ITMO UNIVERSITY

How to Win Coding Competitions: Secrets of Champions

Week 3: Sorting and Search Algorithms
Lecture 4: Quicksort

Maxim Buzdalov
Saint Petersburg 2016

Previous sorting algorithm: Insertion sort

- Incremental: size of the sorted part increases by one each time
- Can only swap adjacent elements
- Running time: $\Omega(N)$, $O(N^2)$, $\Theta(N^2)$ on average

Previous sorting algorithm: Insertion sort

- Incremental: size of the sorted part increases by one each time
- Can only swap adjacent elements
- Running time: $\Omega(N)$, $O(N^2)$, $\Theta(N^2)$ on average
- Can we do it faster?

Previous sorting algorithm: Insertion sort

- Incremental: size of the sorted part increases by one each time
- Can only swap adjacent elements
- Running time: $\Omega(N)$, $O(N^2)$, $\Theta(N^2)$ on average
- Can we do it faster?

Meet Quicksort!

Previous sorting algorithm: Insertion sort

- Incremental: size of the sorted part increases by one each time
- Can only swap adjacent elements
- Running time: $\Omega(N)$, $O(N^2)$, $\Theta(N^2)$ on average
- Can we do it faster?

Meet Quicksort! Author: Tony Hoare, 1959

Previous sorting algorithm: Insertion sort

- Incremental: size of the sorted part increases by one each time
- Can only swap adjacent elements
- Running time: $\Omega(N)$, $O(N^2)$, $\Theta(N^2)$ on average
- Can we do it faster?

Meet Quicksort! Author: Tony Hoare, 1959

Idea of the algorithm:

- Split the array into two parts L and R, such that $L_i \leq R_j$ for all i and j
- Sort the parts recursively $\rightarrow$ the entire array is sorted!

Previous sorting algorithm: Insertion sort

- Incremental: size of the sorted part increases by one each time
- Can only swap adjacent elements
- Running time: $\Omega(N)$, $O(N^2)$, $\Theta(N^2)$ on average
- Can we do it faster?

Meet Quicksort! Author: Tony Hoare, 1959

Idea of the algorithm:

- Split the array into two parts L and R, such that $L_i \leq R_j$ for all i and j
- Sort the parts recursively $\rightarrow$ the entire array is sorted!
  - The Divide-and-Conquer approach

Previous sorting algorithm: Insertion sort

- Incremental: size of the sorted part increases by one each time
- Can only swap adjacent elements
- Running time: $\Omega(N)$, $O(N^2)$, $\Theta(N^2)$ on average
- Can we do it faster?

Meet Quicksort! Author: Tony Hoare, 1959

Idea of the algorithm:

- Split the array into two parts L and R, such that $L_i \leq R_j$ for all i and j
- Sort the parts recursively $\to$ the entire array is sorted!
  - The Divide-and-Conquer approach
- For best results, these parts should be approximately equal

procedure $\textsc{Quicksort}(A, \prec, s, e)$
  $s' \leftarrow s,\ e' \leftarrow e,\ M \leftarrow A[(s+e)/2]$
  while $s' \le e'$ **do**
    while $A[s'] \prec M$ **do** $s' \leftarrow s' + 1$ **end while**
    while $M \prec A[e']$ **do** $e' \leftarrow e' - 1$ **end while**
    if $s' \le e'$ **then**
      $A[s'] \Leftrightarrow A[e']$
      $s' \leftarrow s' + 1,\ e' \leftarrow e' - 1$
    end if
  end while
  if $s \le e'$ **then** $\textsc{Quicksort}(A, \prec, s, e')$ **end if**
  if $s' \le e$ **then** $\textsc{Quicksort}(A, \prec, s', e)$ **end if**
end procedure

procedure $\textsc{Quicksort}(A, \prec, s, e)$
    $s' \leftarrow s$, $e' \leftarrow e$, $M \leftarrow A[(s+e)/2]$ $\triangleright$ M: the pivot value.
    while $s' \le e'$ **do**
        while $A[s'] \prec M$ **do** $s' \leftarrow s' + 1$ **end while**
        while $M \prec A[e']$ **do** $e' \leftarrow e' - 1$ **end while**
        if $s' \le e'$ **then**
            $A[s'] \Leftrightarrow A[e']$
            $s' \leftarrow s' + 1$, $e' \leftarrow e' - 1$
        end if
    end while
    if $s \le e'$ **then** $\textsc{Quicksort}(A, \prec, s, e')$ **end if**
    if $s' \le e$ **then** $\textsc{Quicksort}(A, \prec, s', e)$ **end if**
end procedure

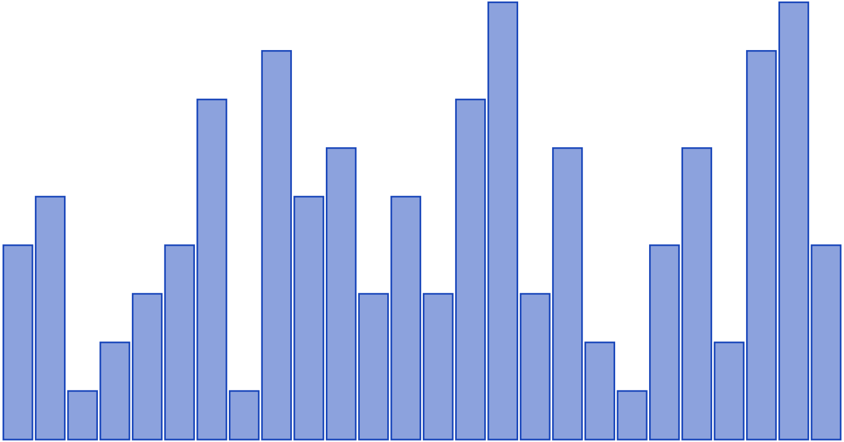
procedure $\textsc{Quicksort}(A, \prec, s, e)$
    $s' \leftarrow s$, $e' \leftarrow e$, $M \leftarrow A[(s+e)/2]$ ▷ M: the pivot value. Selection may vary
    while $s' \le e'$ **do**
        while $A[s'] \prec M$ **do** $s' \leftarrow s' + 1$ **end while**
        while $M \prec A[e']$ **do** $e' \leftarrow e' - 1$ **end while**
        if $s' \le e'$ **then**
            $A[s'] \Leftrightarrow A[e']$
            $s' \leftarrow s' + 1$, $e' \leftarrow e' - 1$
        end if
    end while
    if $s \le e'$ **then** $\textsc{Quicksort}(A, \prec, s, e')$ **end if**
    if $s' \le e$ **then** $\textsc{Quicksort}(A, \prec, s', e)$ **end if**
end procedure

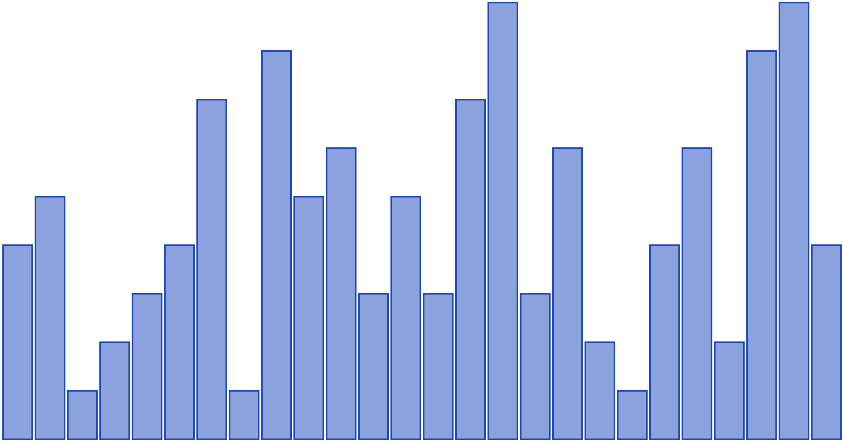
procedure $\textsc{Quicksort}(A, \prec, s, e)$
    $s' \leftarrow s,\ e' \leftarrow e,\ M \leftarrow A[(s+e)/2]$ $\triangleright$ M: the pivot value. Selection may vary
    while $s' \le e'$ **do**
        while $A[s'] \prec M$ **do** $s' \leftarrow s' + 1$ **end while** $\triangleright$ If $i \in [s; s')$ then $A[i] \preceq M$
        while $M \prec A[e']$ **do** $e' \leftarrow e' - 1$ **end while**
        if $s' \le e'$ **then**
            $A[s'] \Leftrightarrow A[e']$
            $s' \leftarrow s' + 1,\ e' \leftarrow e' - 1$
        end if
    end while
    if $s \le e'$ **then** $\textsc{Quicksort}(A, \prec, s, e')$ **end if**
    if $s' \le e$ **then** $\textsc{Quicksort}(A, \prec, s', e)$ **end if**
end procedure

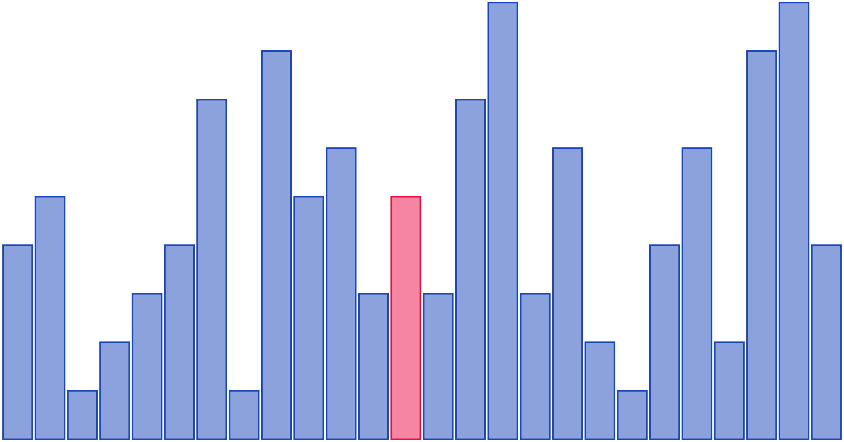
procedure $\textsc{Quicksort}(A, \prec, s, e)$
    $s' \leftarrow s,\ e' \leftarrow e,\ M \leftarrow A[(s+e)/2]$ ▷ M: the pivot value. Selection may vary
    while $s' \le e'$ **do**
        while $A[s'] \prec M$ **do** $s' \leftarrow s' + 1$ **end while** ▷ If $i \in [s; s')$ then $A[i] \preceq M$
        while $M \prec A[e']$ **do** $e' \leftarrow e' - 1$ **end while** ▷ If $i \in (e'; e]$ then $M \preceq A[i]$
        if $s' \le e'$ **then**
            $A[s'] \Leftrightarrow A[e']$
            $s' \leftarrow s' + 1,\ e' \leftarrow e' - 1$
        end if
    end while
    if $s \le e'$ **then** $\textsc{Quicksort}(A, \prec, s, e')$ **end if**
    if $s' \le e$ **then** $\textsc{Quicksort}(A, \prec, s', e)$ **end if**
end procedure

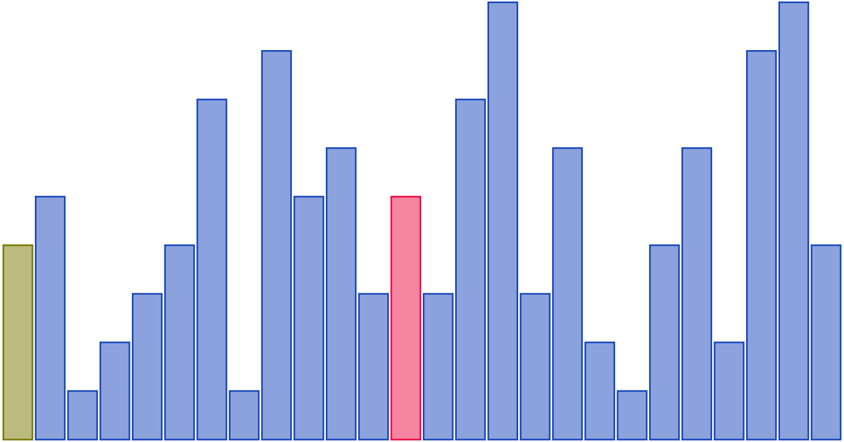
procedure $\textsc{Quicksort}(A, \prec, s, e)$
    $s' \leftarrow s,\ e' \leftarrow e,\ M \leftarrow A[(s+e)/2]$ ▷ M: the pivot value. Selection may vary
    while $s' \le e'$ **do**
        while $A[s'] \prec M$ **do** $s' \leftarrow s' + 1$ **end while** ▷ If $i \in [s; s')$ then $A[i] \preceq M$
        while $M \prec A[e']$ **do** $e' \leftarrow e' - 1$ **end while** ▷ If $i \in (e'; e]$ then $M \preceq A[i]$
        if $s' \le e'$ **then** ▷ If the array is not yet split completely...
            $A[s'] \Leftrightarrow A[e']$ ▷ swap the elements and continue splitting
            $s' \leftarrow s' + 1,\ e' \leftarrow e' - 1$
        end if
    end while
    if $s \le e'$ **then** $\textsc{Quicksort}(A, \prec, s, e')$ **end if**
    if $s' \le e$ **then** $\textsc{Quicksort}(A, \prec, s', e)$ **end if**
end procedure

