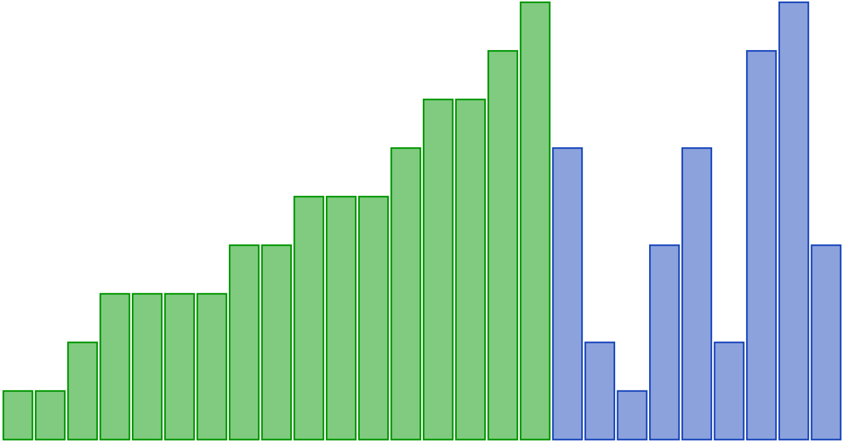


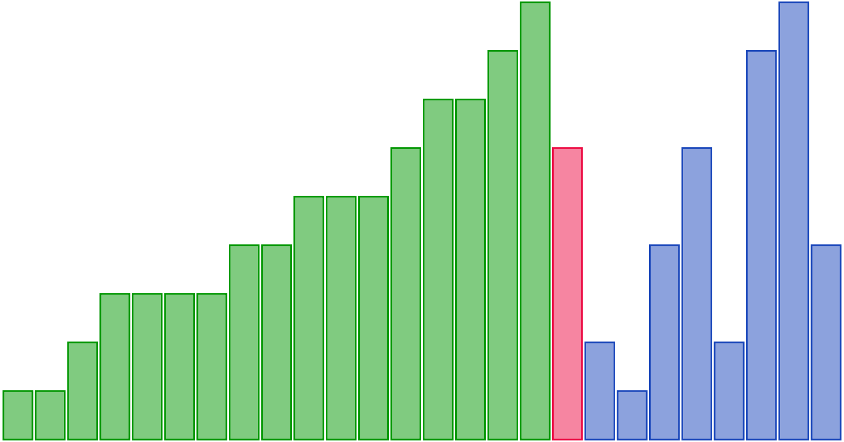


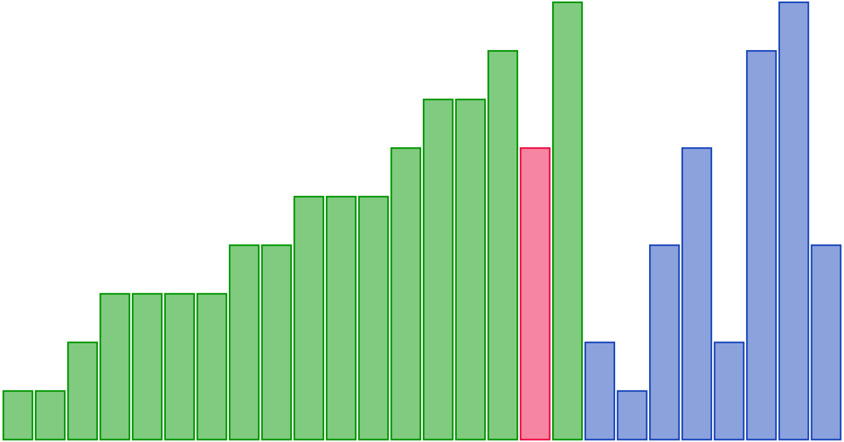


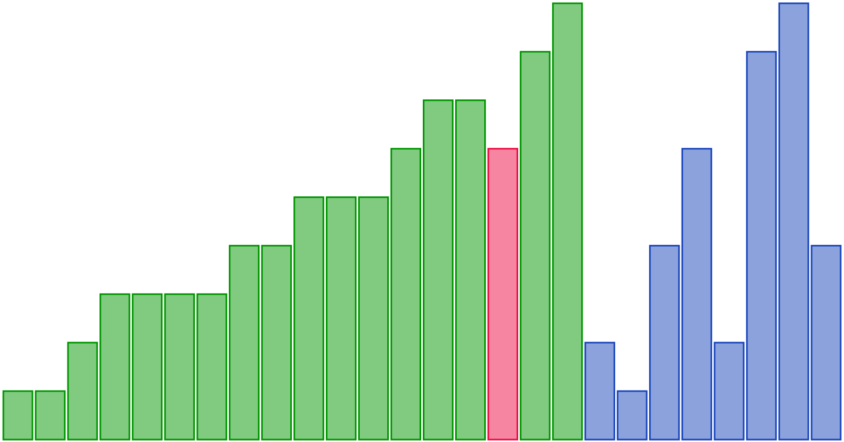


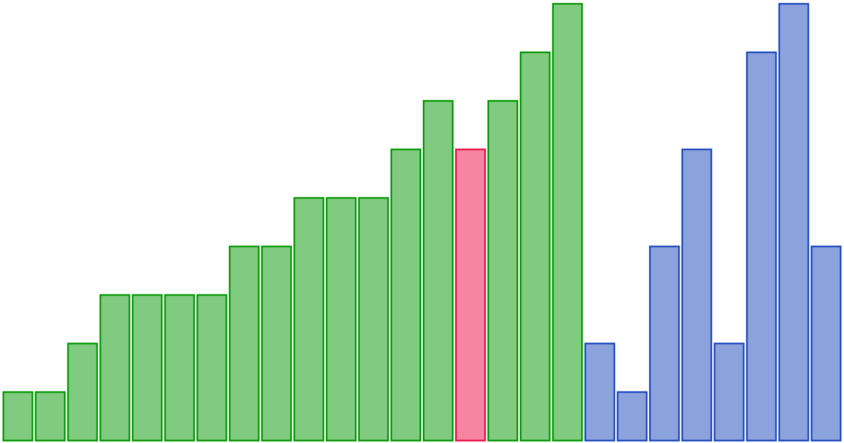


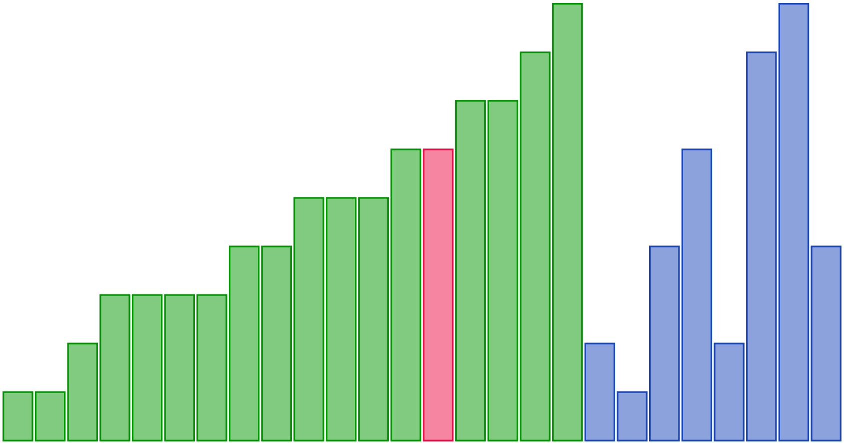


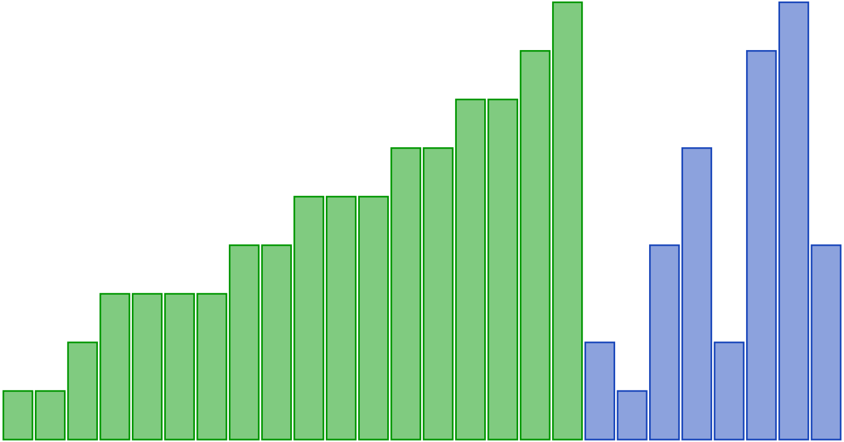


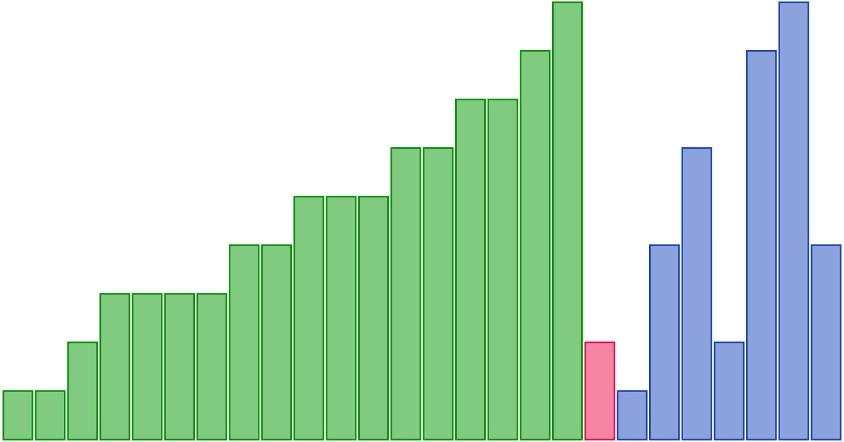


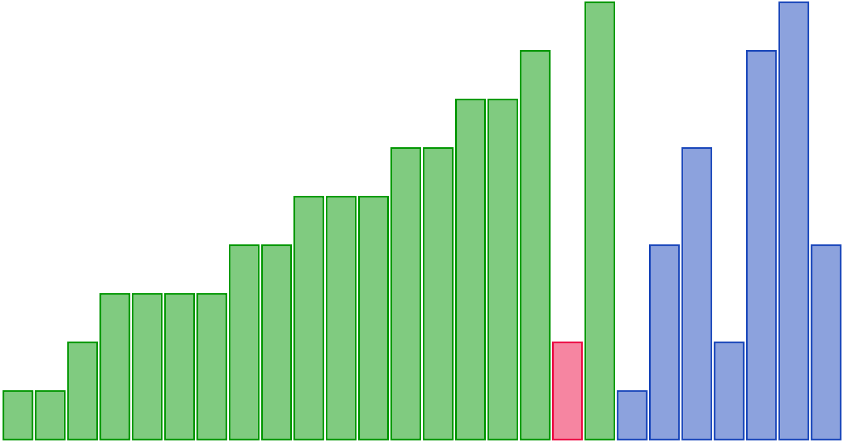


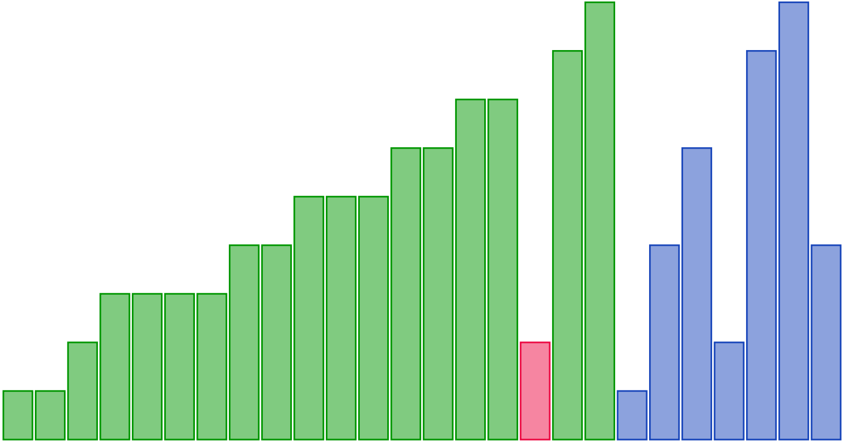


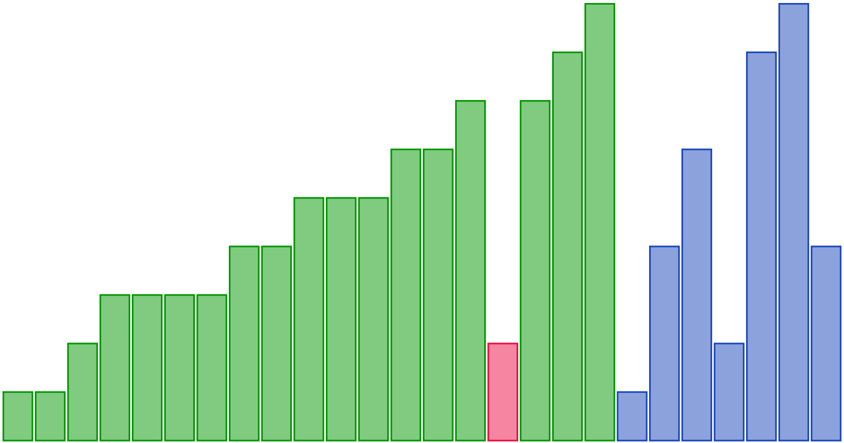


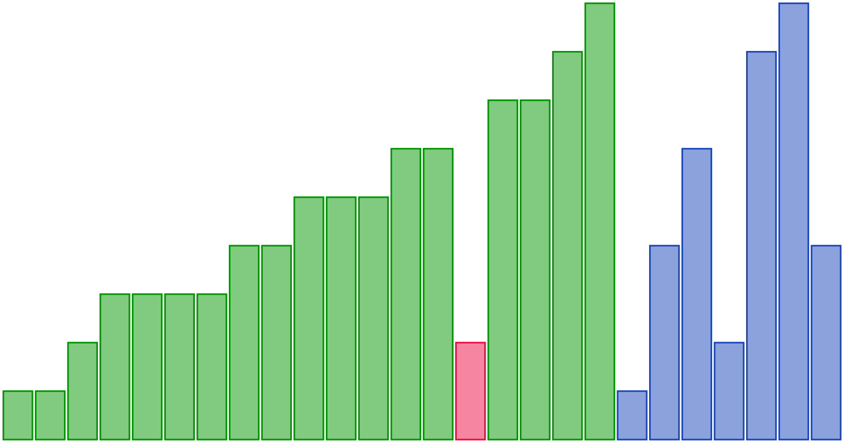


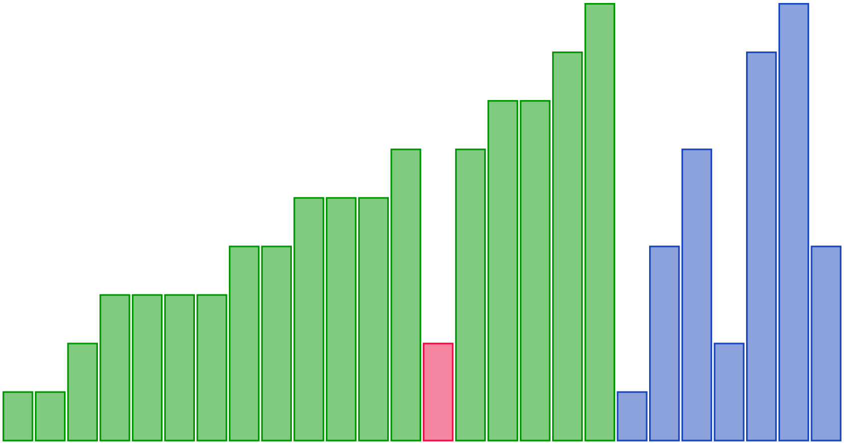


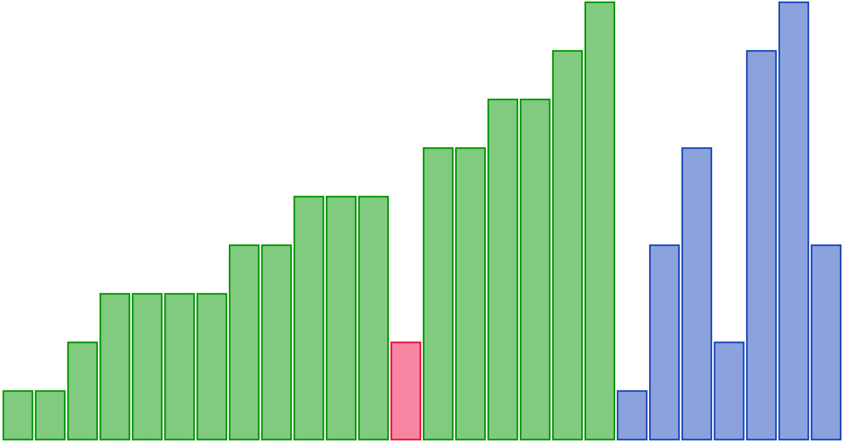


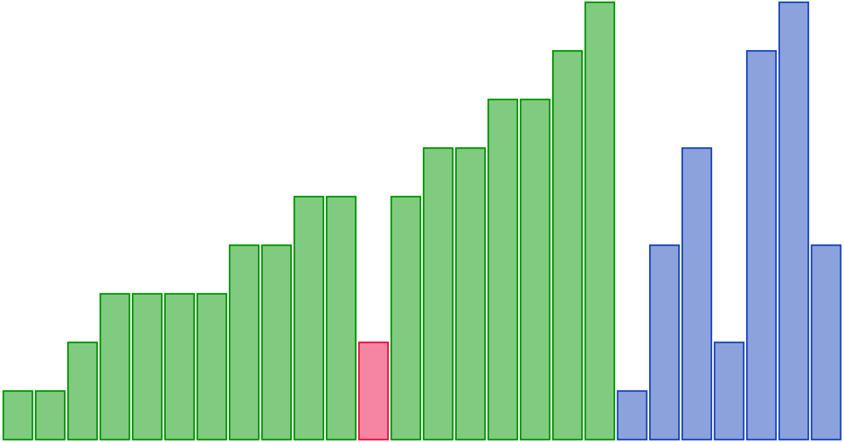


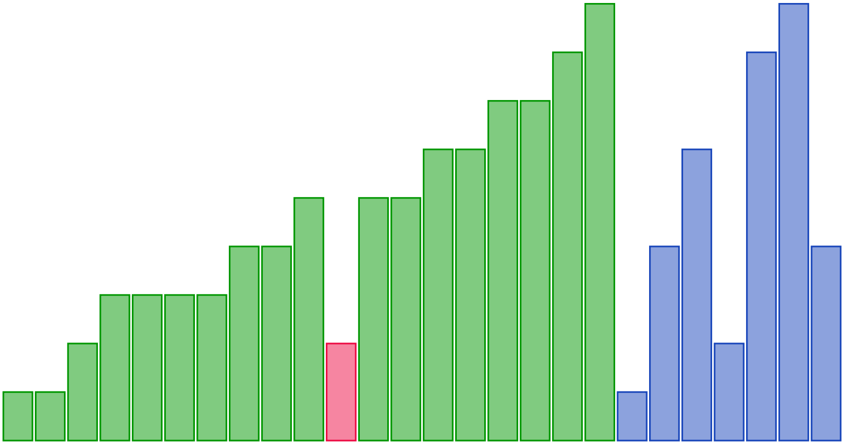


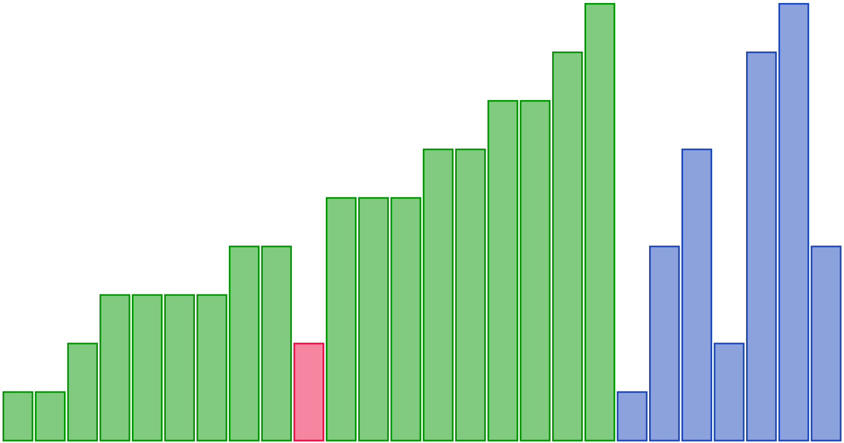


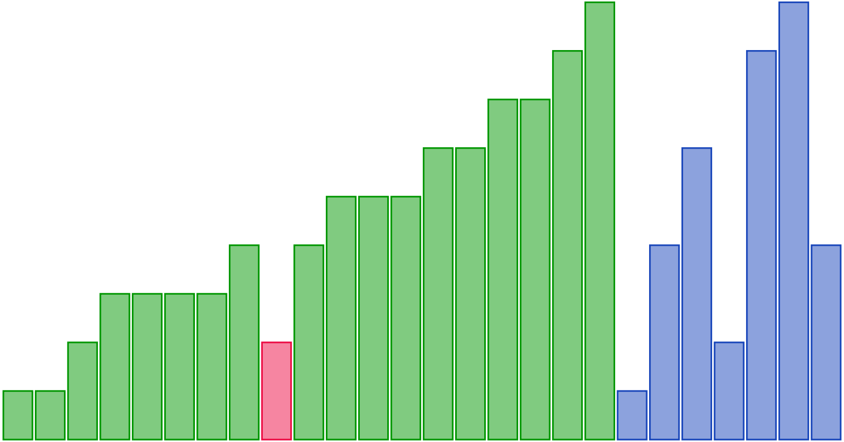


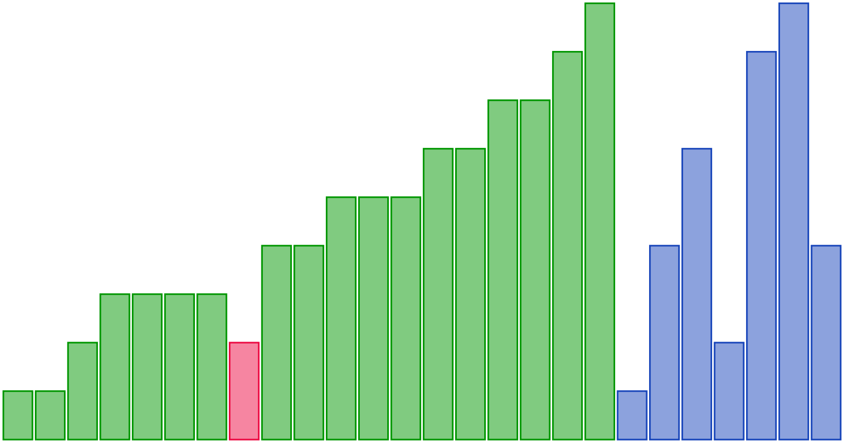


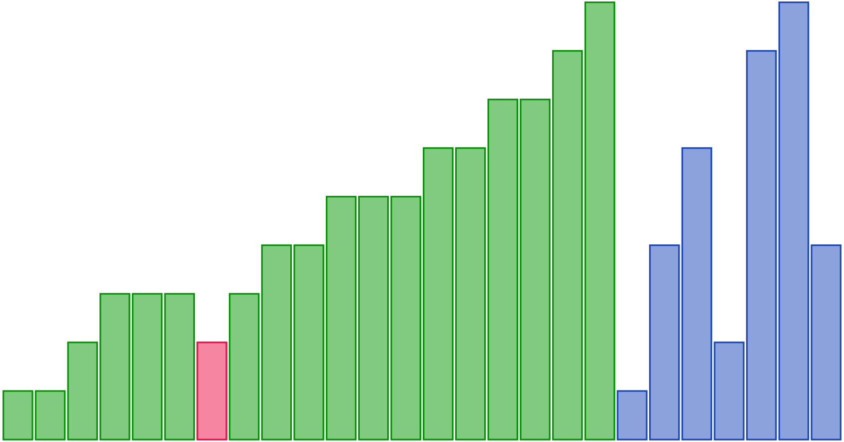


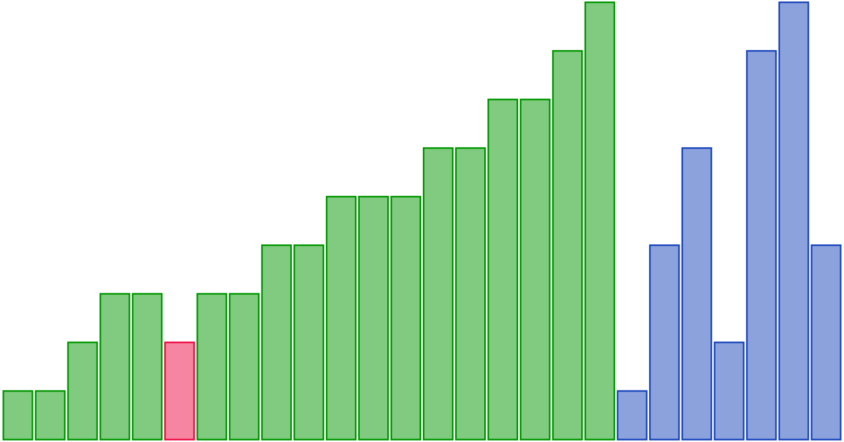


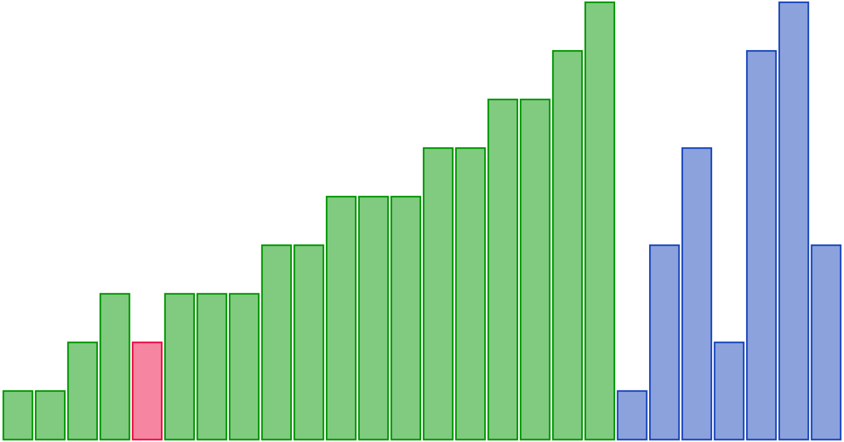


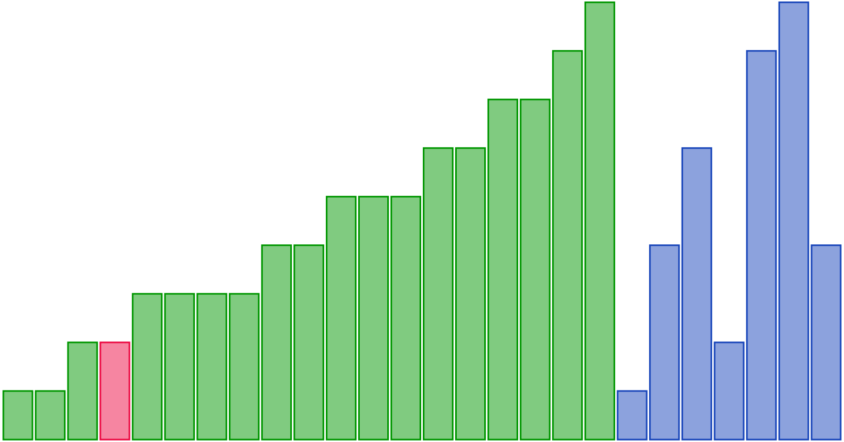


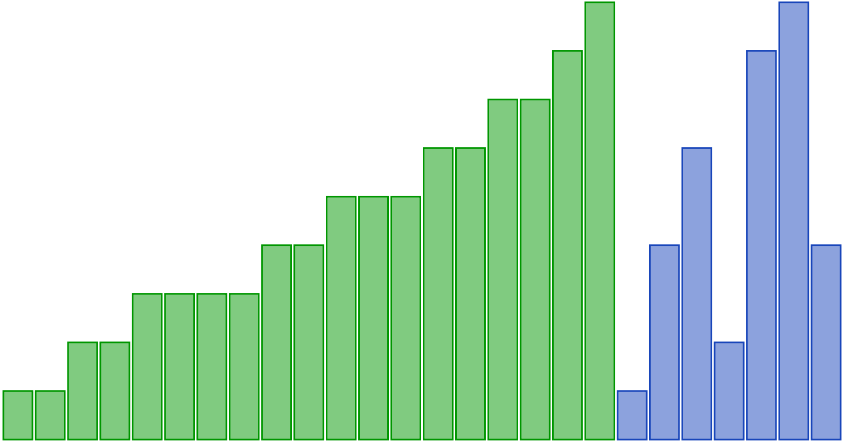


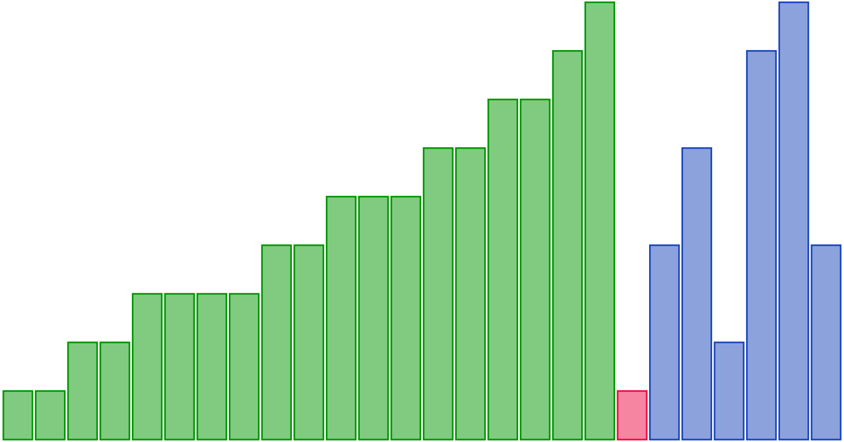


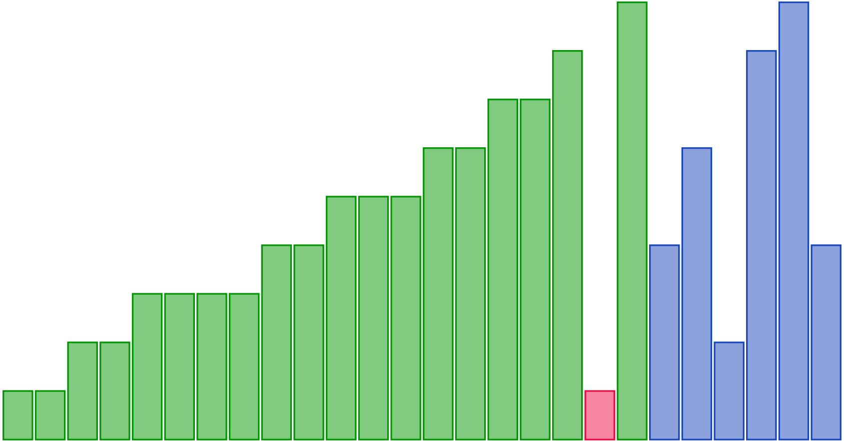


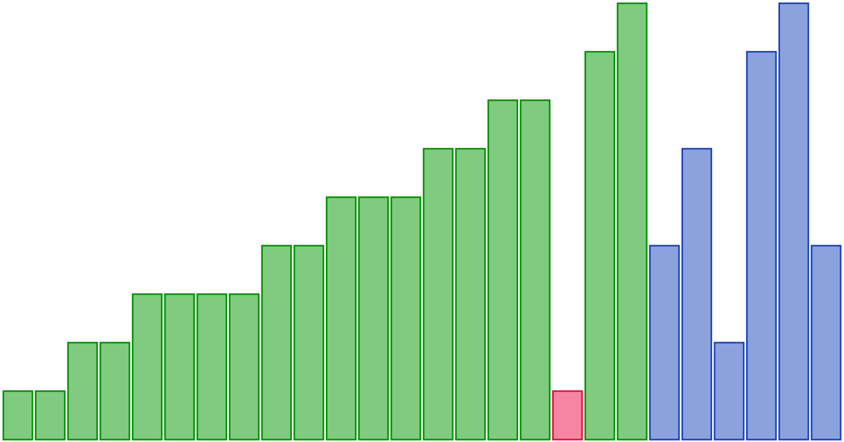


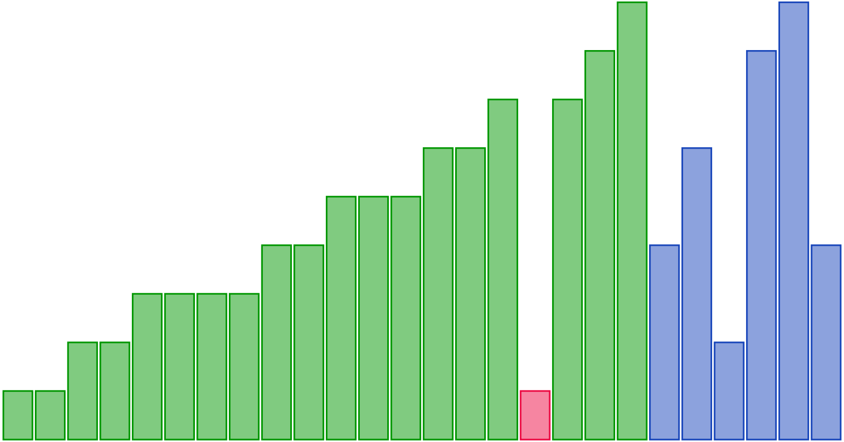


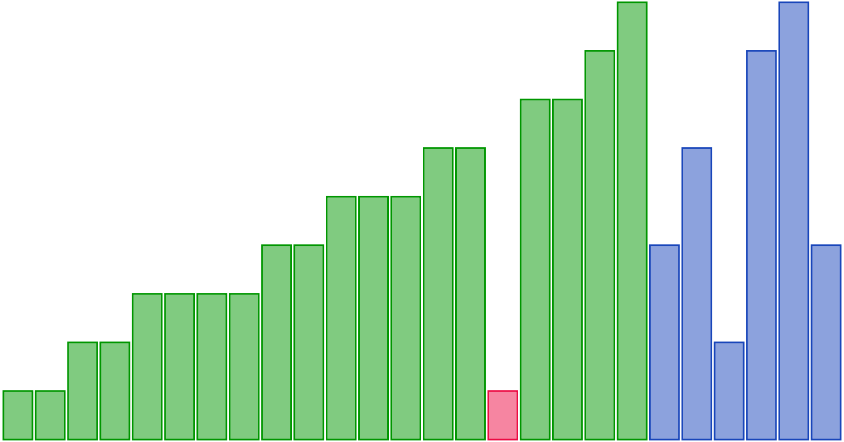


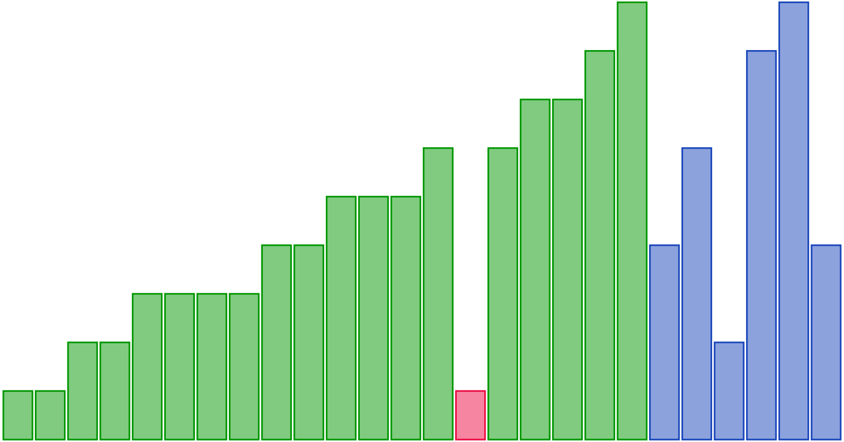


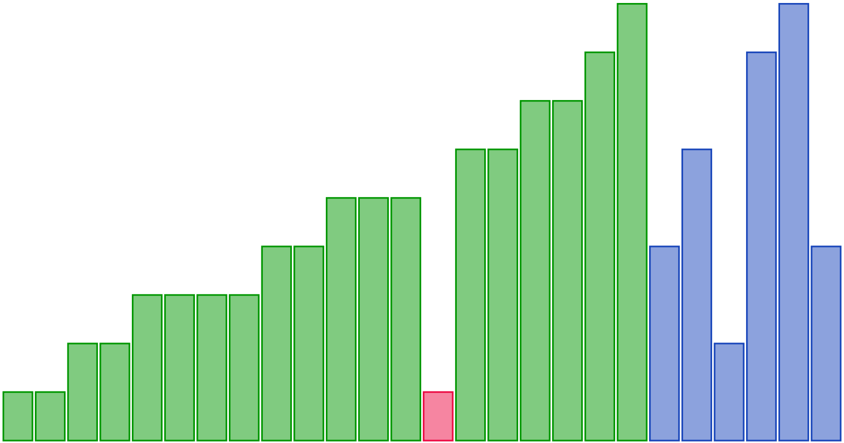


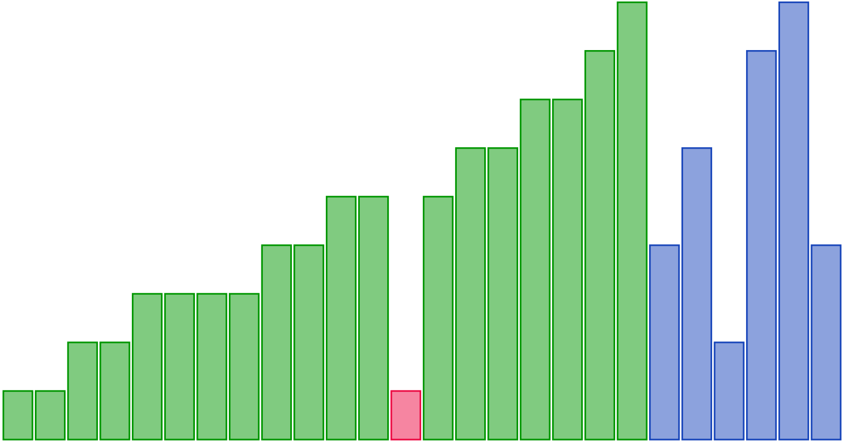


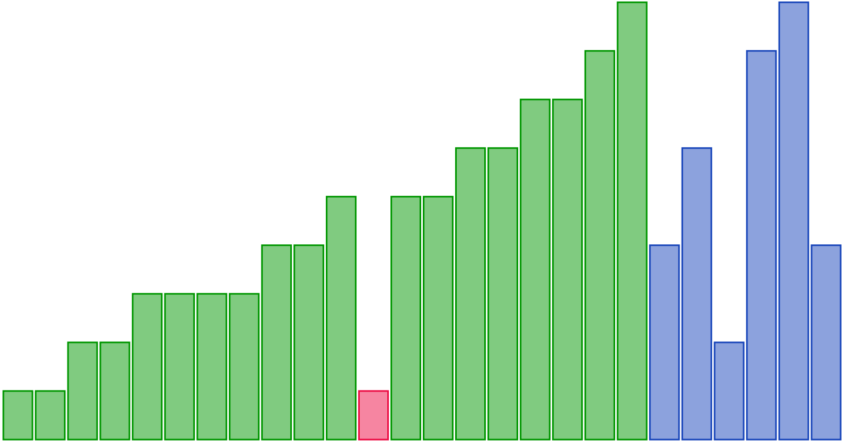


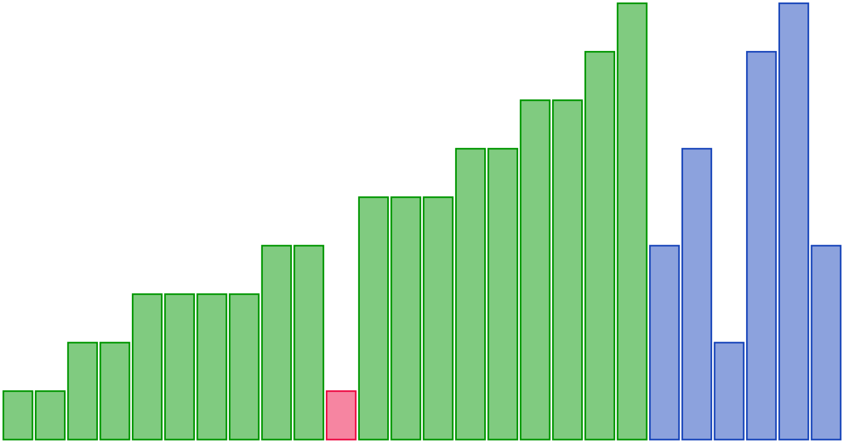


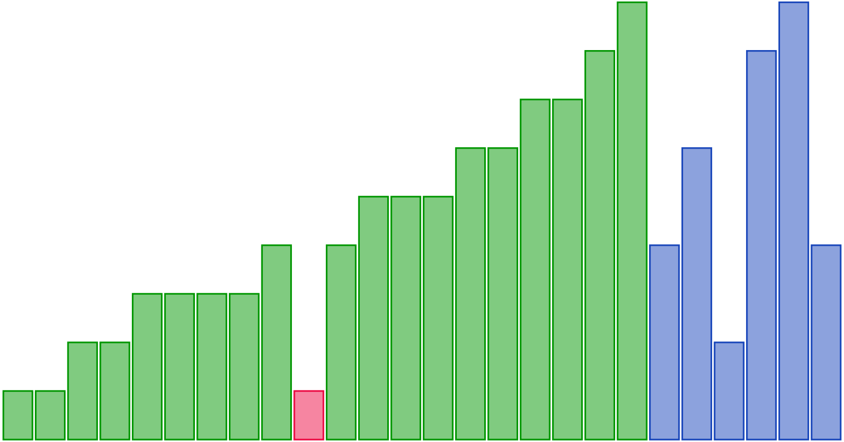


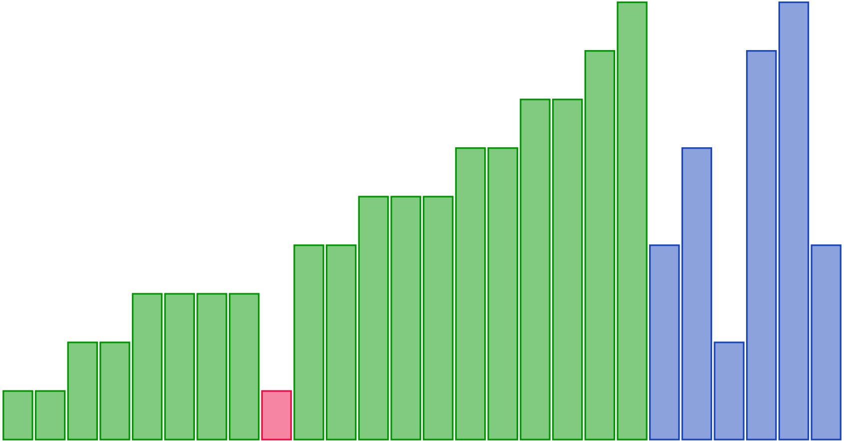