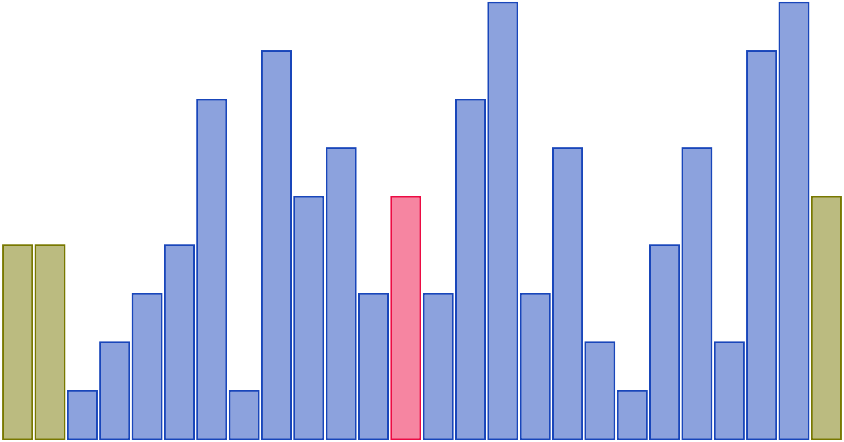
procedure $\textsc{Quicksort}(A, \prec, s, e)$
    $s' \leftarrow s,\ e' \leftarrow e,\ M \leftarrow A[(s+e)/2]$ ▷ M: the pivot value. Selection may vary
    while $s' \le e'$ **do**
        while $A[s'] \prec M$ **do** $s' \leftarrow s'+1$ **end while** ▷ If $i \in [s; s')$ then $A[i] \preceq M$
        while $M \prec A[e']$ **do** $e' \leftarrow e'-1$ **end while** ▷ If $i \in (e'; e]$ then $M \preceq A[i]$
        if $s' \le e'$ **then** ▷ If the array is not yet split completely...
            $A[s'] \Leftrightarrow A[e']$ ▷ swap the elements and continue splitting
            $s' \leftarrow s'+1,\ e' \leftarrow e'-1$
        end if
    end while
    if $s \le e'$ **then** $\textsc{Quicksort}(A, \prec, s, e')$ **end if** ▷ If $i \in (e'; s')$, $A[i] = M$
    if $s' \le e$ **then** $\textsc{Quicksort}(A, \prec, s', e)$ **end if** ▷ and is in the right place
end procedure

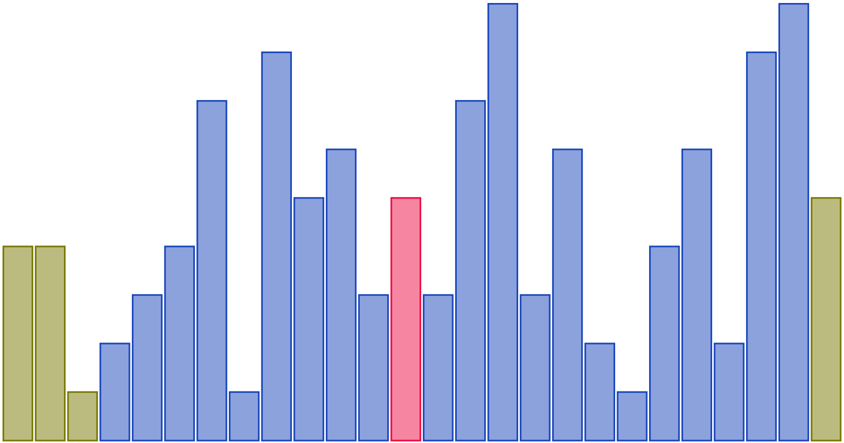


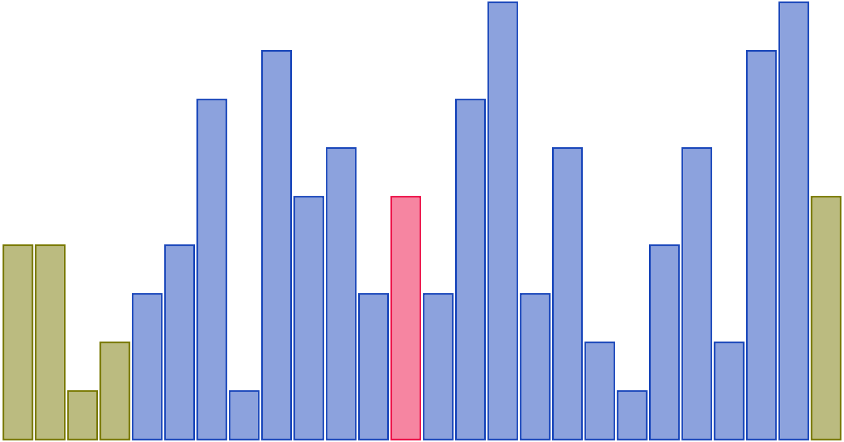


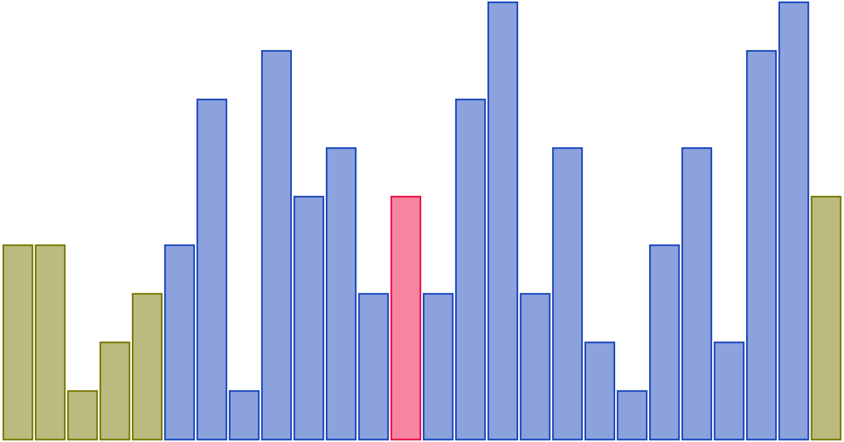


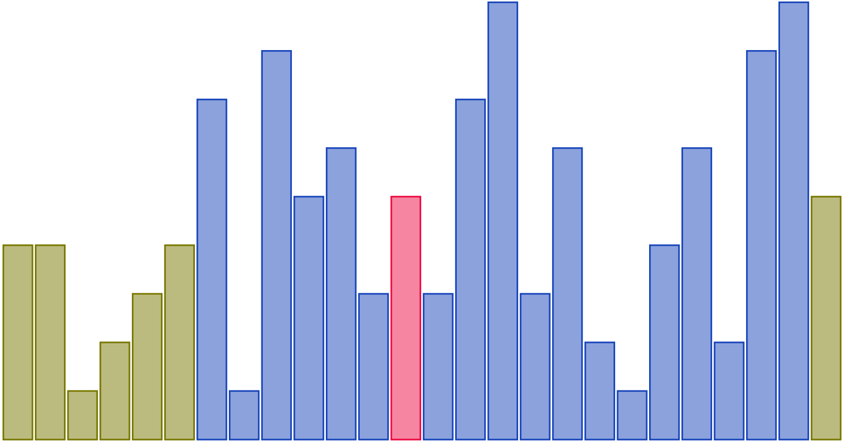


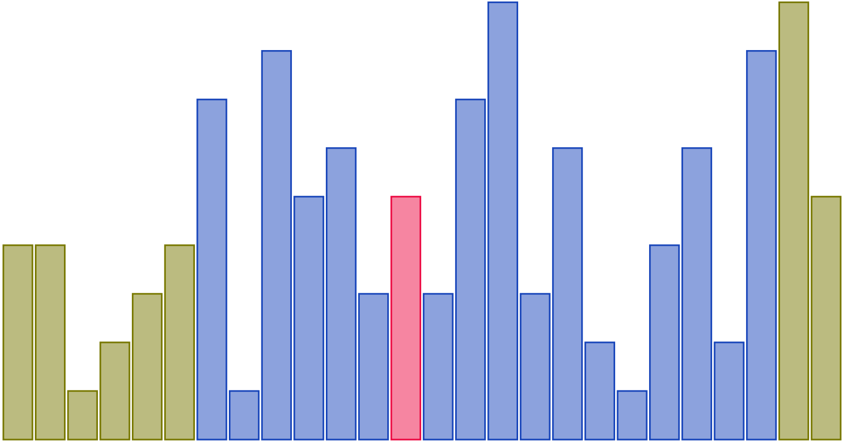


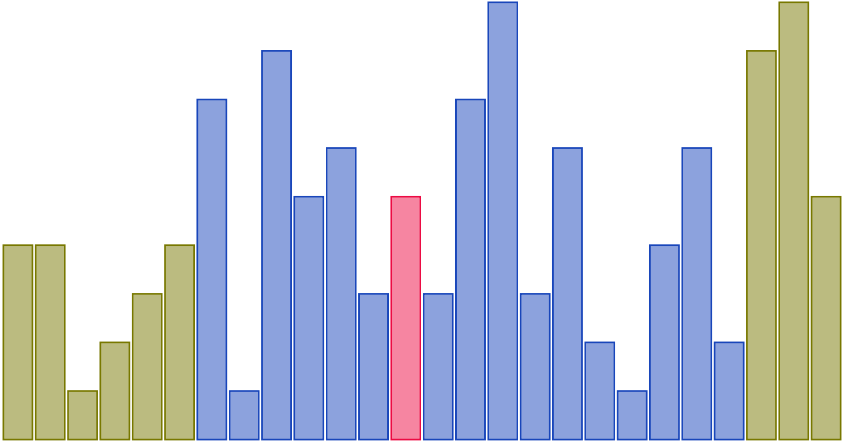


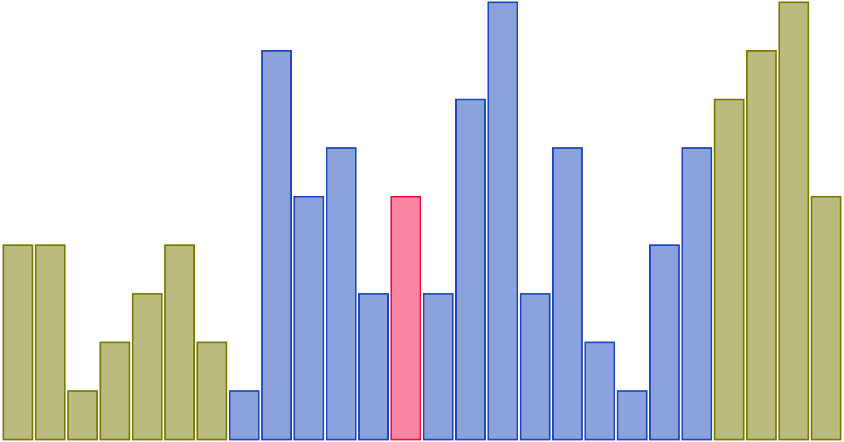


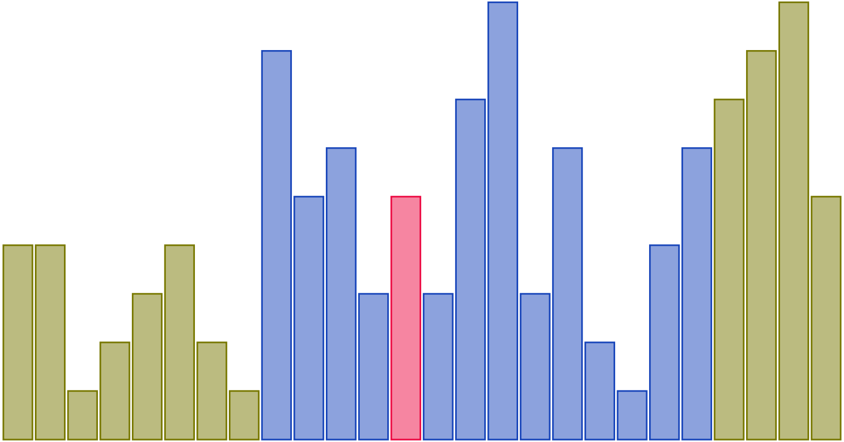


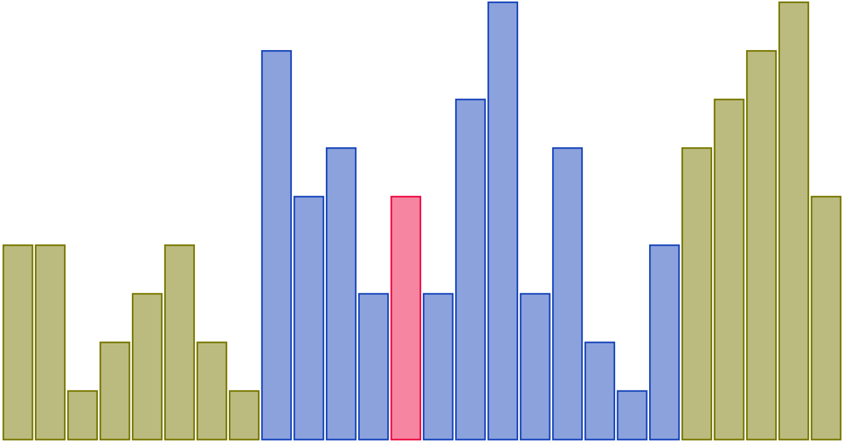


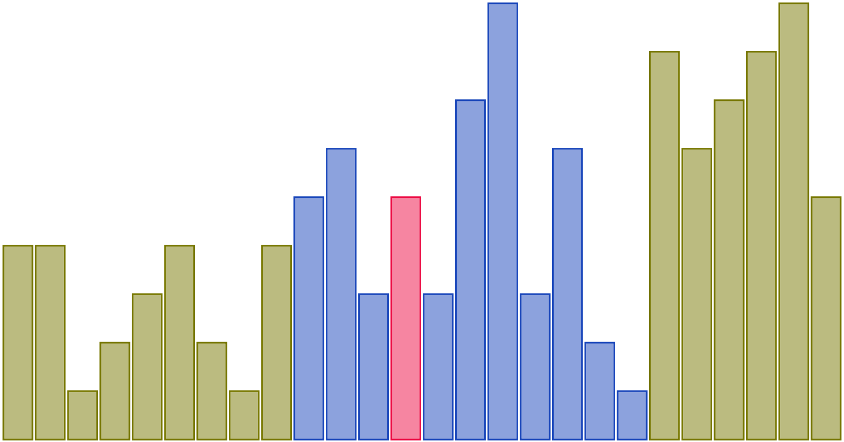


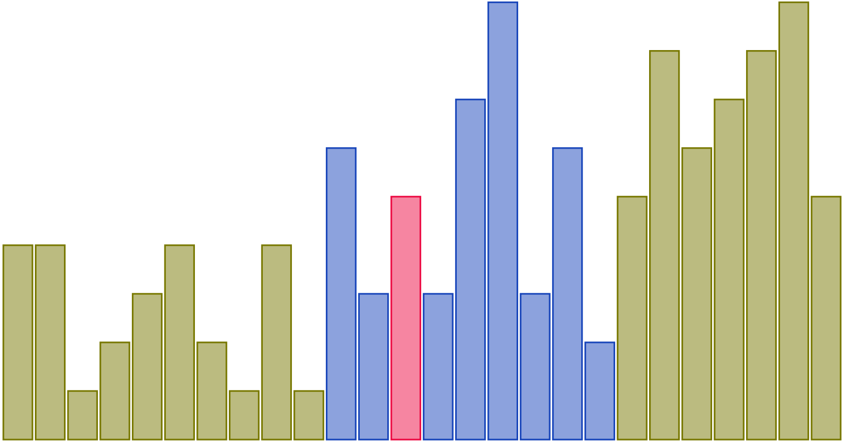


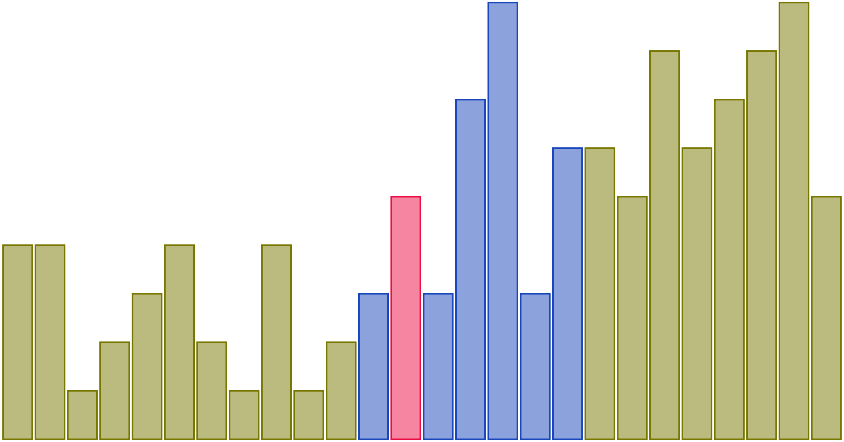


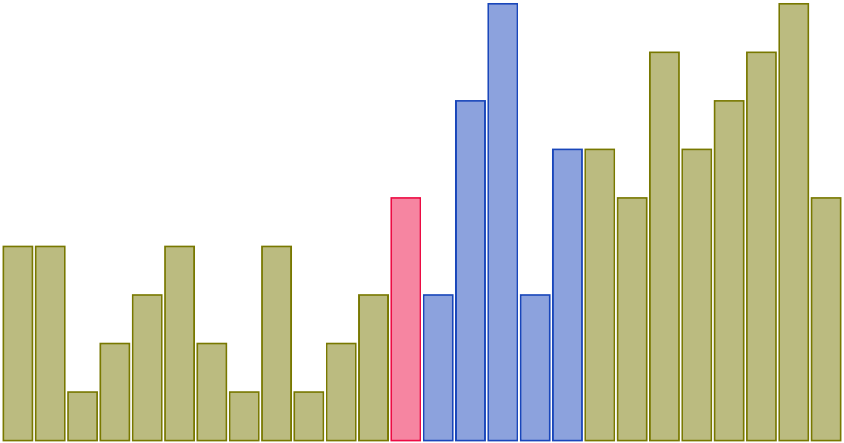