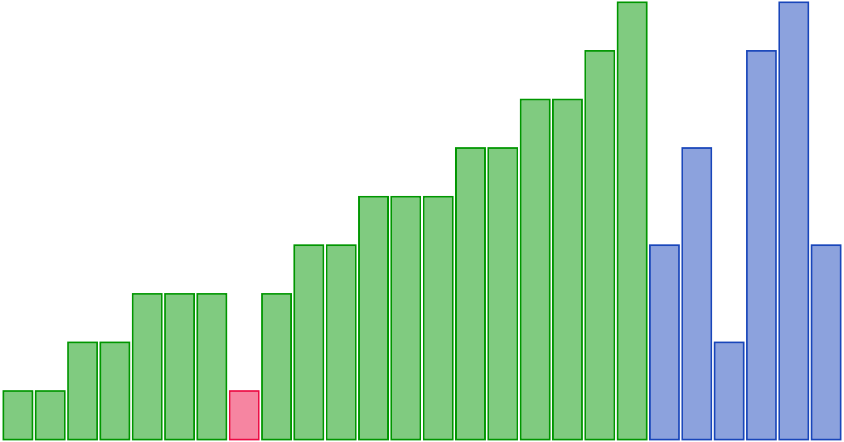


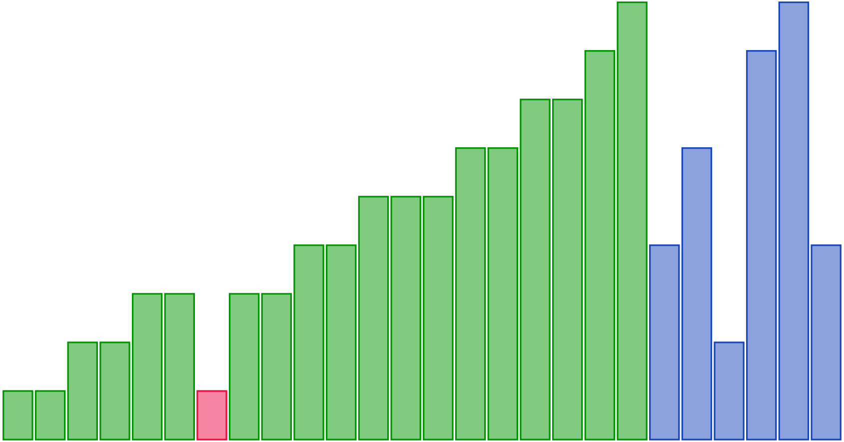


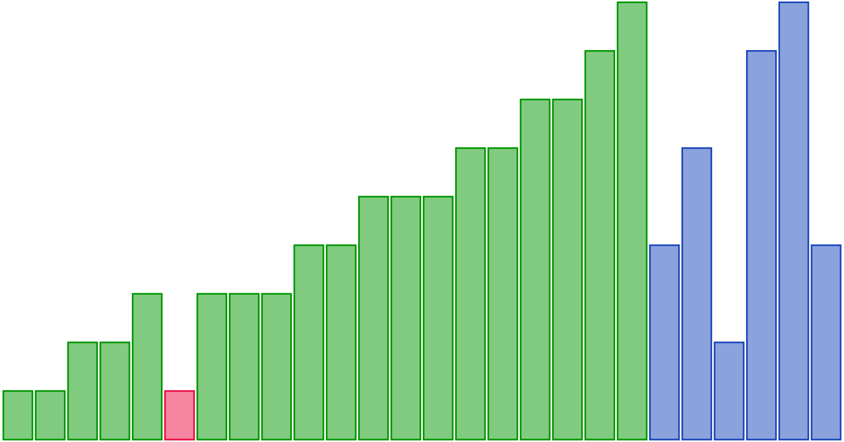


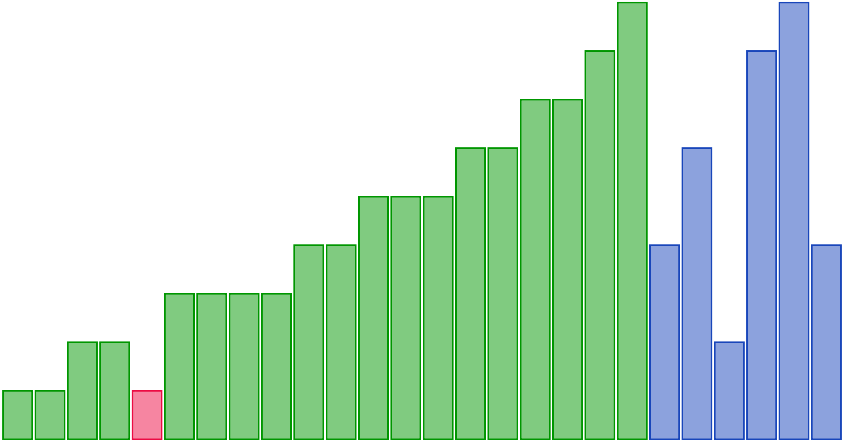


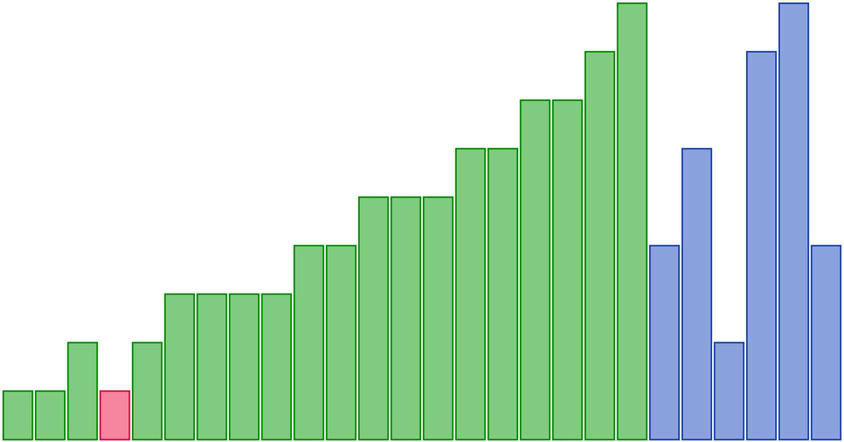


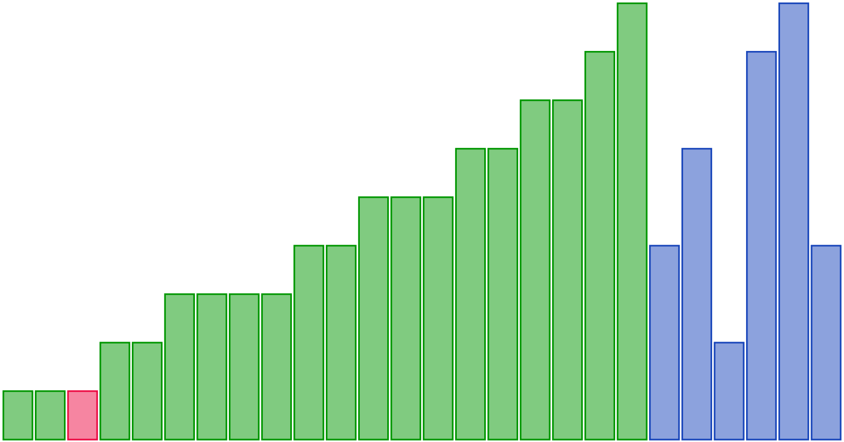


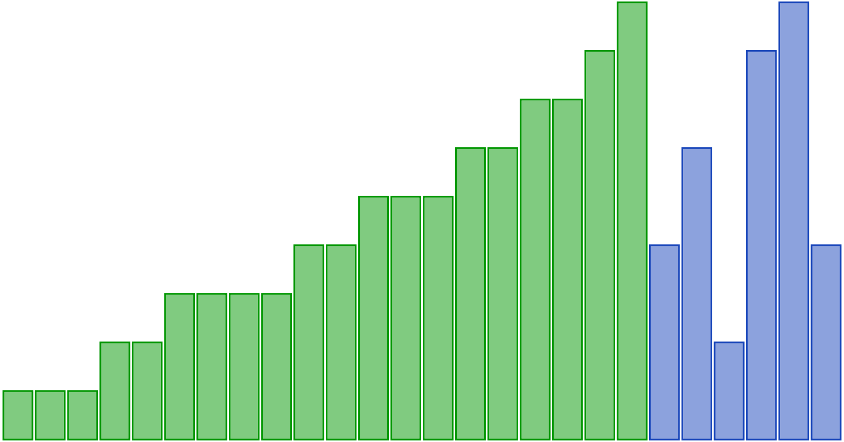


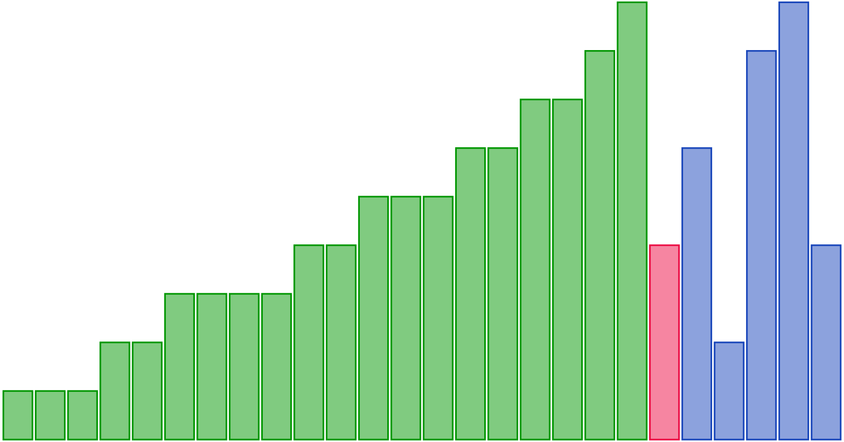


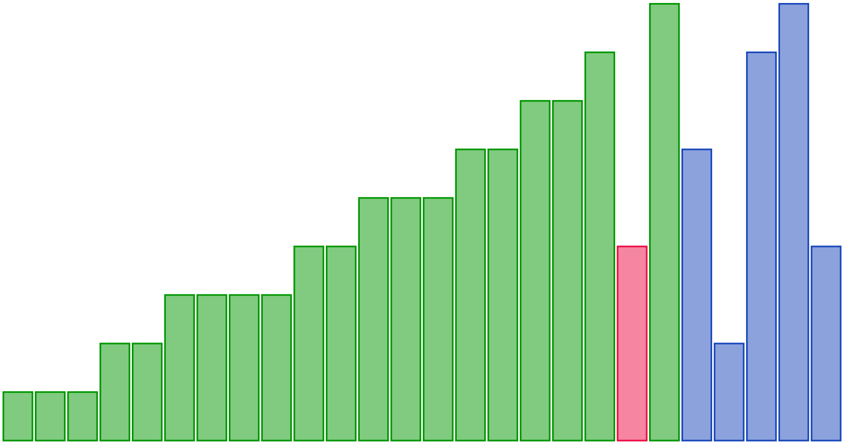


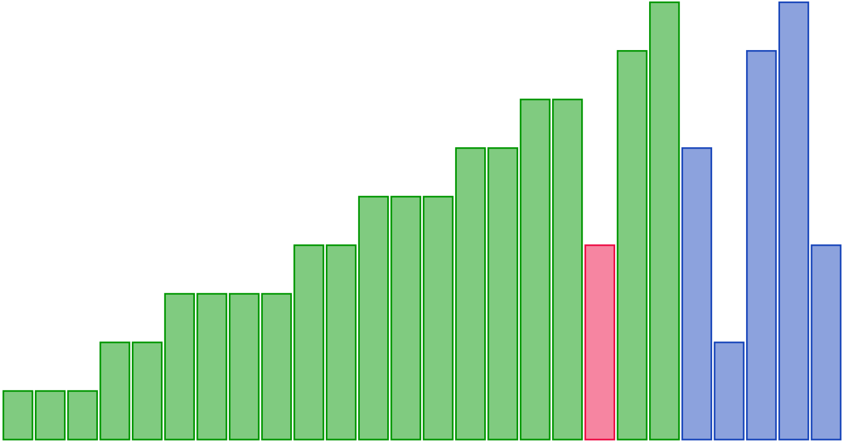


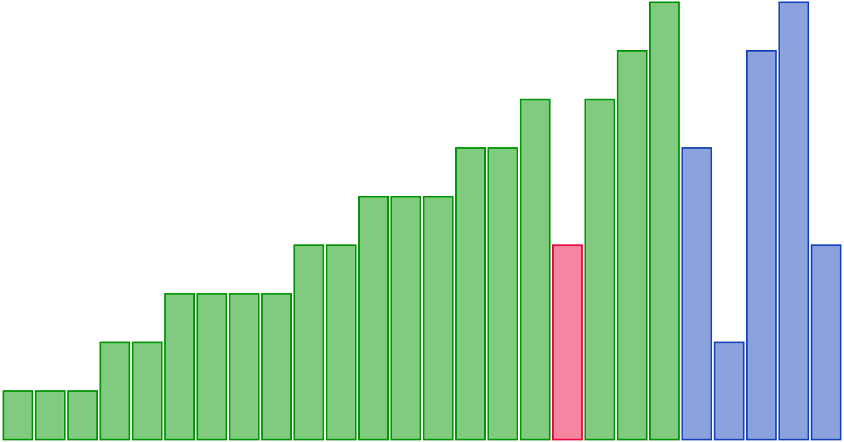


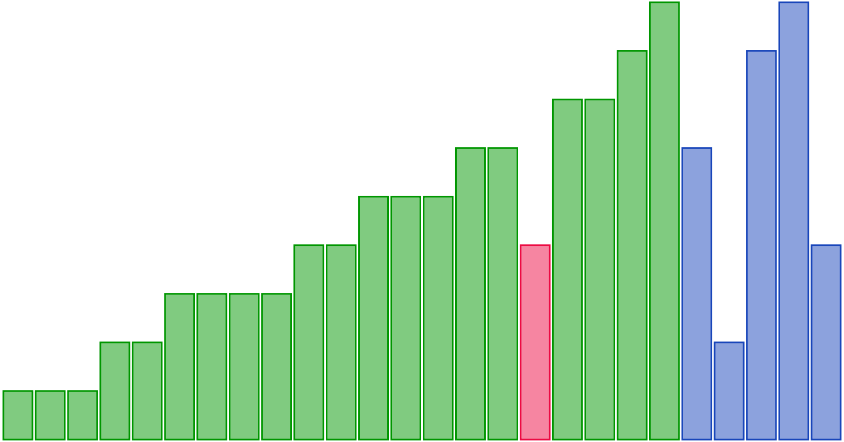


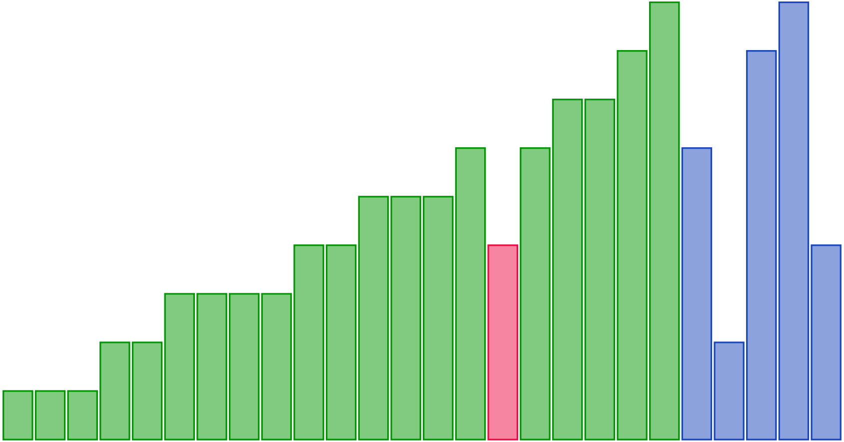




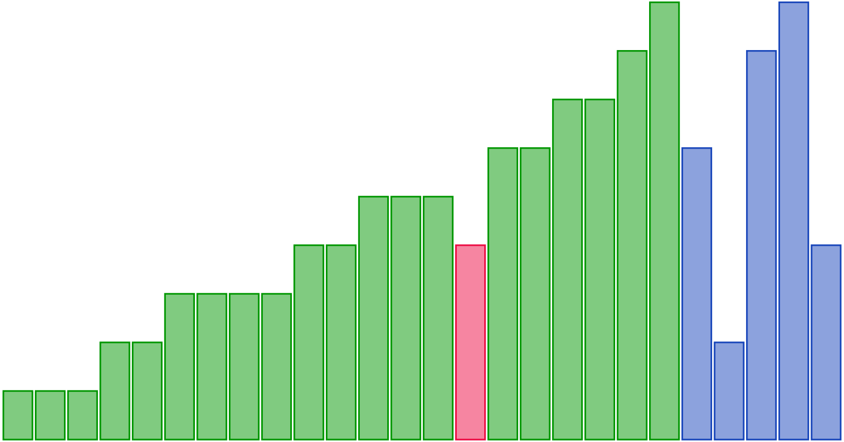


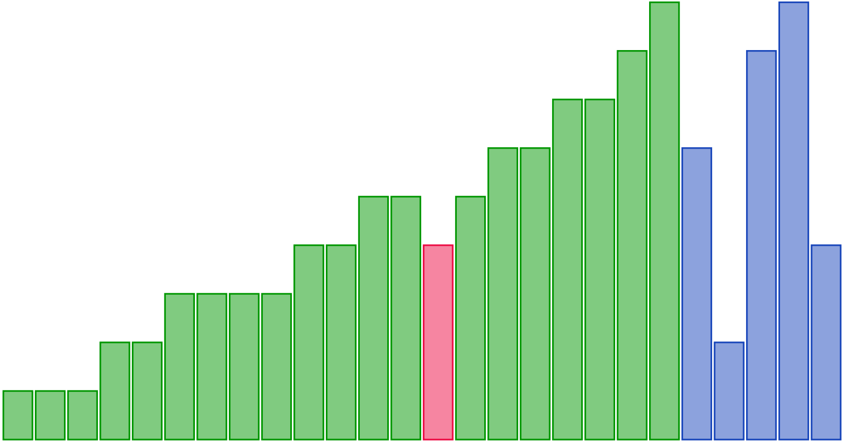


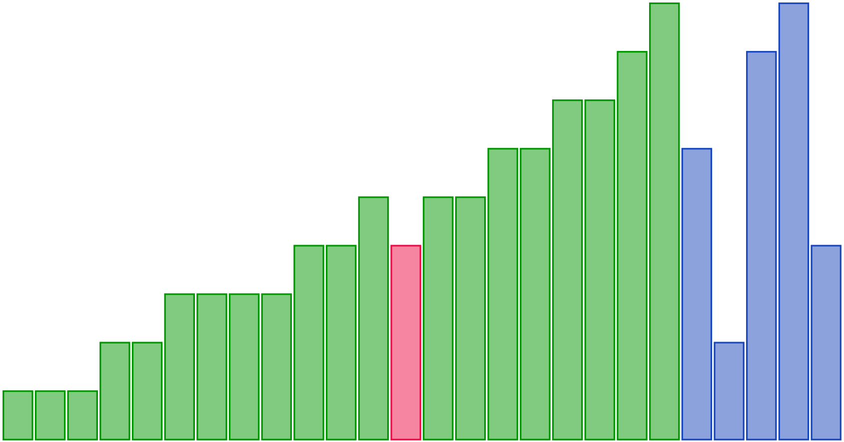


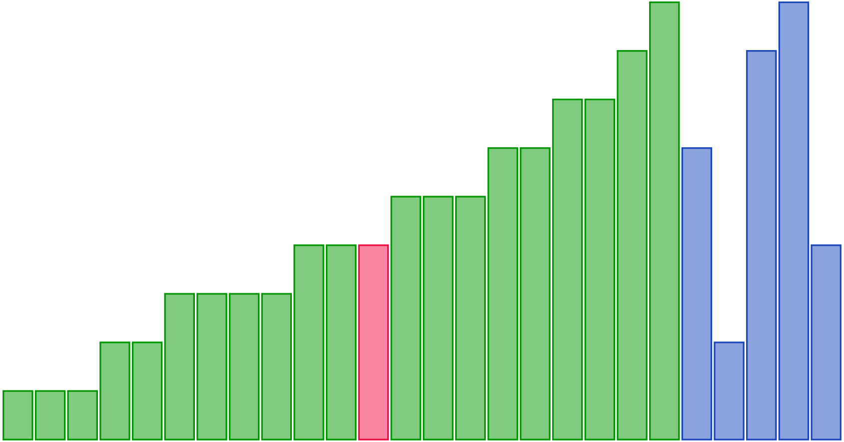


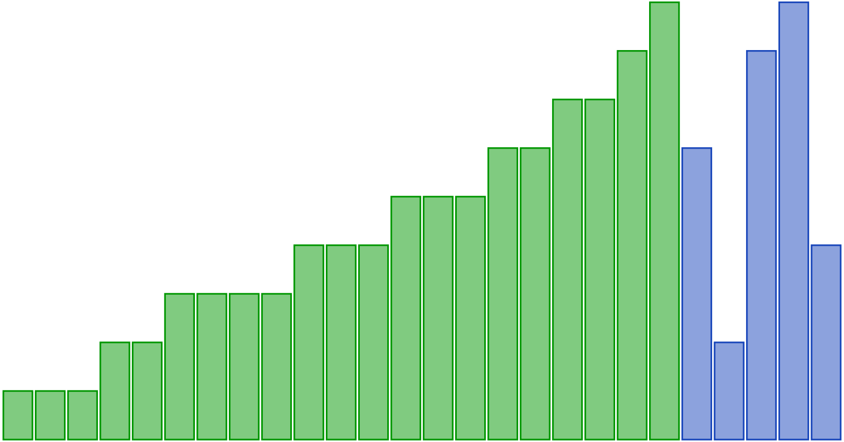
Insertion sort: Example

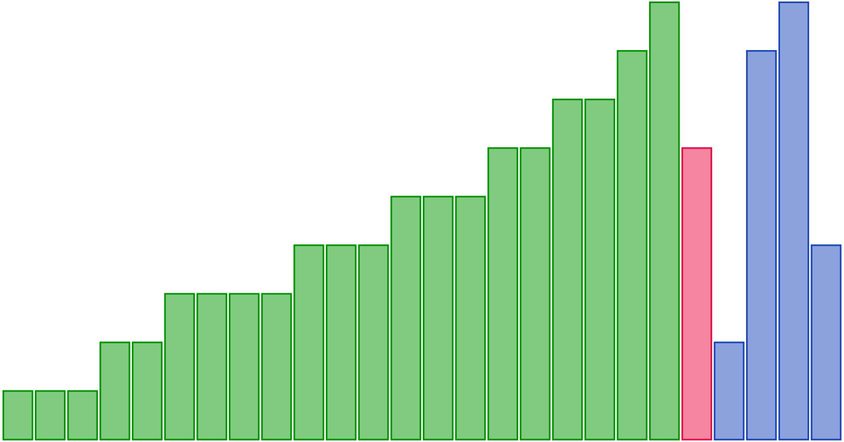


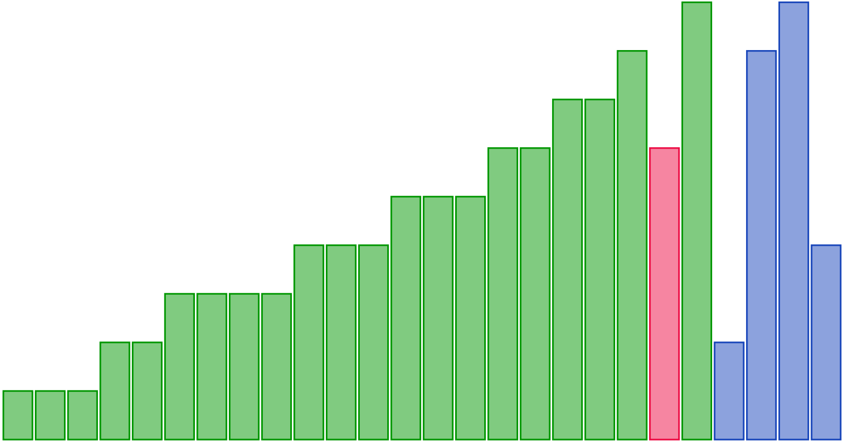


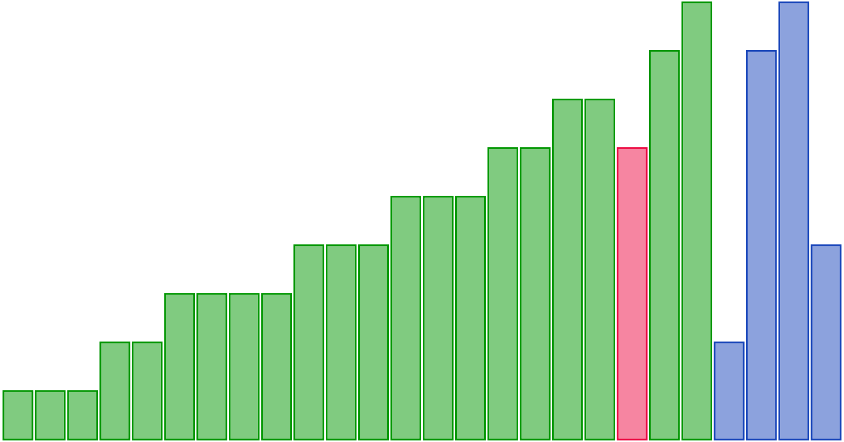


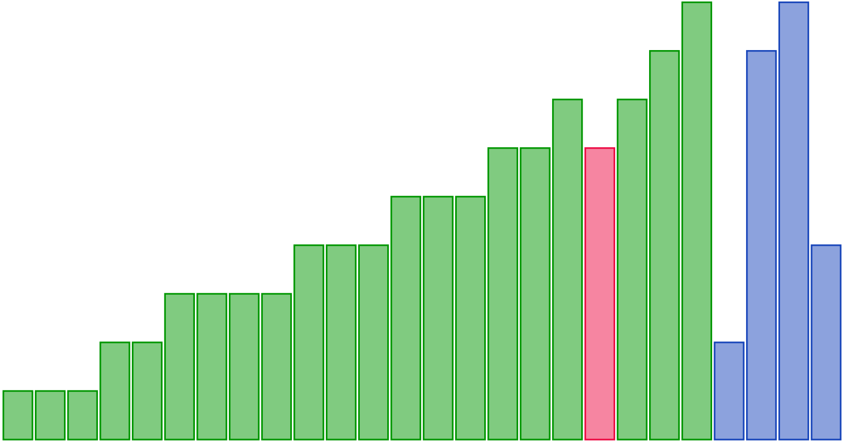


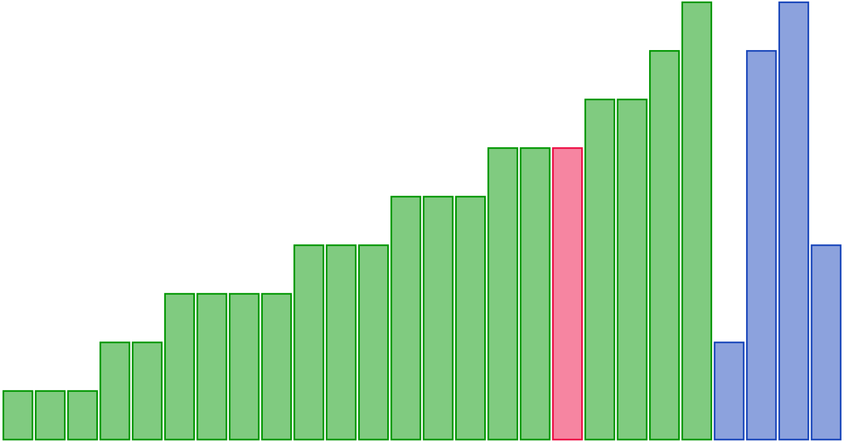


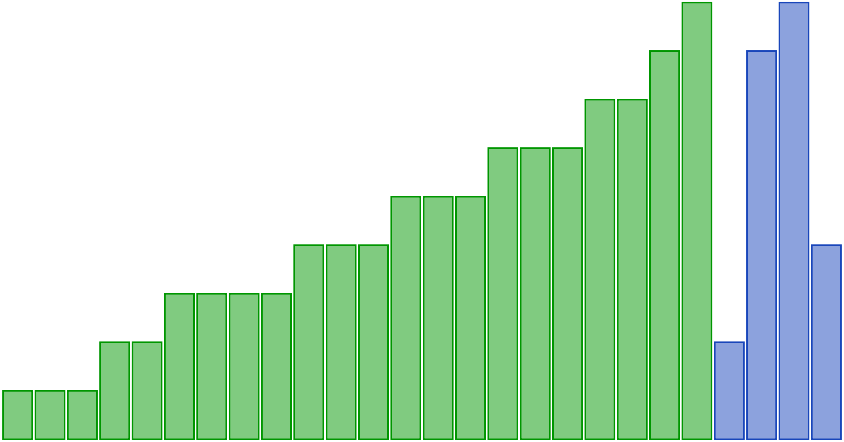


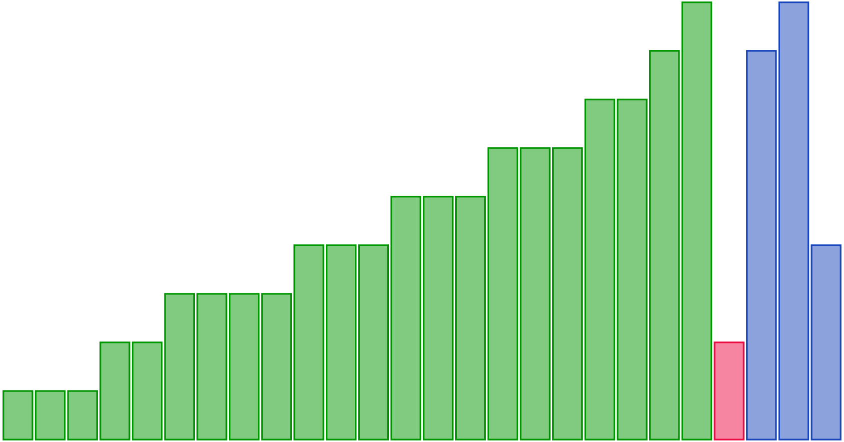


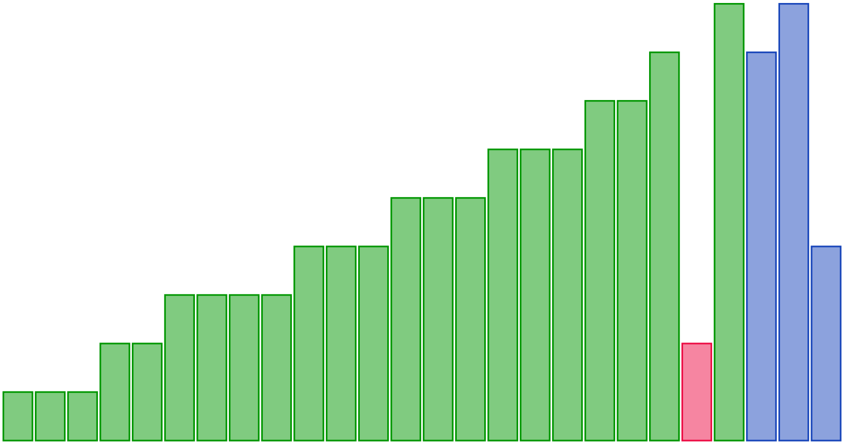


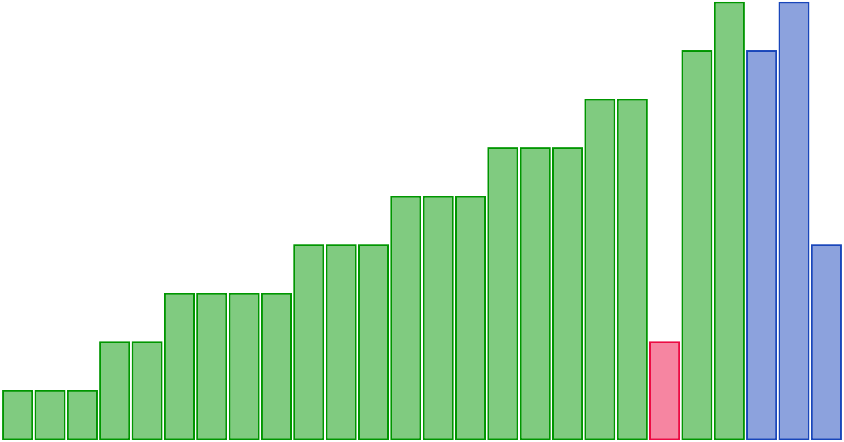


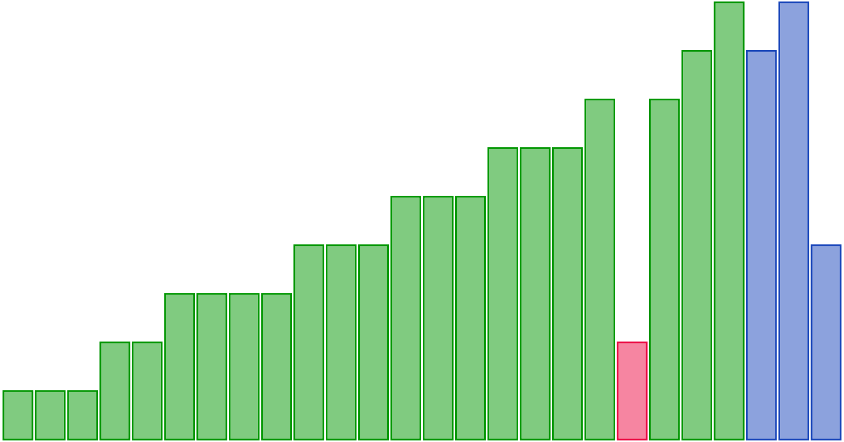


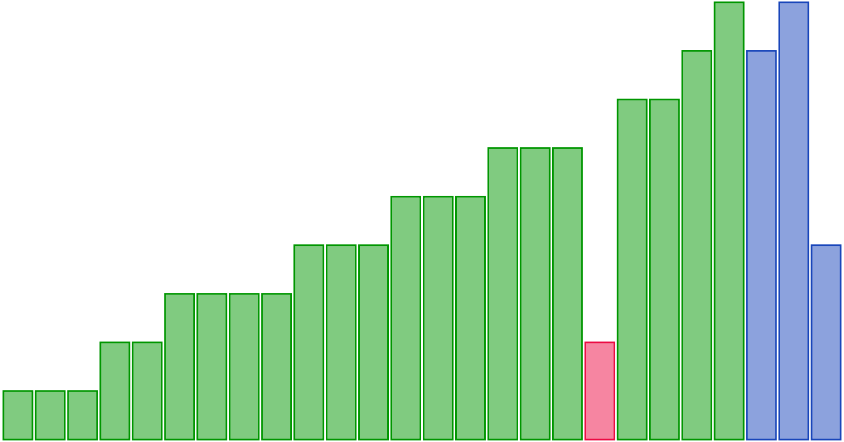


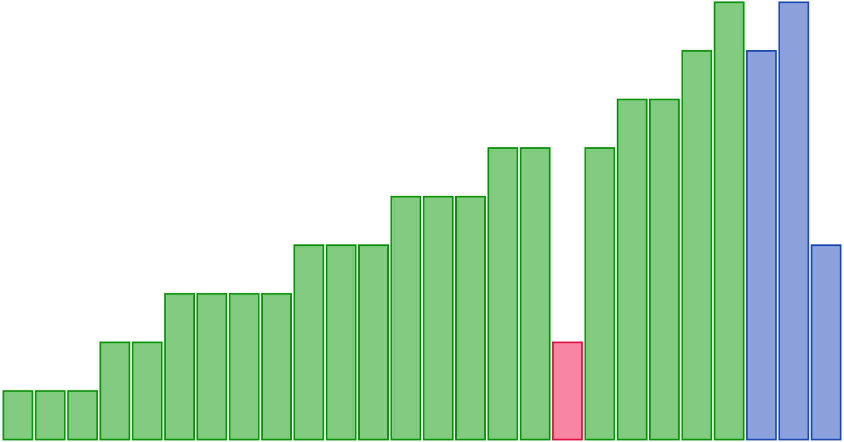


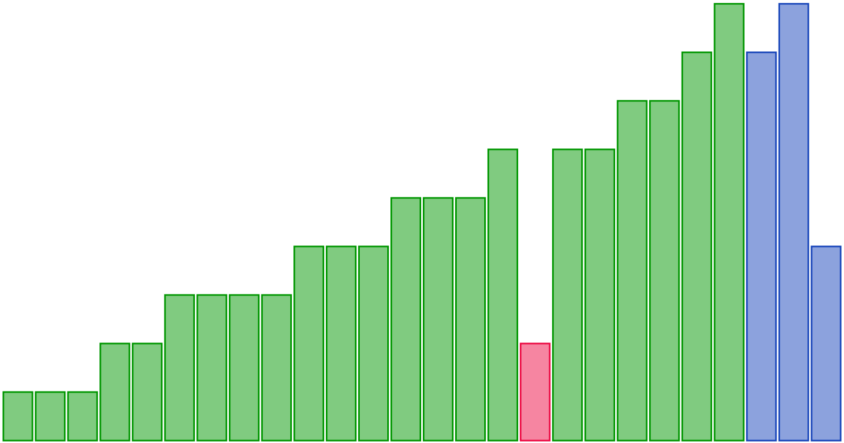


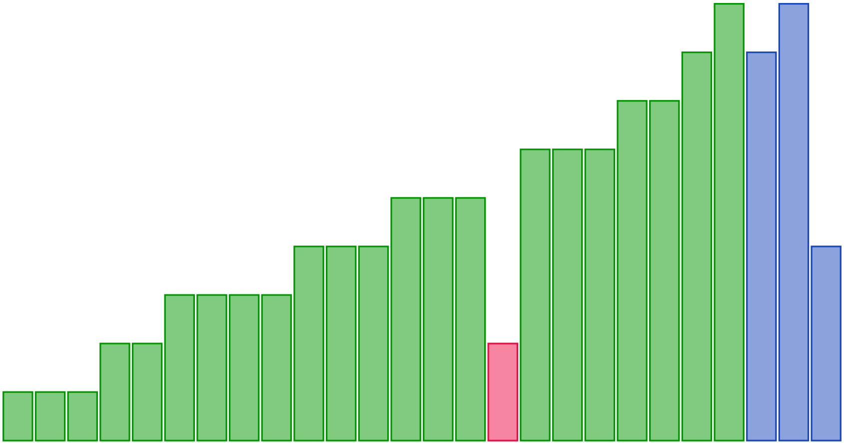


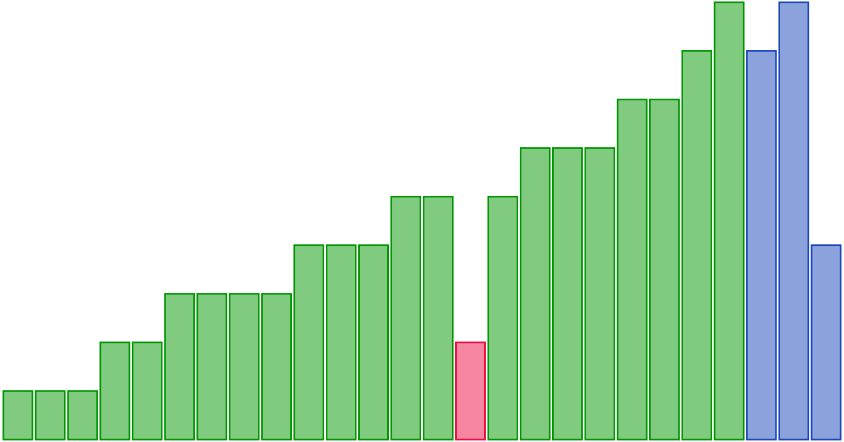


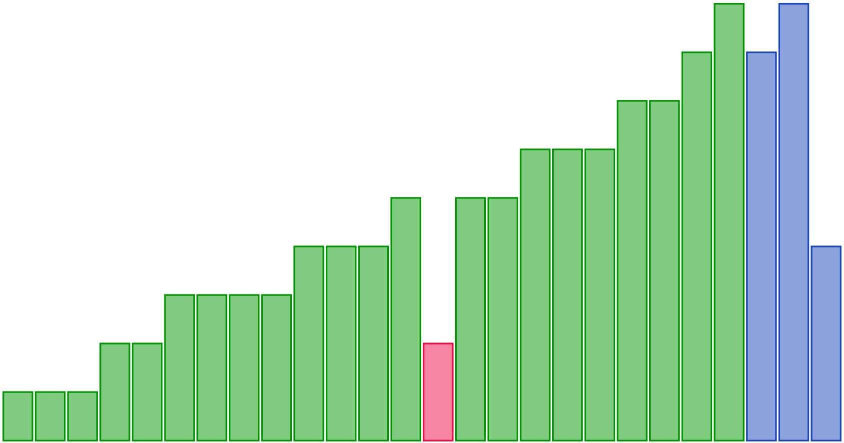


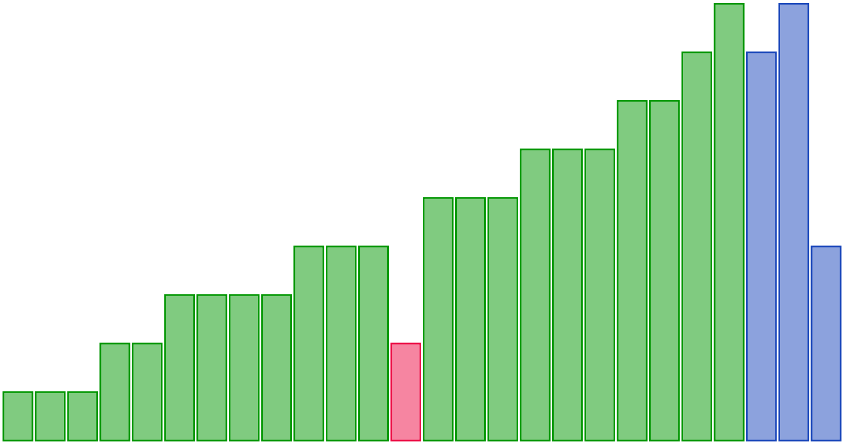