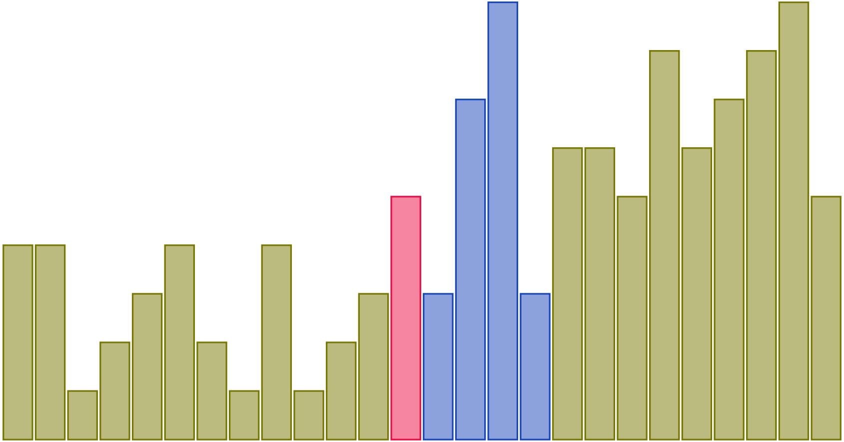


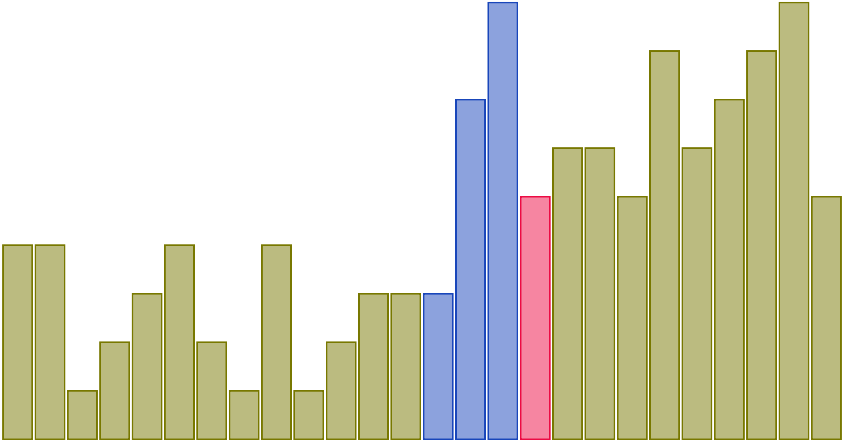


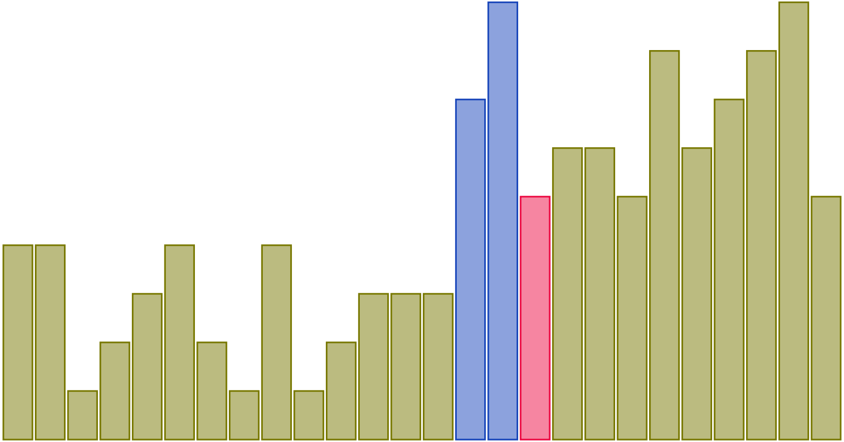


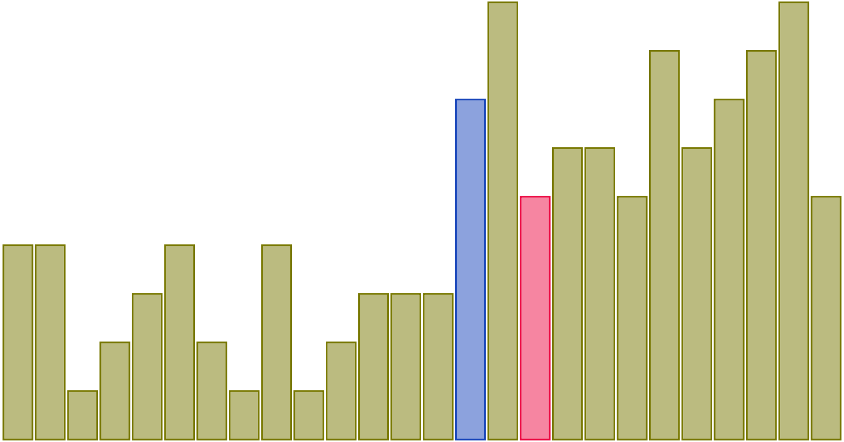




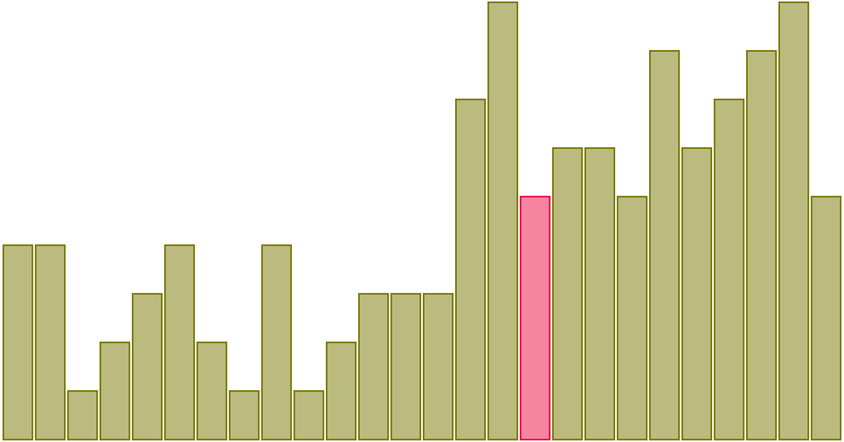


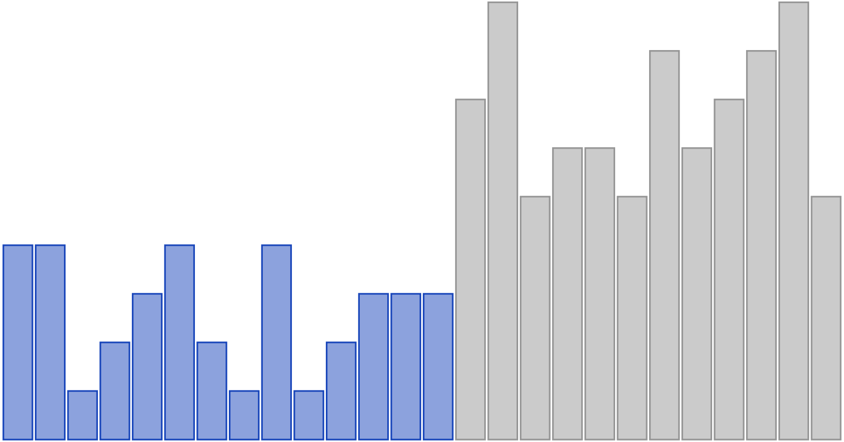


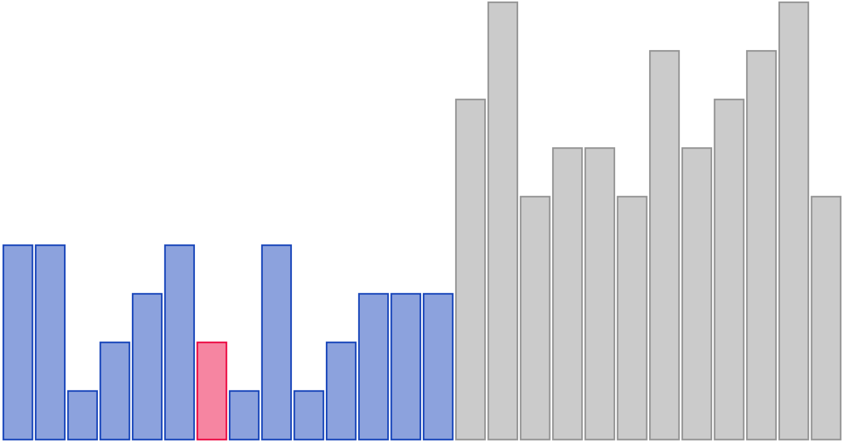


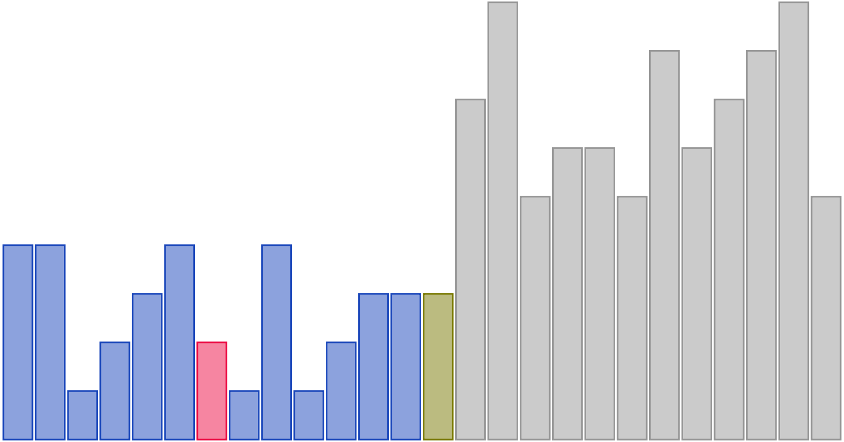


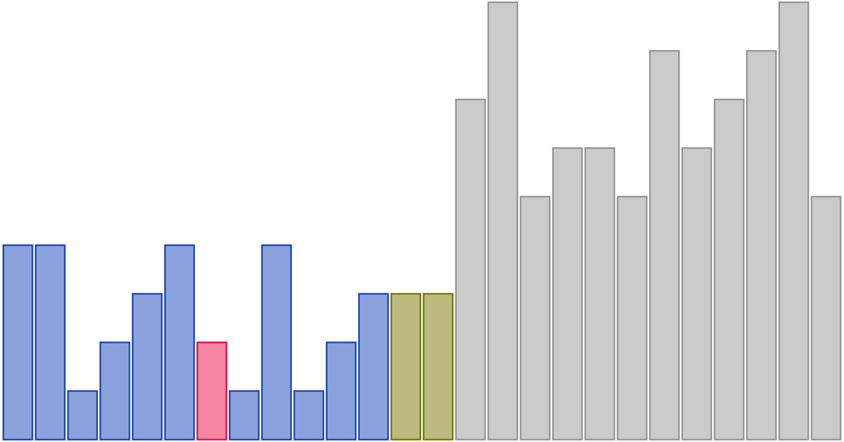
Quicksort: Example

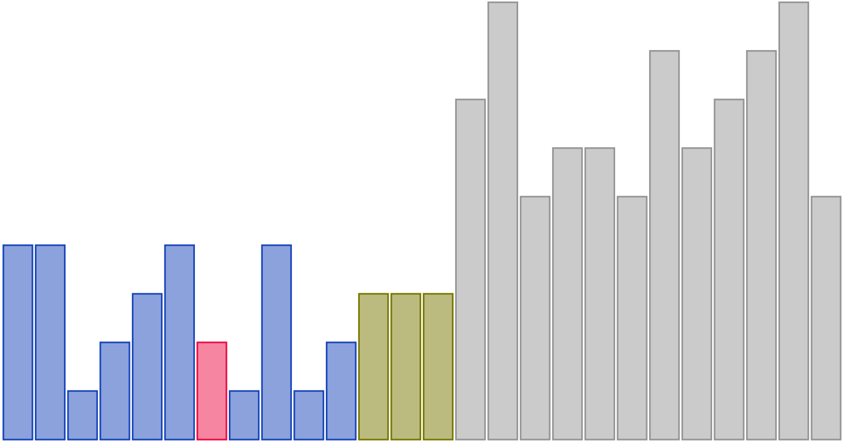


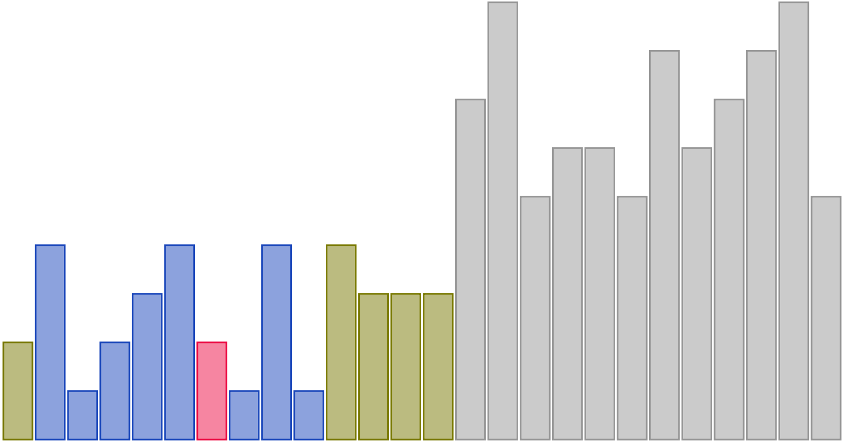


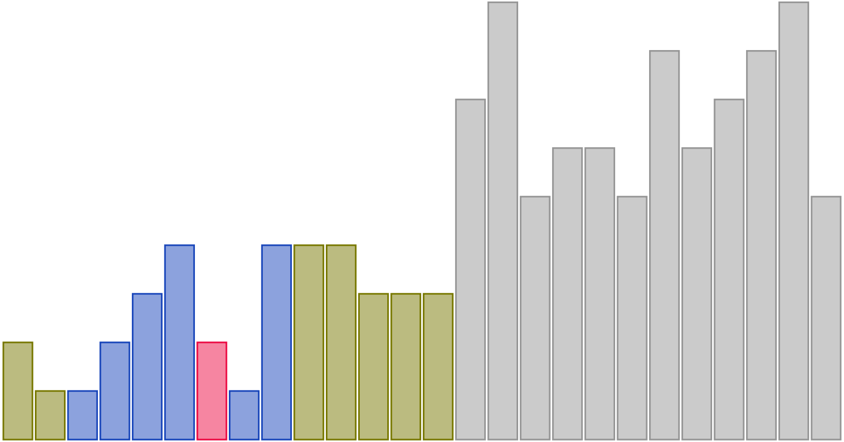


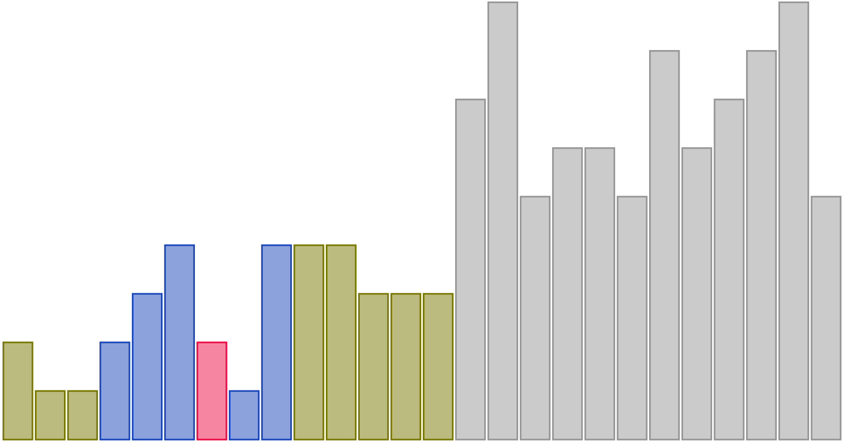