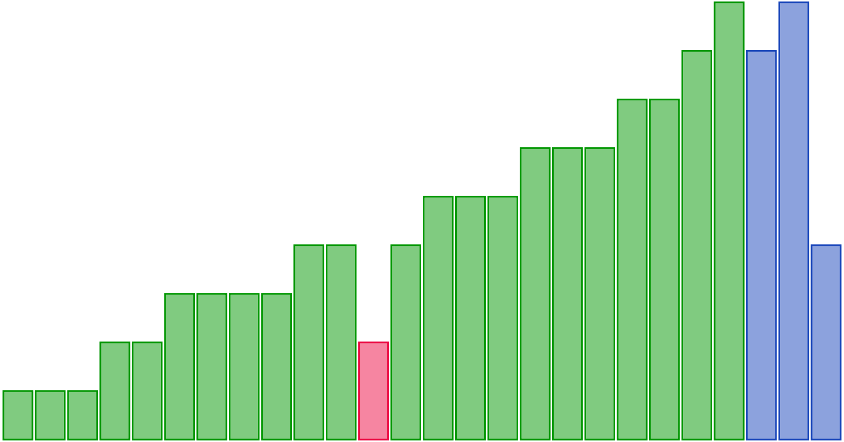


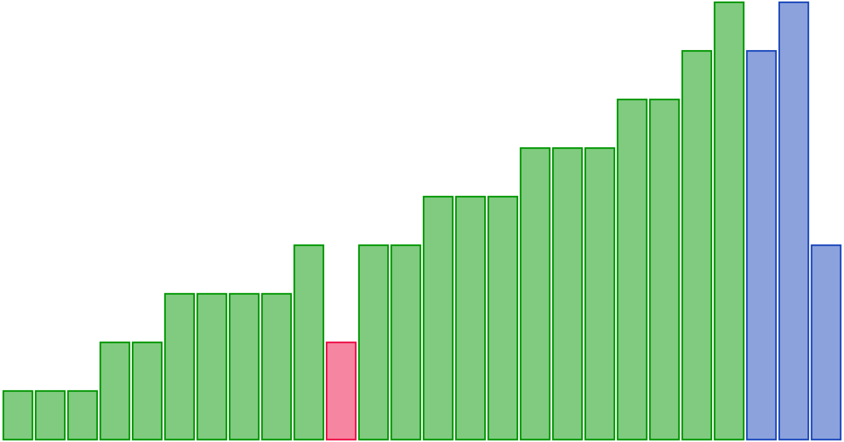


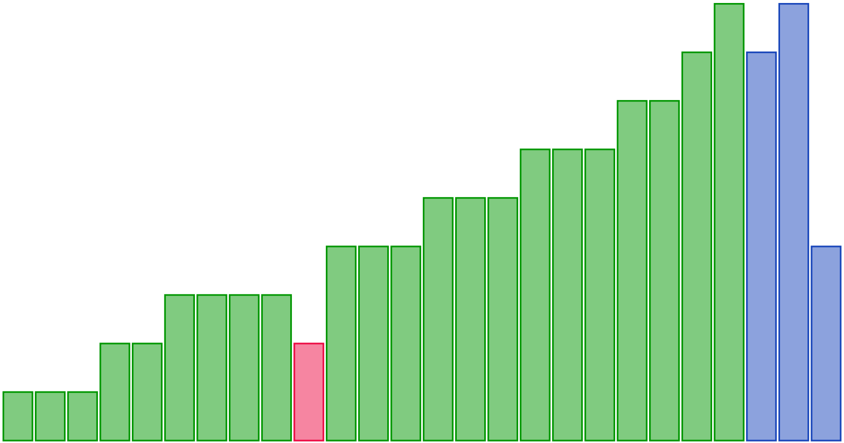


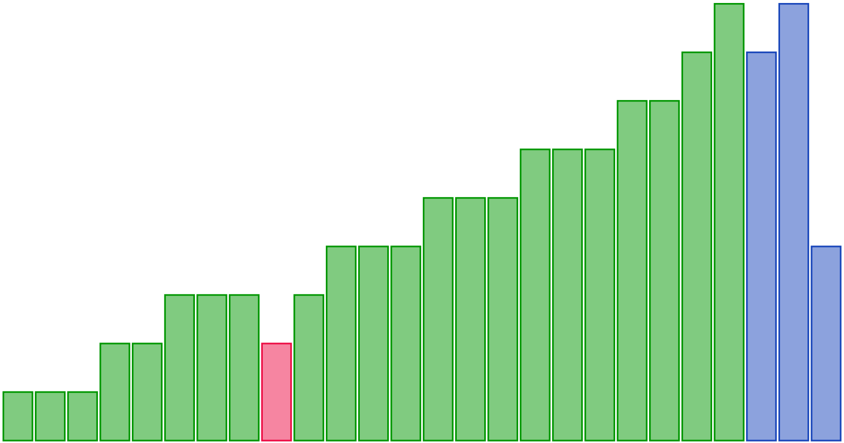


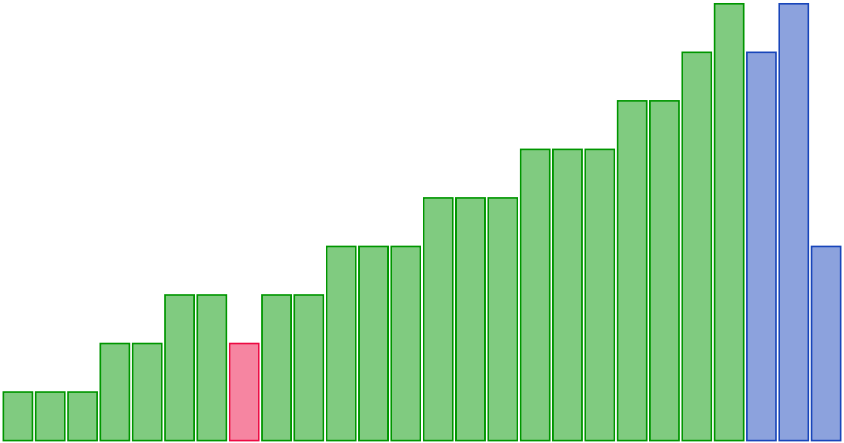


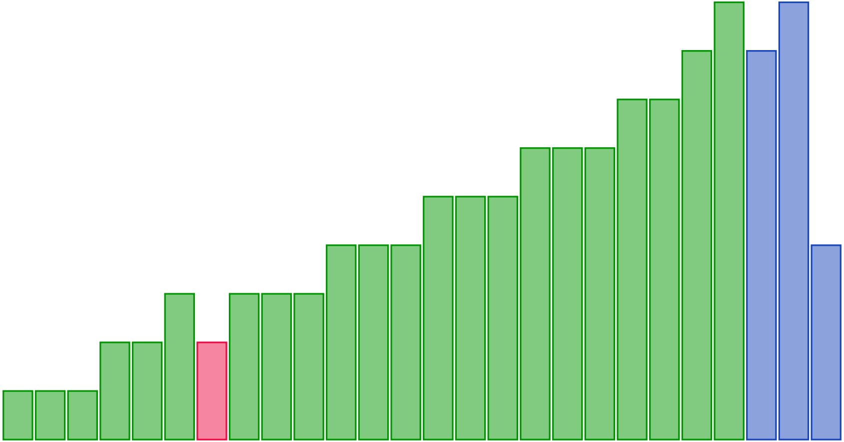


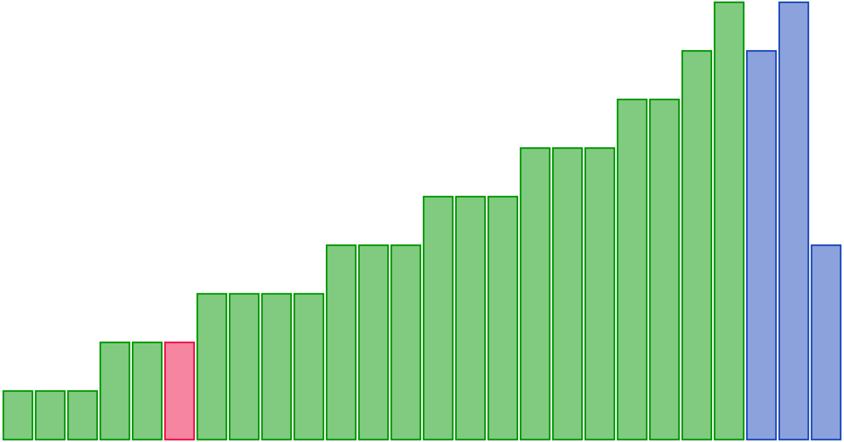


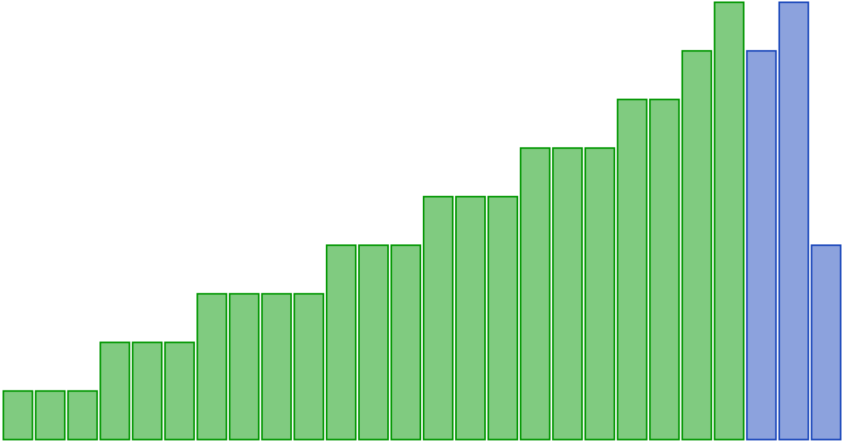


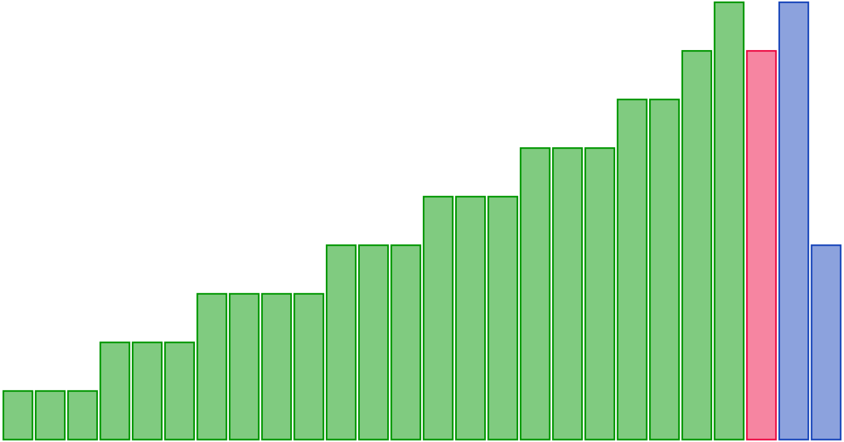


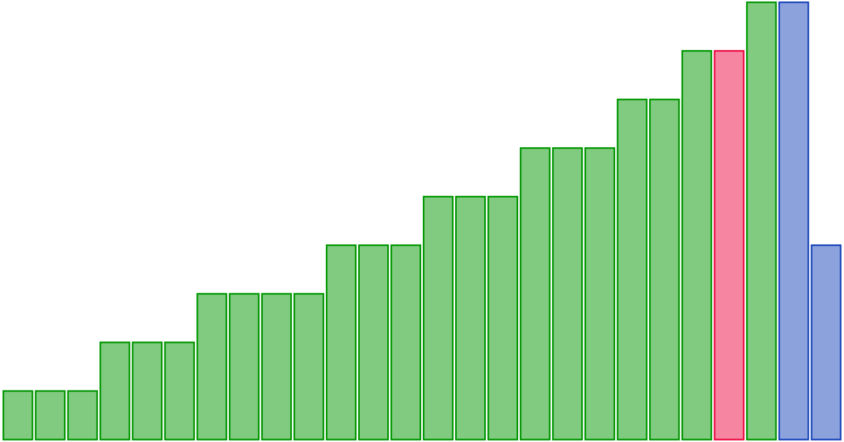


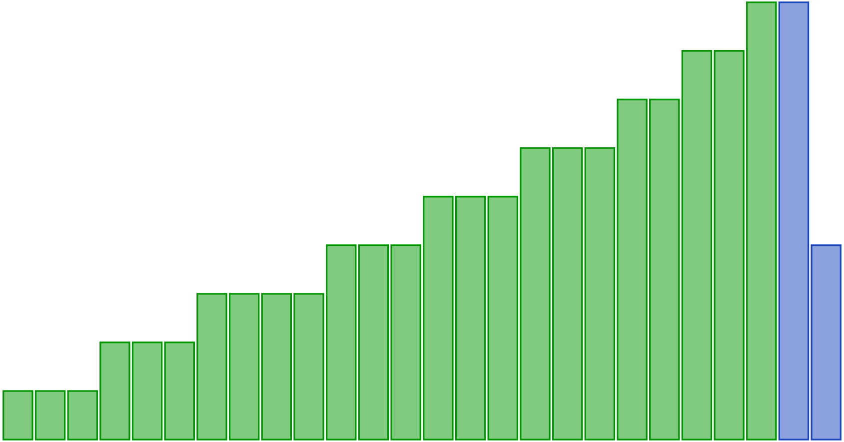


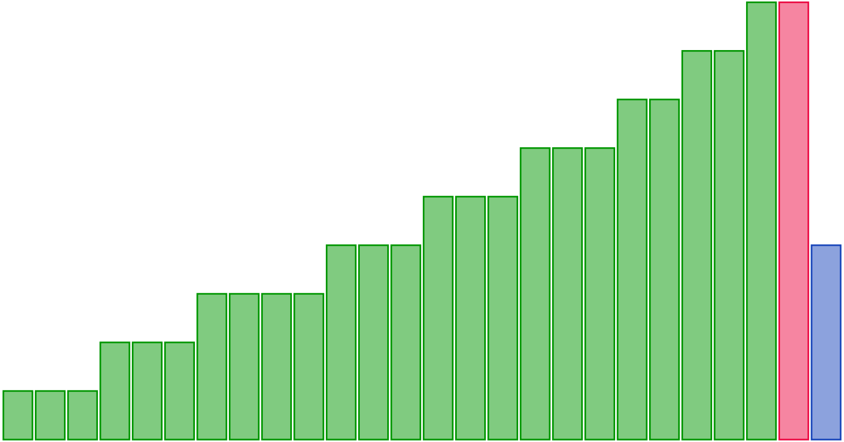


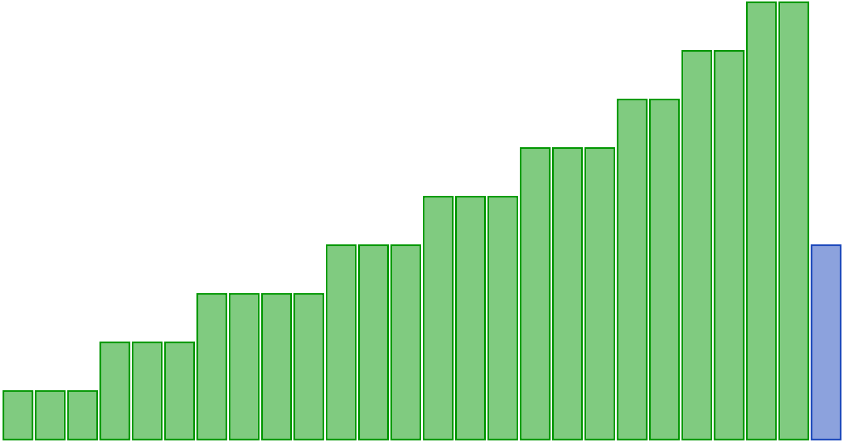


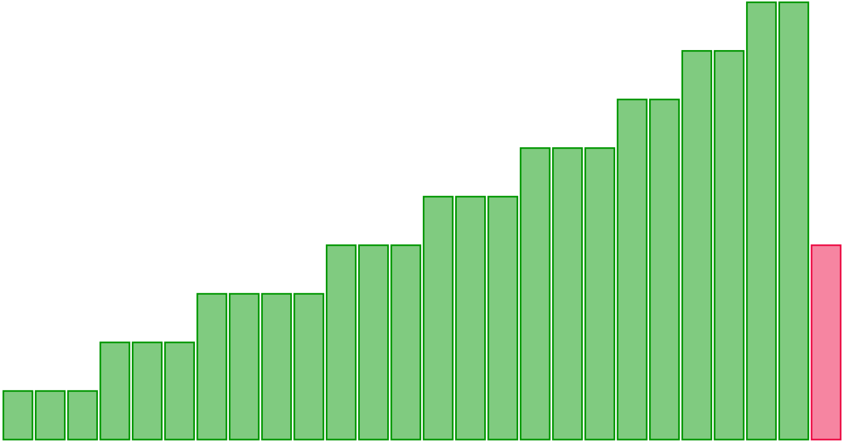


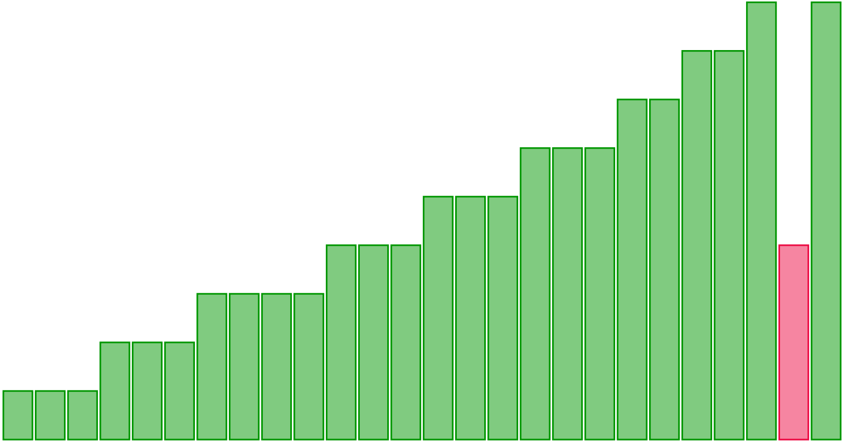


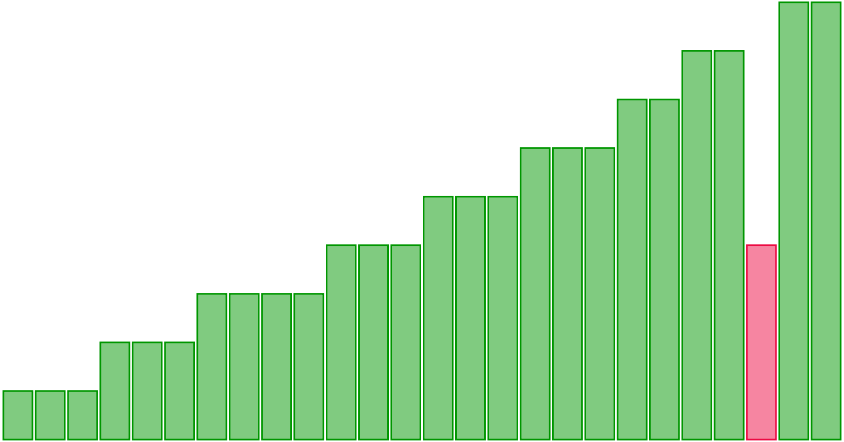


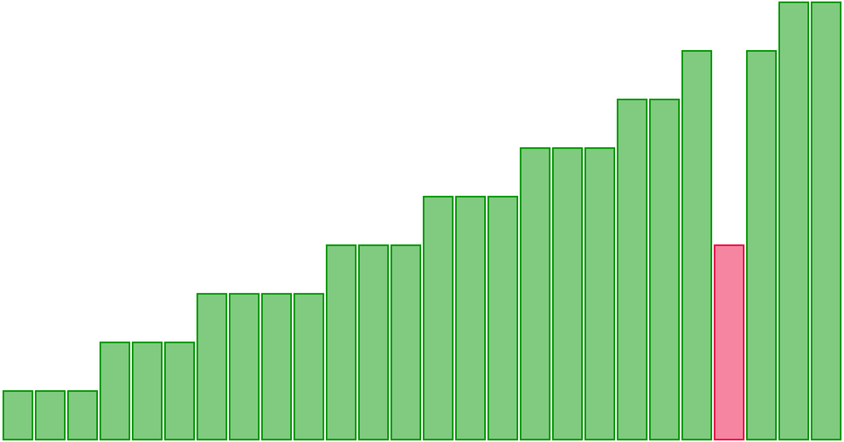


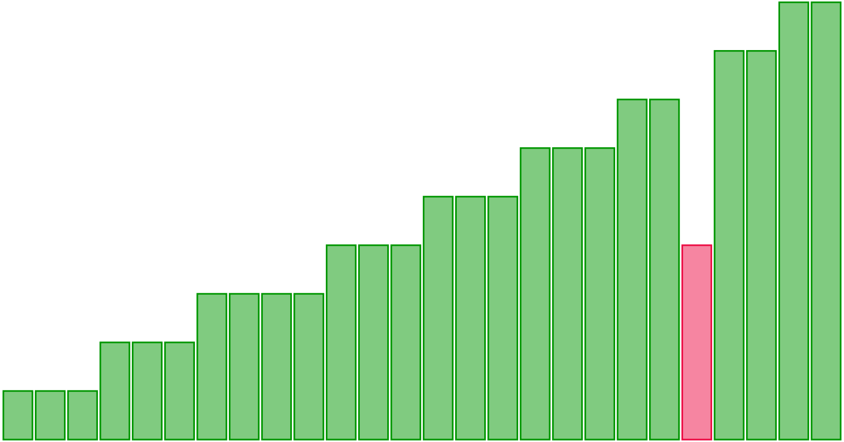


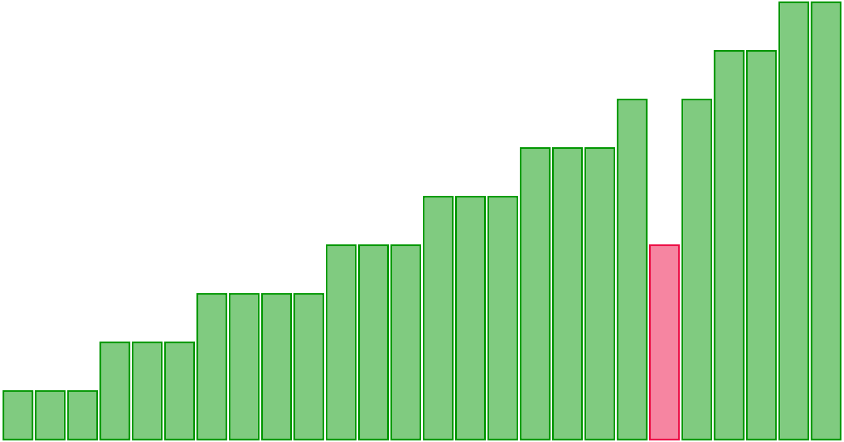


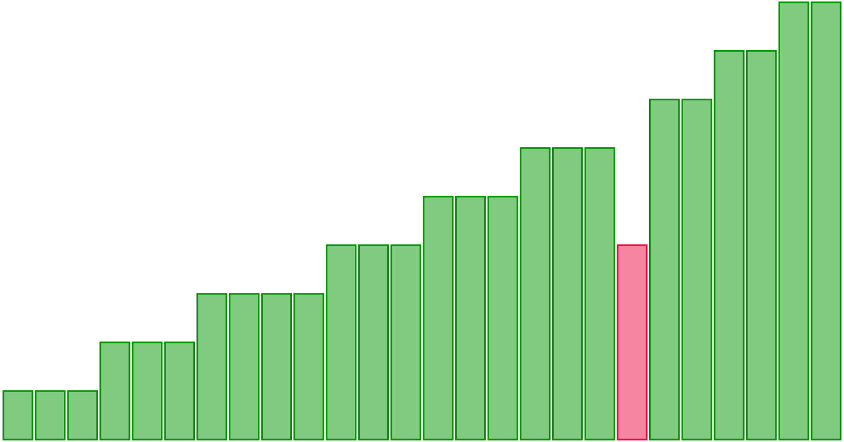


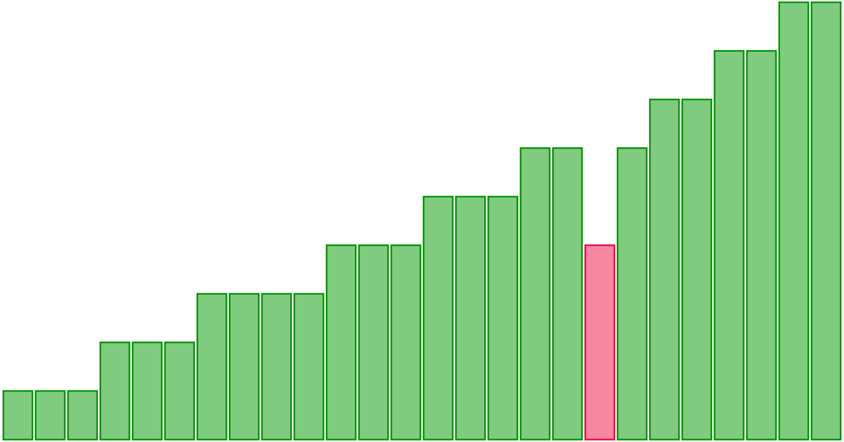


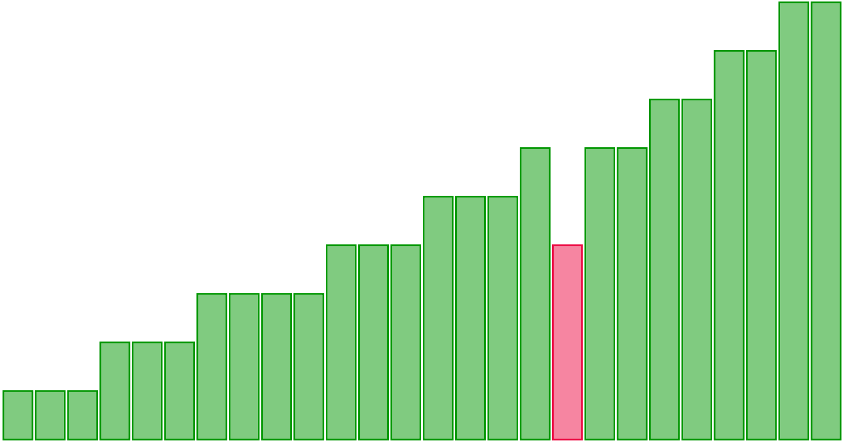


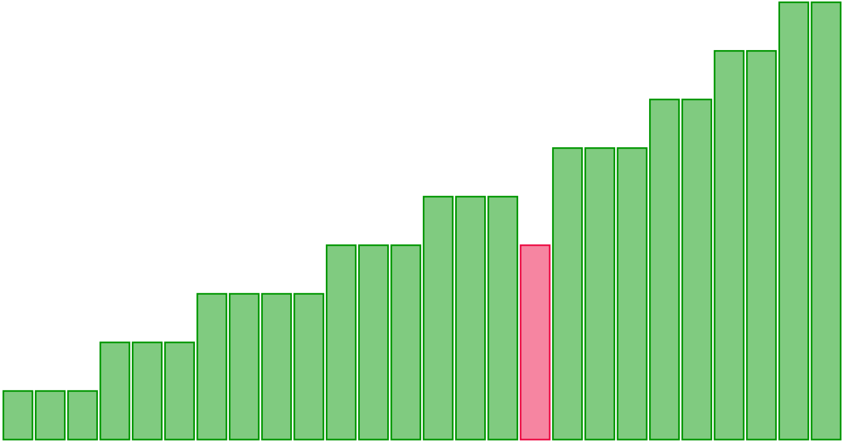


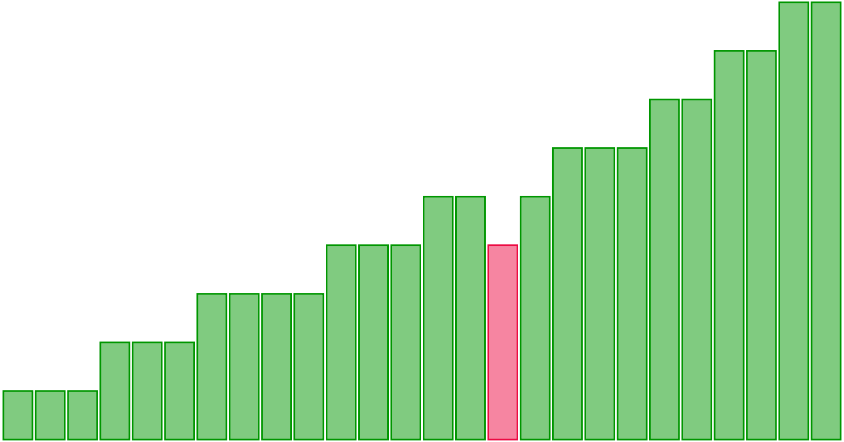


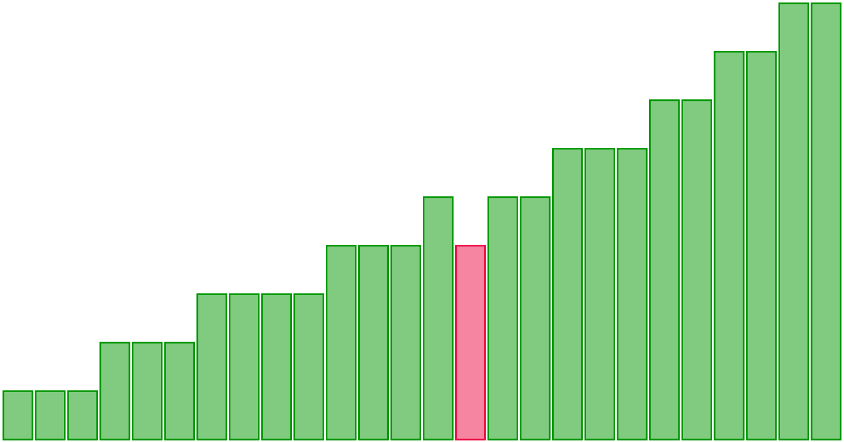


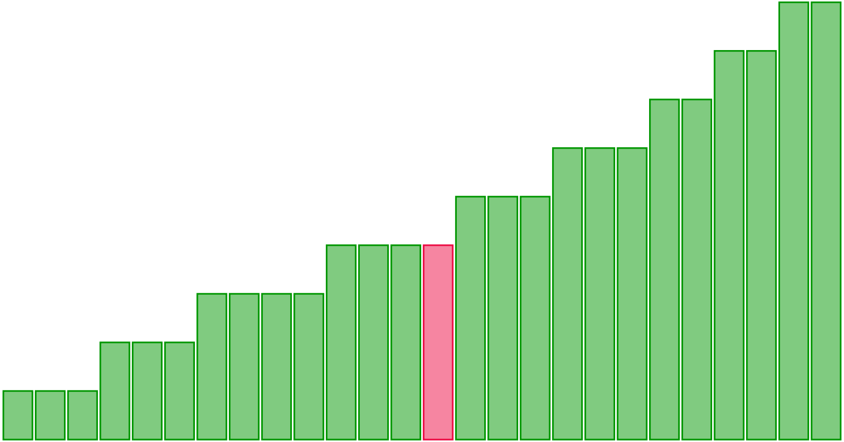


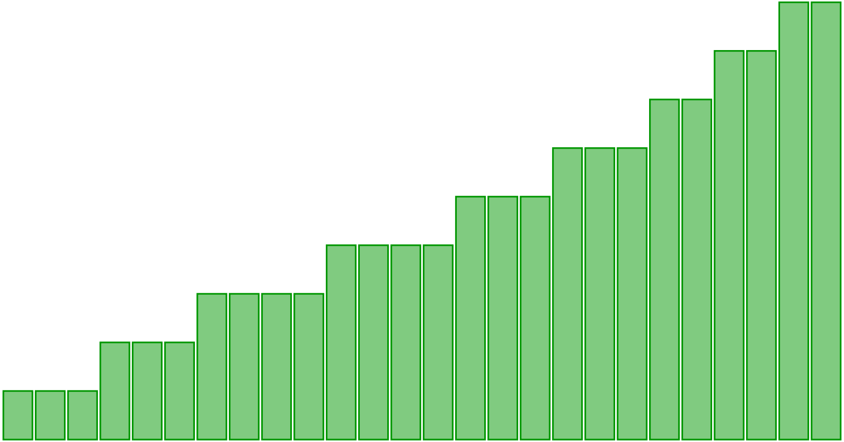












Theorem

After $t \geq 1$ iterations of the insertion sort, the $A[1:t]$ part of the input is sorted.

Theorem

After $t \geq 1$ iterations of the insertion sort, the $A[1:t]$ part of the input is sorted.

Proof.

We use mathematical induction.

Theorem

After $t \geq 1$ iterations of the insertion sort, the $A[1:t]$ part of the input is sorted.

Proof.

We use mathematical induction.

- **Induction base**. For $t = 1$, $A[1:1]$ consists of one element, thus it is sorted.

Theorem

After $t \geq 1$ iterations of the insertion sort, the $A[1:t]$ part of the input is sorted.

Proof.

We use mathematical induction.

- **Induction base**. For $t = 1$, $A[1:1]$ consists of one element, thus it is sorted.
- **Induction step**. Let $x = A[t]$. By induction assumption, $A[1:t-1]$ is sorted.

Theorem

After $t \geq 1$ iterations of the insertion sort, the $A[1:t]$ part of the input is sorted.

Proof.

We use mathematical induction.

- **Induction base**. For $t = 1$, $A[1:1]$ consists of one element, thus it is sorted.
- **Induction step**. Let $x = A[t]$. By induction assumption, $A[1:t-1]$ is sorted. Thus, there exists an index $j \in [1;t]$ such that:
  - for all $i < j$, $A[i] \leq x$
  - for all $i \geq j$, $A[i] > x$

Theorem

After $t \geq 1$ iterations of the insertion sort, the $A[1:t]$ part of the input is sorted.

Proof.

We use mathematical induction.

- **Induction base**. For $t = 1$, $A[1:1]$ consists of one element, thus it is sorted.
- **Induction step**. Let $x = A[t]$. By induction assumption, $A[1:t-1]$ is sorted. Thus, there exists an index $j \in [1;t]$ such that:
  - for all $i < j$, $A[i] \leq x$
  - for all $i \geq j$, $A[i] > x$

  The relative order of the elements from $A[1:t-1]$ is not changed while x is propagated backwards. So when $A[j]$ becomes x, $A[1:t]$ becomes ordered. □

Theorem

After $t \geq 1$ iterations of the insertion sort, the $A[1:t]$ part of the input is sorted.