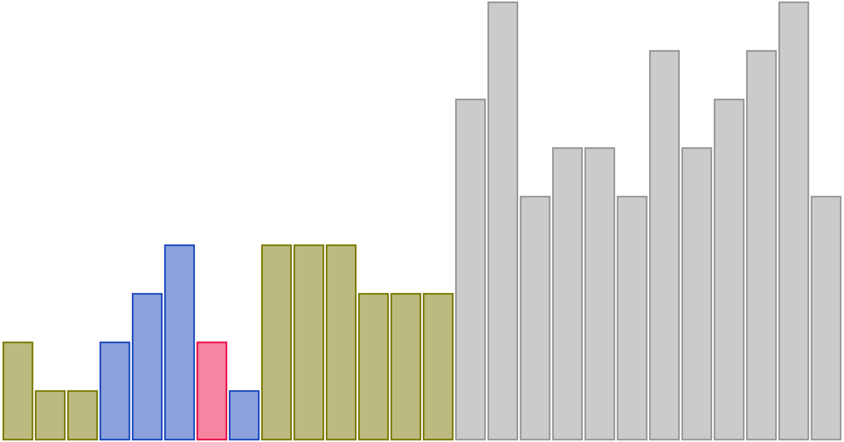


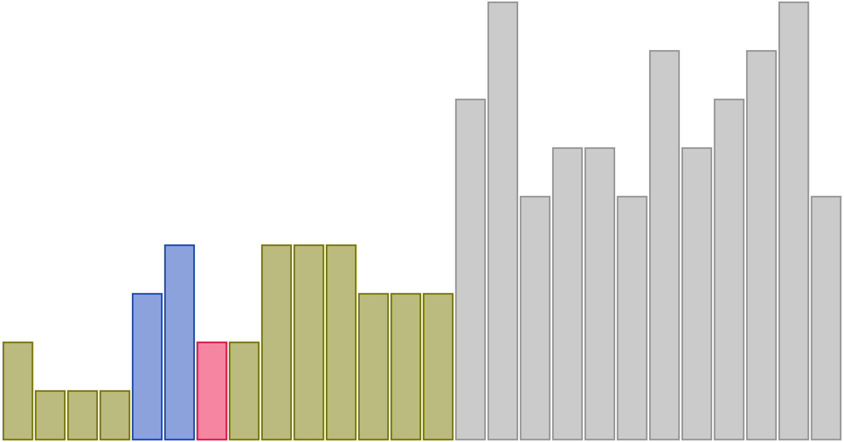


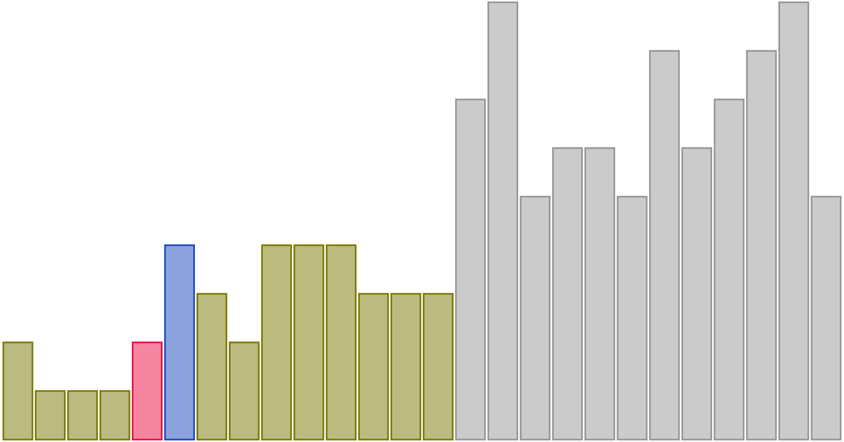


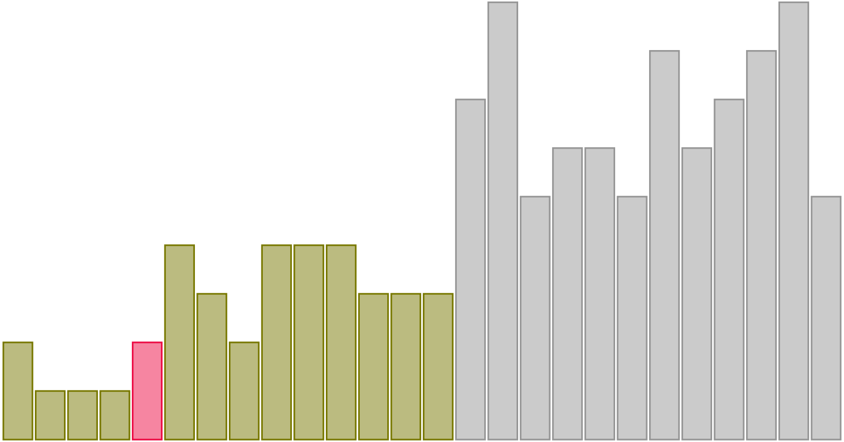


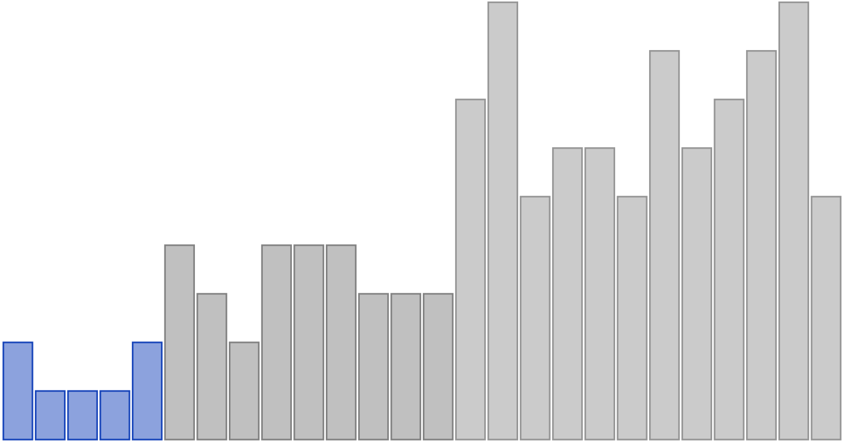


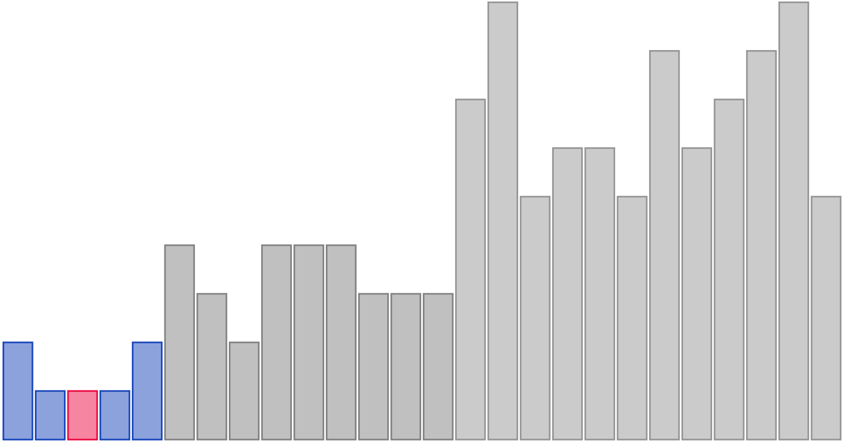


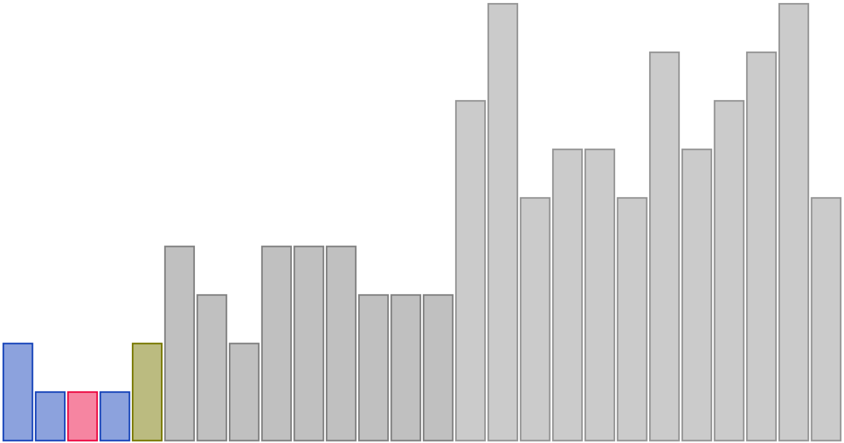


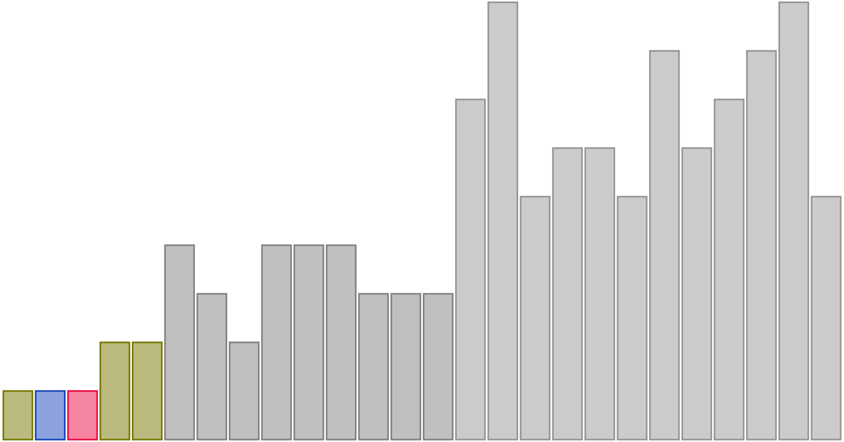


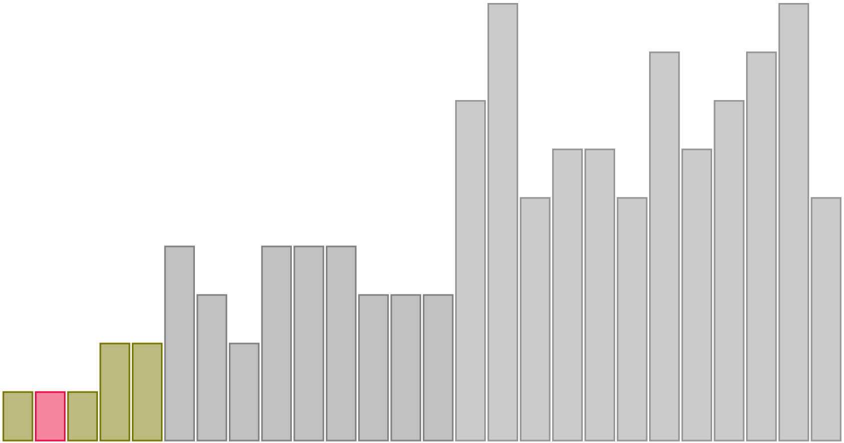


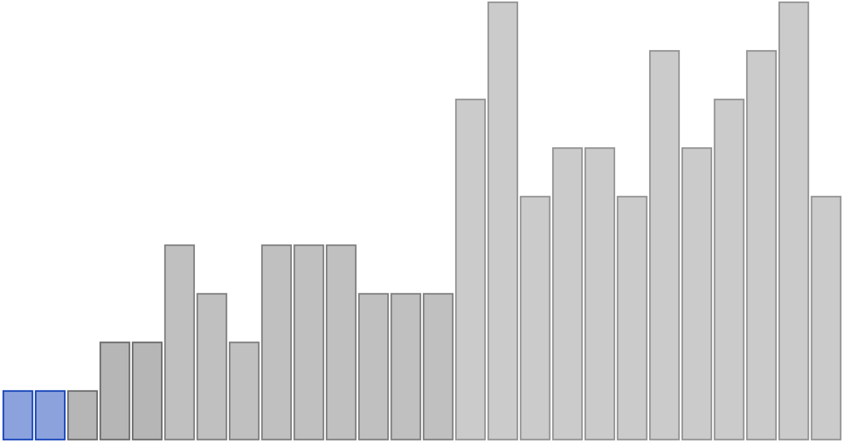


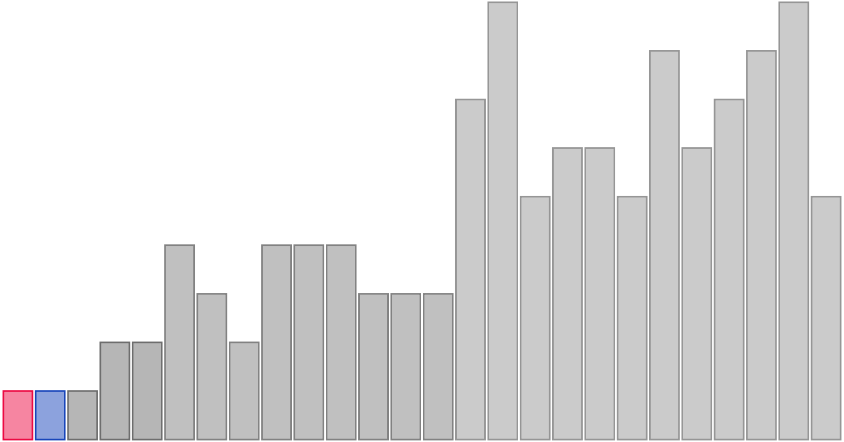


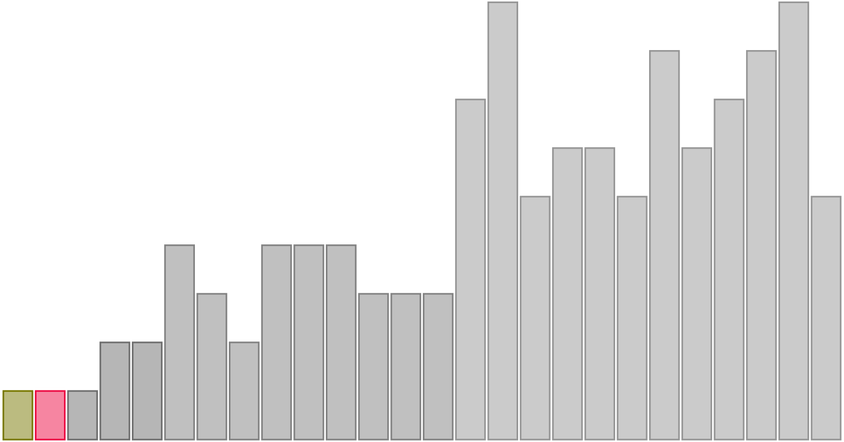


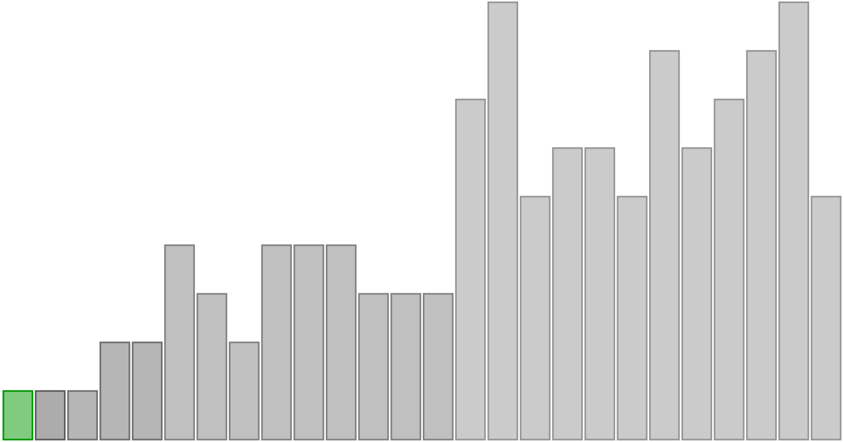