Proof.

We use mathematical induction.

- **Induction base**. For $t = 1$, $A[1:1]$ consists of one element, thus it is sorted.
- **Induction step**. Let $x = A[t]$. By induction assumption, $A[1:t-1]$ is sorted. Thus, there exists an index $j \in [1;t]$ such that:
  - for all $i < j$, $A[i] \leq x$
  - for all $i \geq j$, $A[i] > x$

  The relative order of the elements from $A[1:t-1]$ is not changed while x is propagated backwards. So when $A[j]$ becomes x, $A[1:t]$ becomes ordered. □

Correctness of the insertion sort follows from this theorem with $t = |A|$.

- Let $N = |A|$
- Running time of a t-th iteration: at most $t - 1$ comparisons and swaps
  - At least one comparison for $t > 1$

- Let $N = |A|$
- Running time of a t-th iteration: at most $t - 1$ comparisons and swaps
  - At least one comparison for $t > 1$
- Upper bound on the total running time:
  $O\left(\sum_{i=1}^{N}(t-1)\right) = O\left(\frac{N(N-1)}{2}\right) = O(N^2)$

- Let $N = |A|$
- Running time of a t-th iteration: at most $t - 1$ comparisons and swaps
  - At least one comparison for $t > 1$
- Upper bound on the total running time:
  $O\left(\sum_{i=1}^{N}(t-1)\right) = O\left(\frac{N(N-1)}{2}\right) = O(N^2)$
- Lower bound on the total running time: $\Omega\left(\sum_{i=2}^{N} 1\right) = \Omega(N)$

- Let $N = |A|$
- Running time of a t-th iteration: at most $t - 1$ comparisons and swaps
  - At least one comparison for $t > 1$
- Upper bound on the total running time:
  $O\left(\sum_{i=1}^{N}(t-1)\right) = O\left(\frac{N(N-1)}{2}\right) = O(N^2)$
- Lower bound on the total running time: $\Omega\left(\sum_{i=2}^{N} 1\right) = \Omega(N)$
- Both bounds are strict:
  - Best case: $A = [1, 2, \ldots, N-1, N]$
  - Worst case: $A = [N, N-1, \ldots, 2, 1]$

- Let $N = |A|$
- Running time of a t-th iteration: at most $t - 1$ comparisons and swaps
  - At least one comparison for $t > 1$
- Upper bound on the total running time:
  $O\left(\sum_{i=1}^{N}(t-1)\right) = O\left(\frac{N(N-1)}{2}\right) = O(N^2)$
- Lower bound on the total running time: $\Omega\left(\sum_{i=2}^{N} 1\right) = \Omega(N)$
- Both bounds are strict:
  - Best case: $A = [1, 2, \ldots, N-1, N]$
  - Worst case: $A = [N, N-1, \ldots, 2, 1]$
- What about the average case?

A more precise running time estimation. . .

A more precise running time estimation...

Inversion: the number of situations when $i < j$ and $A_i > A_j$

A more precise running time estimation...

Inversion: the number of situations when $i < j$ and $A_i > A_j$

Observations:

- The number of swaps in the insertion sort is equal to the number of inversions
  - Each swap decreases the number of inversions by one
  - At the end, we have a sorted array, which has zero inversions

A more precise running time estimation...

Inversion: the number of situations when $i < j$ and $A_i > A_j$

Observations:

- The number of swaps in the insertion sort is equal to the number of inversions
  - Each swap decreases the number of inversions by one
  - At the end, we have a sorted array, which has zero inversions