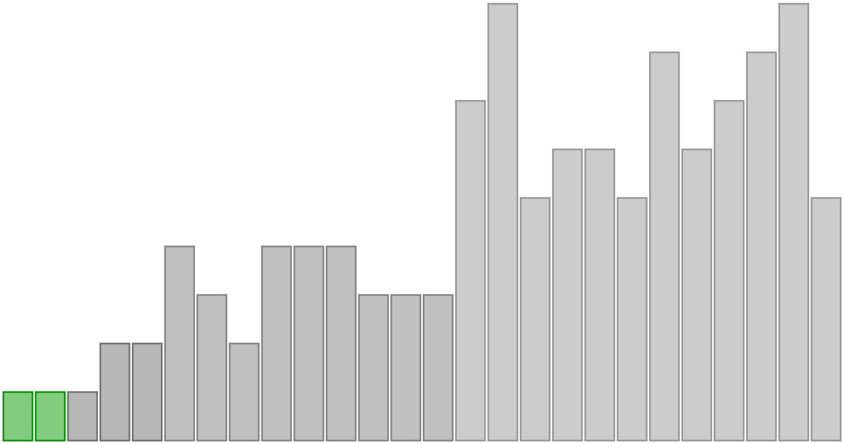


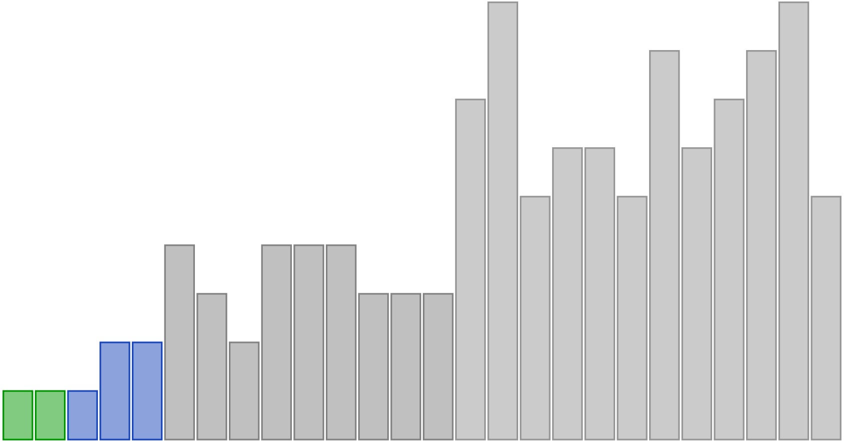


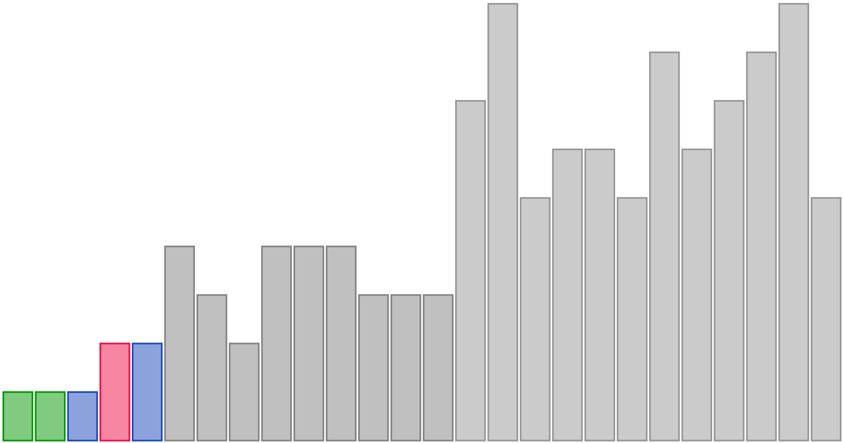


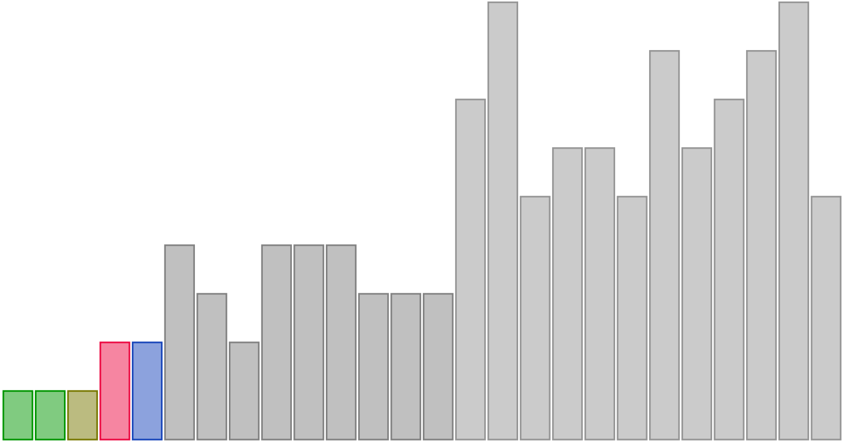


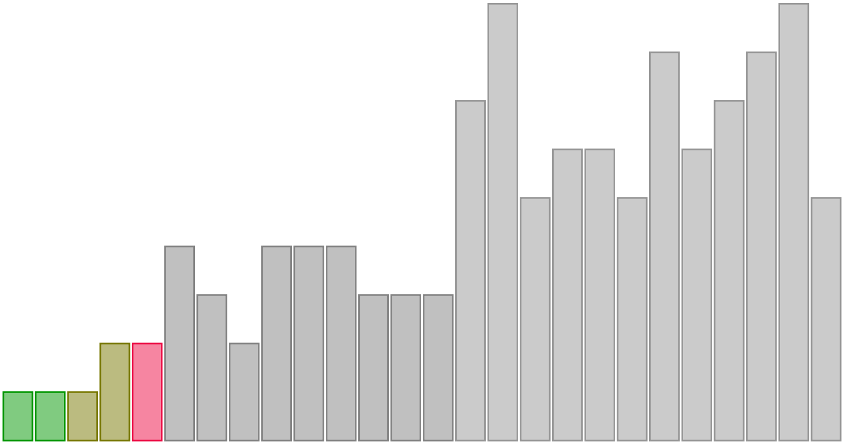


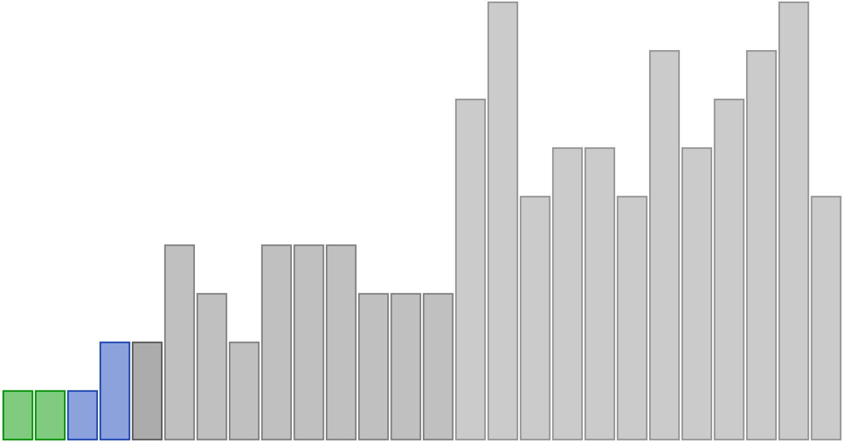


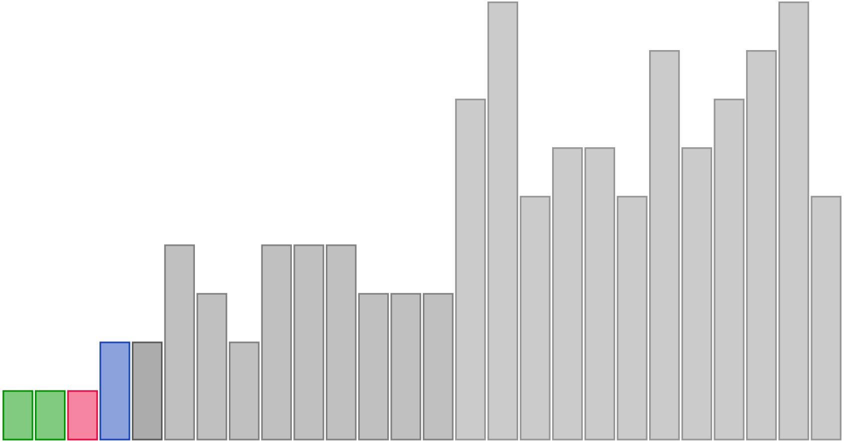


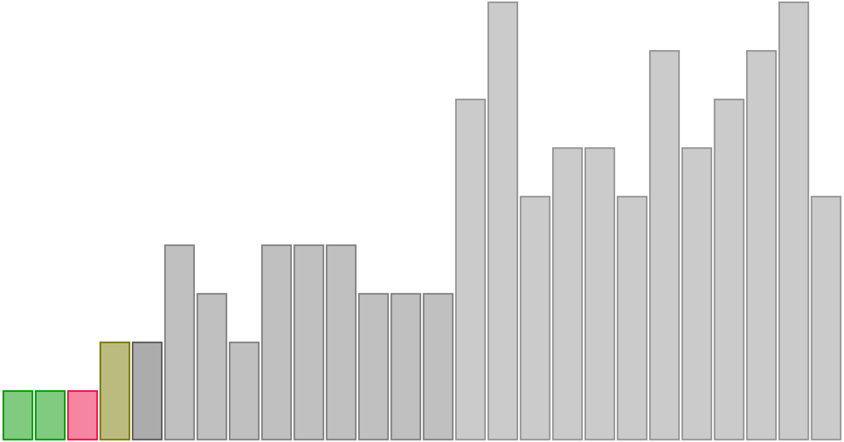


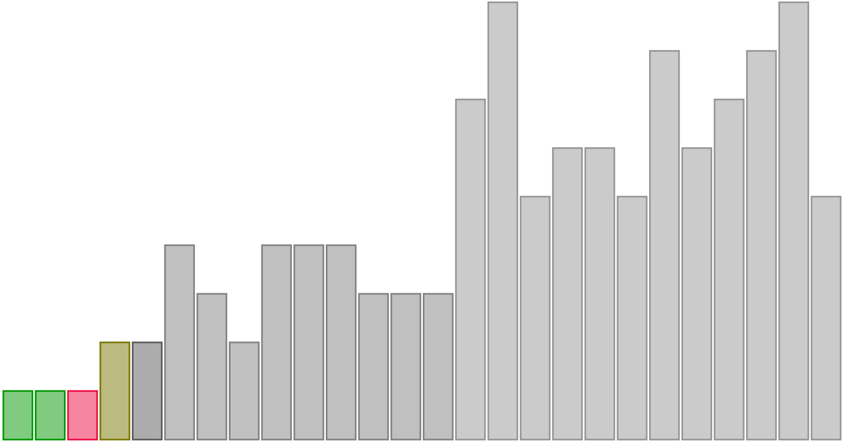


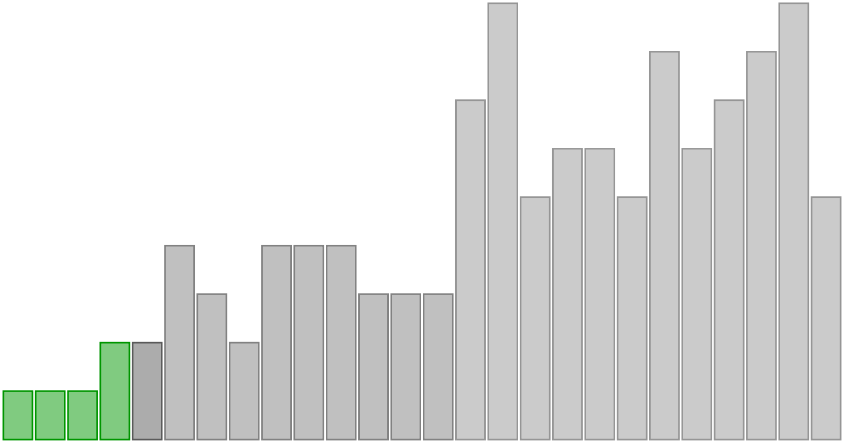


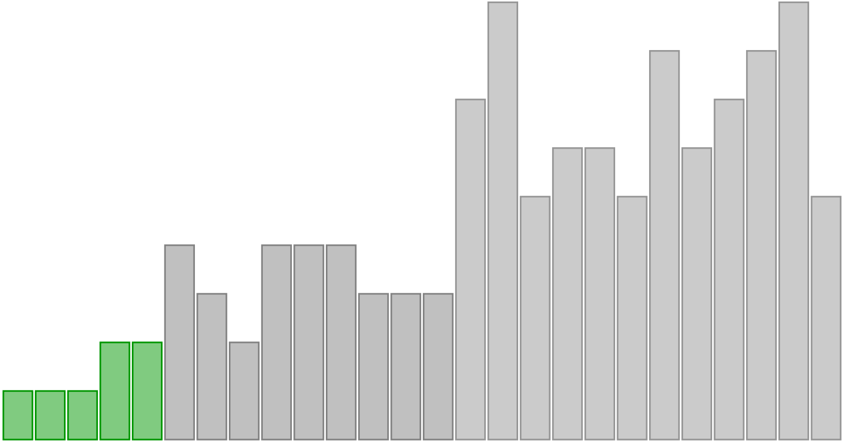


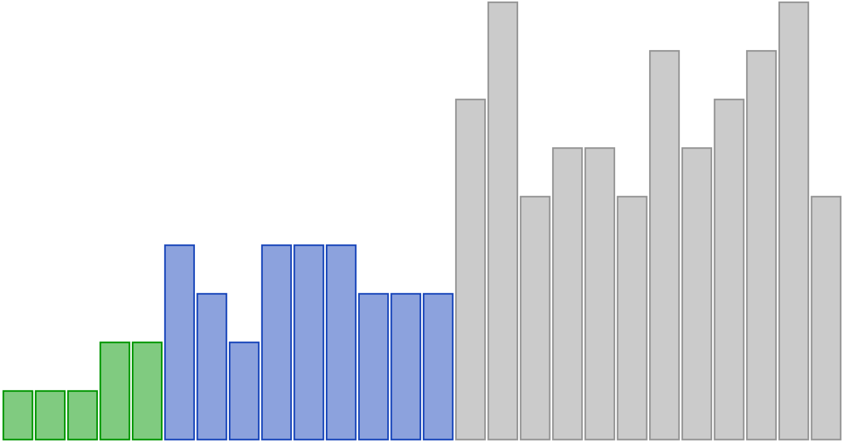


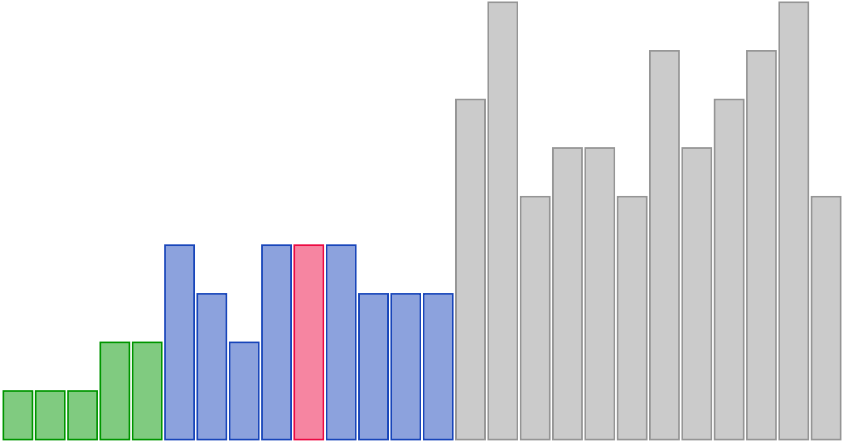


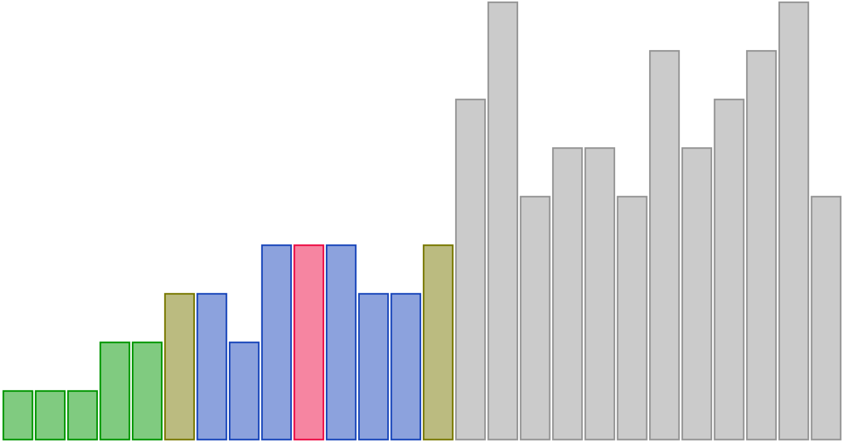