- The running time is proportional to the number of inversions
  - ... plus at most $N - 1$ comparisons which do not result in swaps

A more precise running time estimation...

Inversion: the number of situations when $i < j$ and $A_i > A_j$

Observations:

- The number of swaps in the insertion sort is equal to the number of inversions
  - Each swap decreases the number of inversions by one
  - At the end, we have a sorted array, which has zero inversions
- The running time is proportional to the number of inversions
  - ... plus at most $N - 1$ comparisons which do not result in swaps
- The average running time of the insertion sort over all permutations is $\Theta(N^2)$

A more precise running time estimation...

Inversion: the number of situations when $i < j$ and $A_i > A_j$

Observations:

- The number of swaps in the insertion sort is equal to the number of inversions
  - Each swap decreases the number of inversions by one
  - At the end, we have a sorted array, which has zero inversions
- The running time is proportional to the number of inversions
  - ... plus at most $N - 1$ comparisons which do not result in swaps
- The average running time of the insertion sort over all permutations is $\Theta(N^2)$
  - Count the total number of inversions in all permutations

A more precise running time estimation...
Inversion: the number of situations when $i < j$ and $A_i > A_j$
Observations:

- The number of swaps in the insertion sort is equal to the number of inversions
  - Each swap decreases the number of inversions by one