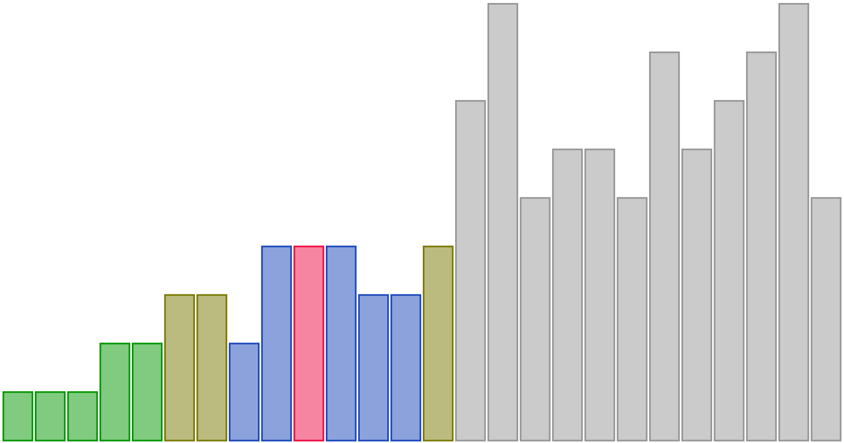


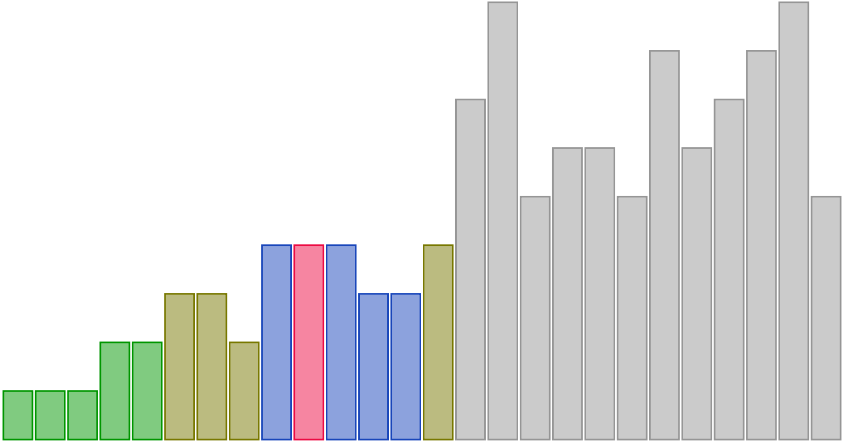


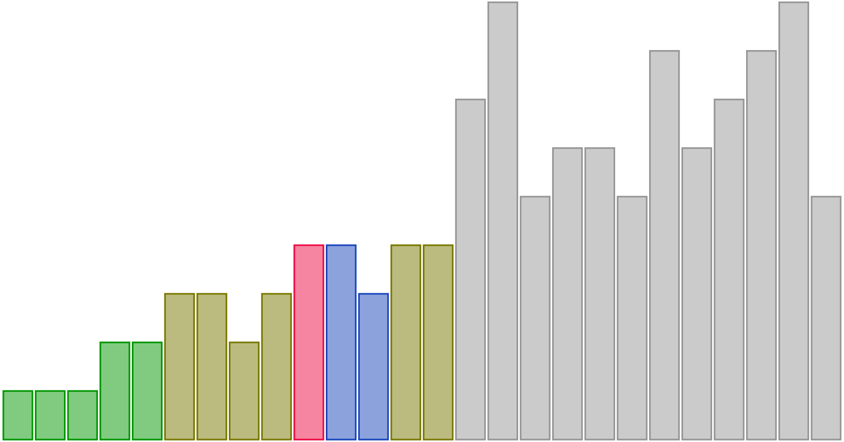


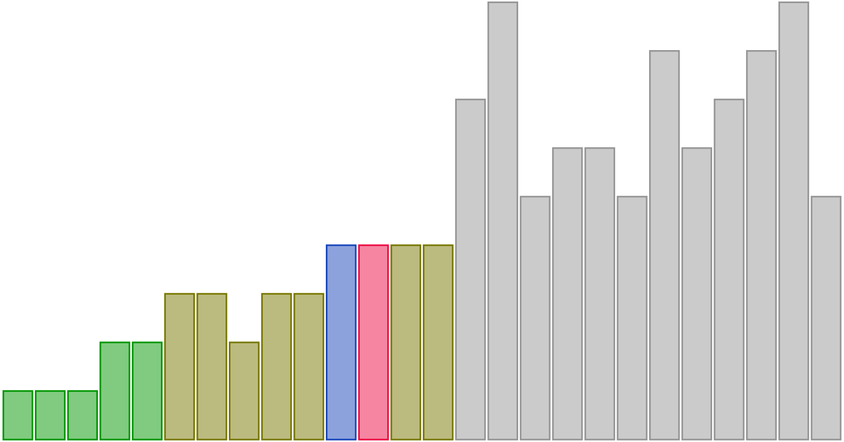


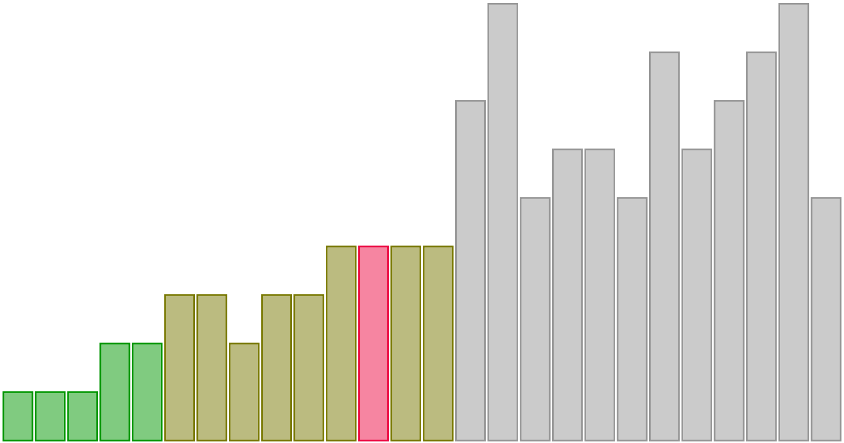


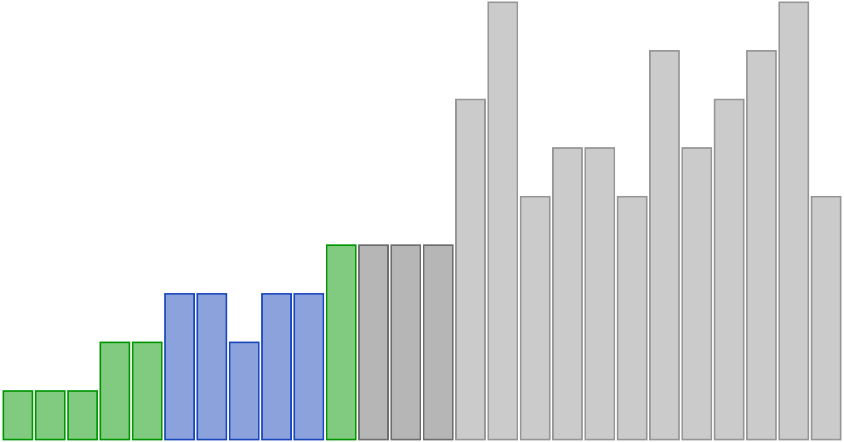


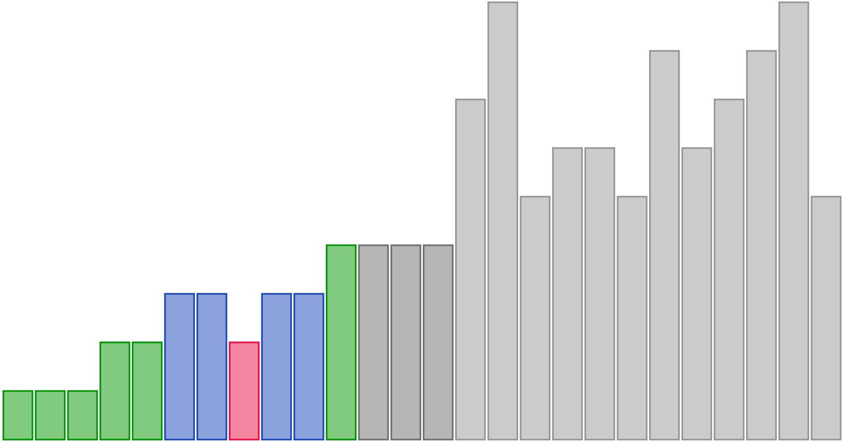


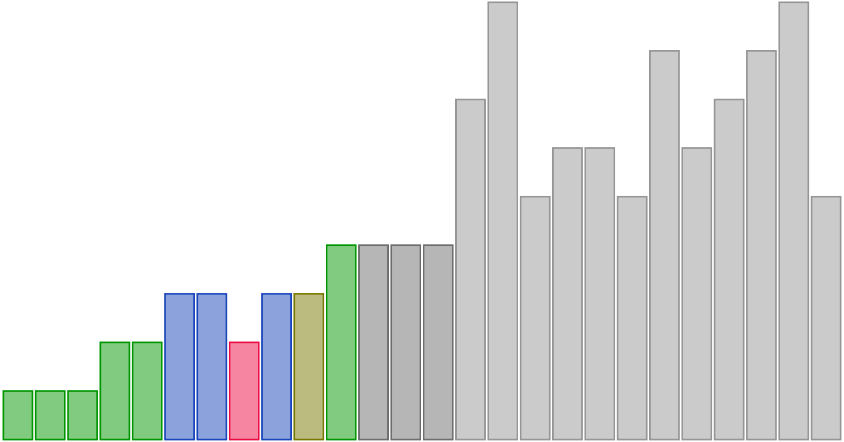


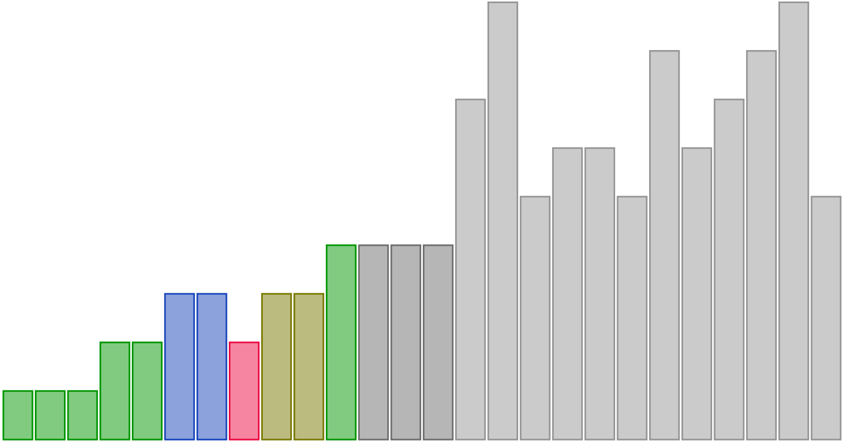


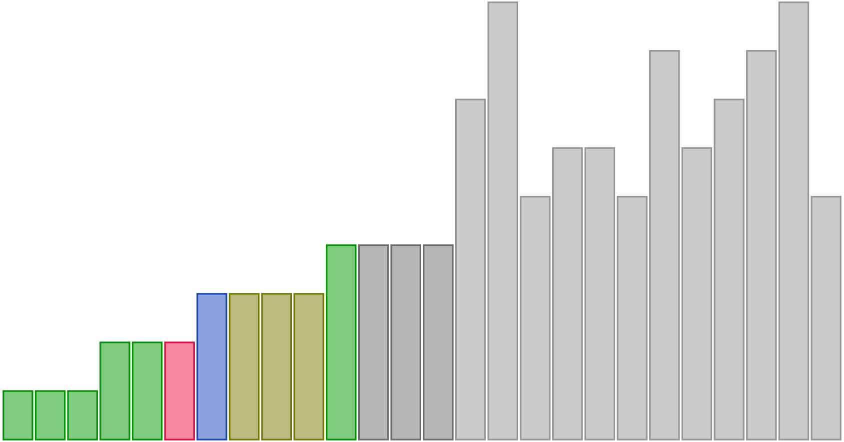


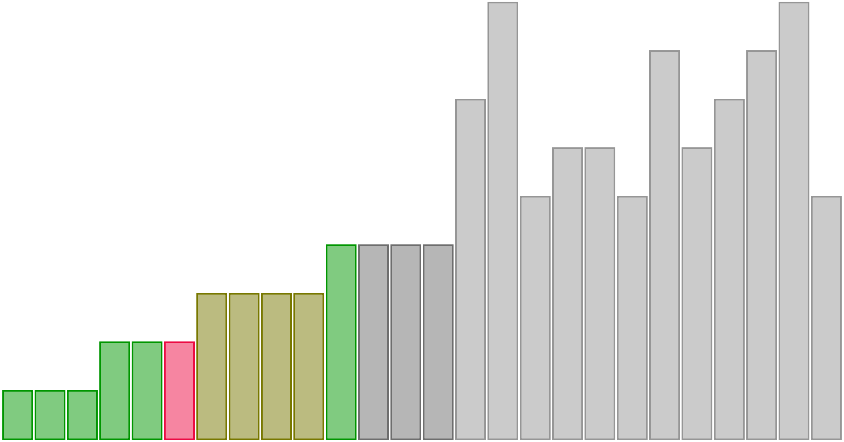


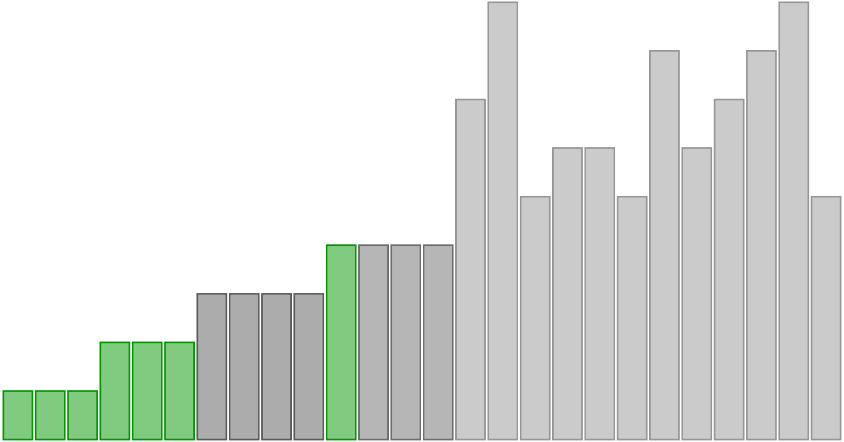


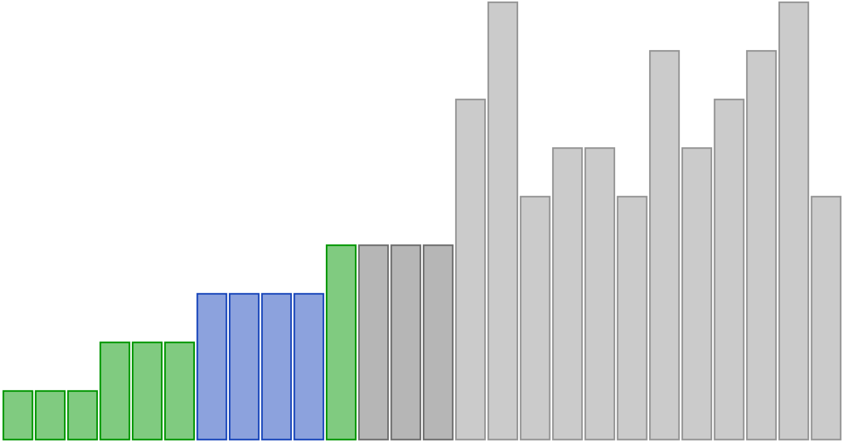