  - At the end, we have a sorted array, which has zero inversions
- The running time is proportional to the number of inversions
  - ... plus at most $N - 1$ comparisons which do not result in swaps
- The average running time of the insertion sort over all permutations is $\Theta(N^2)$
  - Count the total number of inversions in all permutations
    - Consider two arbitrary indices $1 \le i < j \le N$
    - Each permutation with $A_i < A_j$ has $= 1$ corresponding one with $A_i > A_j$
    - $N!/2$ permutations with inversion on i and j

A more precise running time estimation: $\Theta(N^2)$ on average

Inversion: the number of situations when $i < j$ and $A_i > A_j$

Observations:

- The number of swaps in the insertion sort is equal to the number of inversions
  - Each swap decreases the number of inversions by one
  - At the end, we have a sorted array, which has zero inversions
- The running time is proportional to the number of inversions
  - ... plus at most $N - 1$ comparisons which do not result in swaps
- The average running time of the insertion sort over all permutations is $\Theta(N^2)$
  - Count the total number of inversions in all permutations
    - Consider two arbitrary indices $1 \leq i < j \leq N$
    - Each permutation with $A_i < A_j$ has $= 1$ corresponding one with $A_i > A_j$
    - $N!/2$ permutations with inversion on i and j
  - Average number of inversions per permutation: $\frac{N!}{2} \cdot \frac{N(N-1)}{2} \cdot \frac{1}{N!} = \frac{N(N-1)}{4} = \Theta(N^2)$