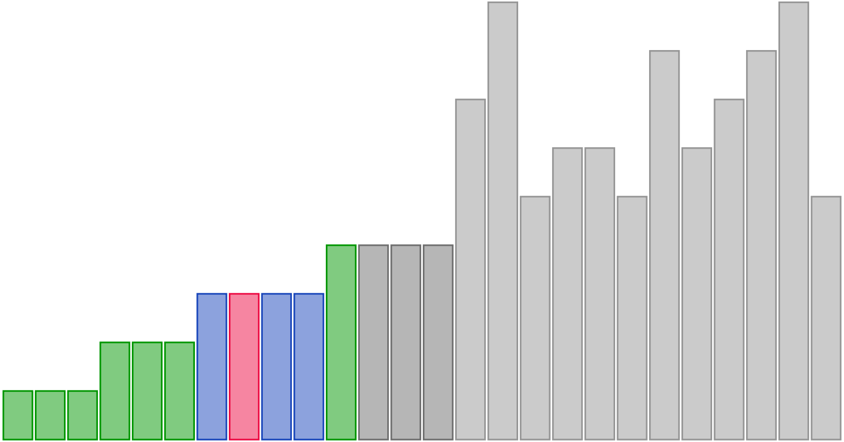


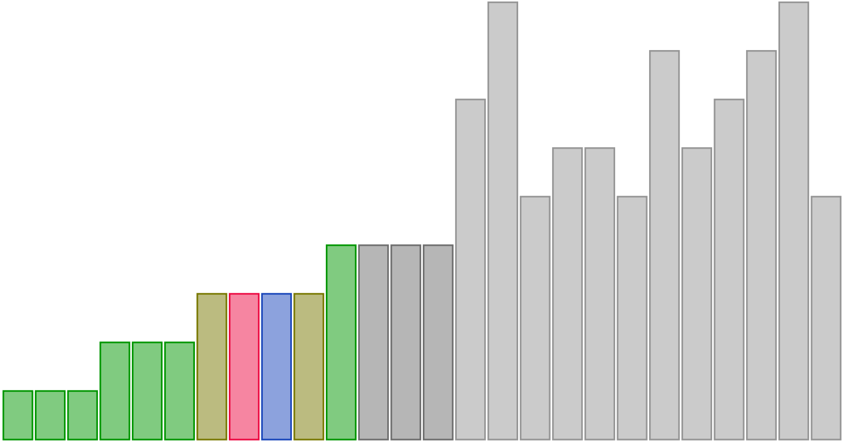


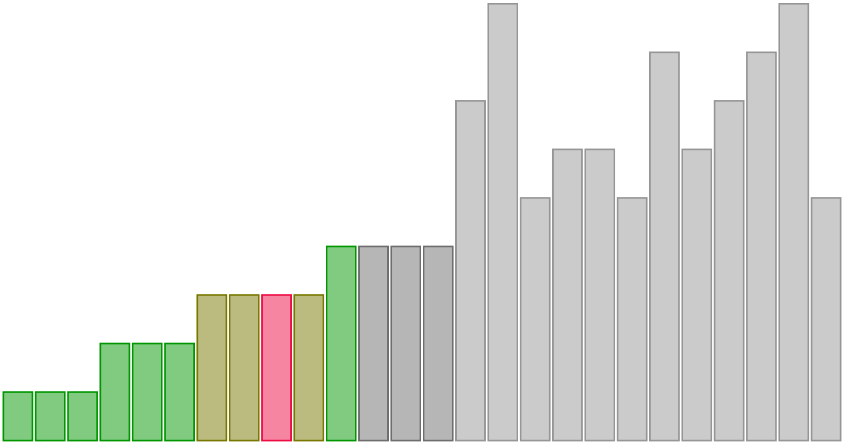


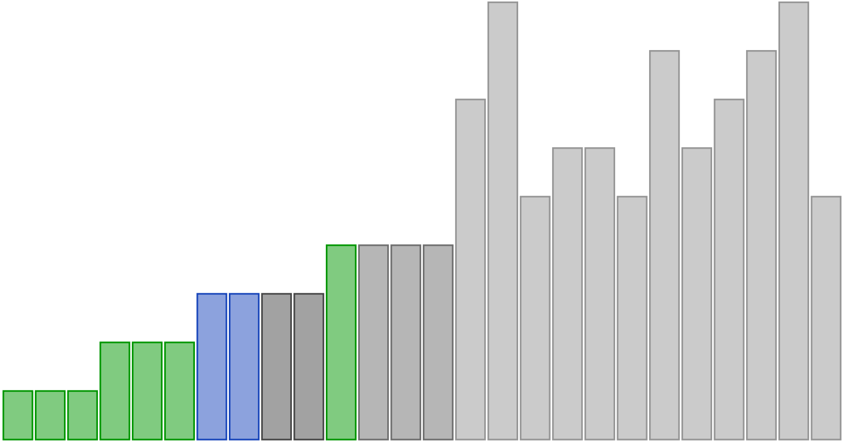


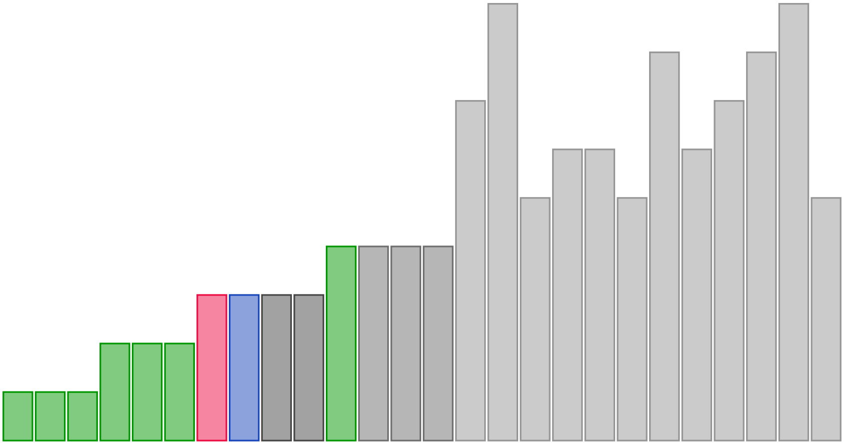


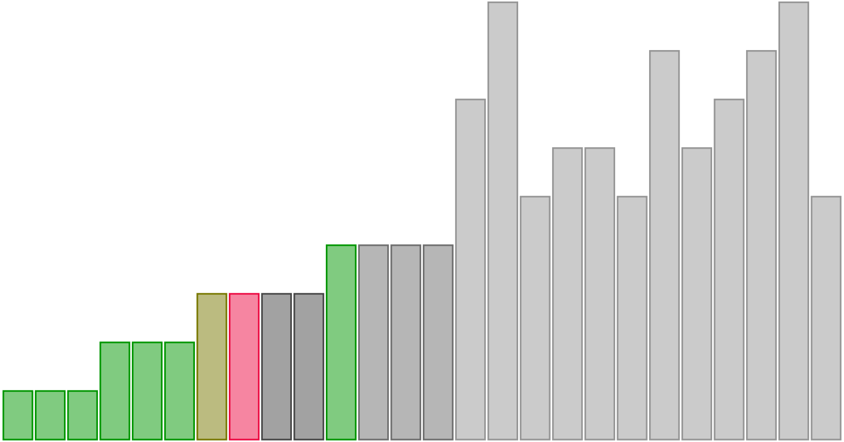


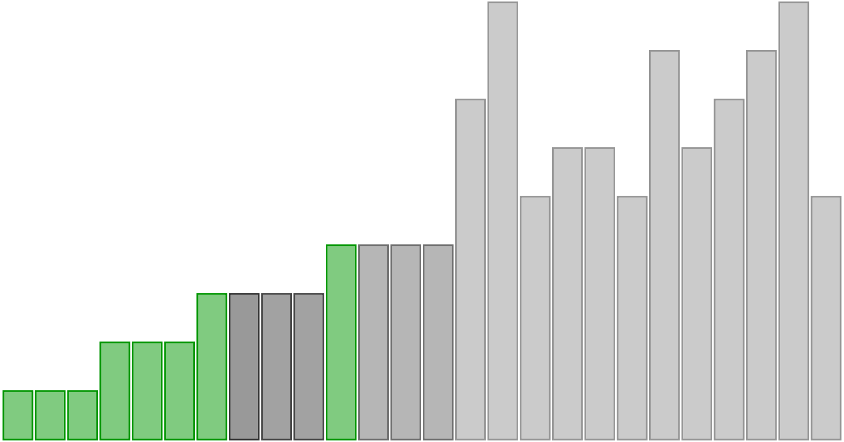


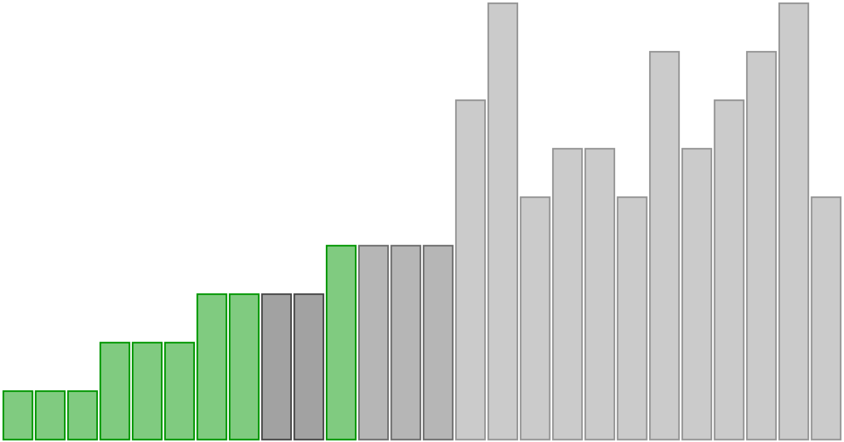


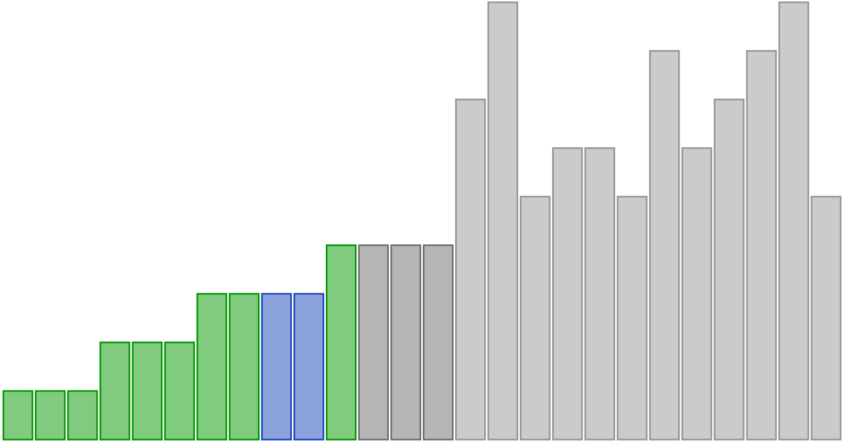


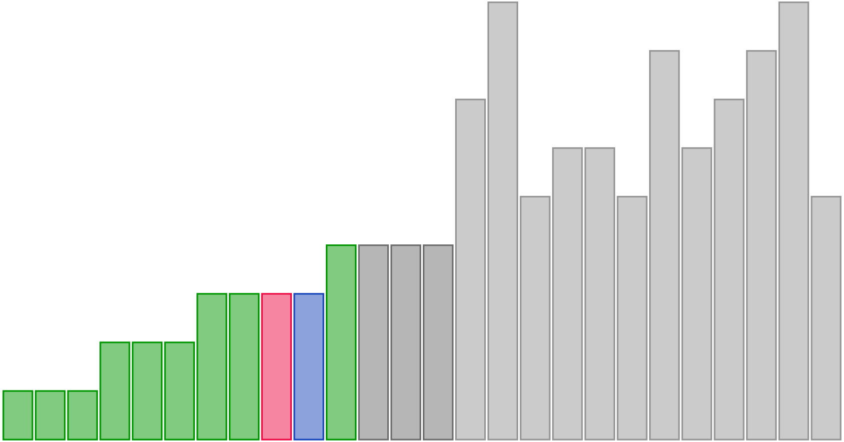


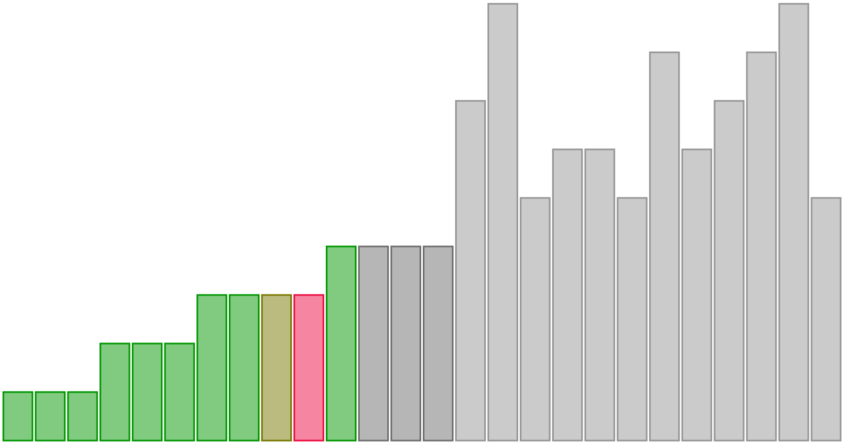


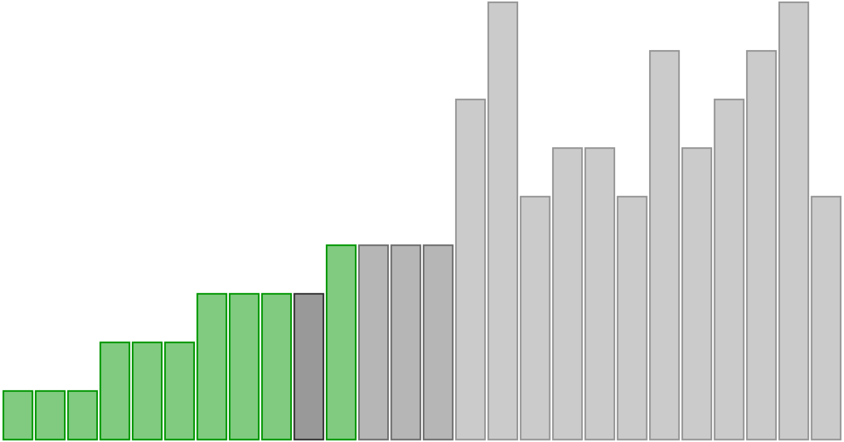


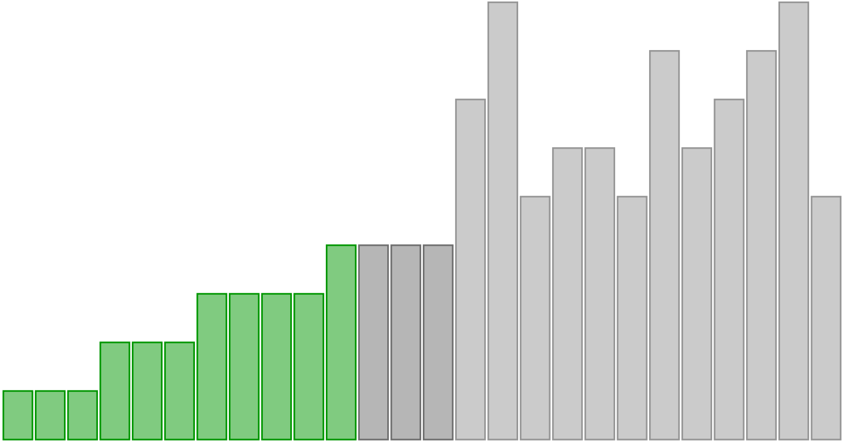


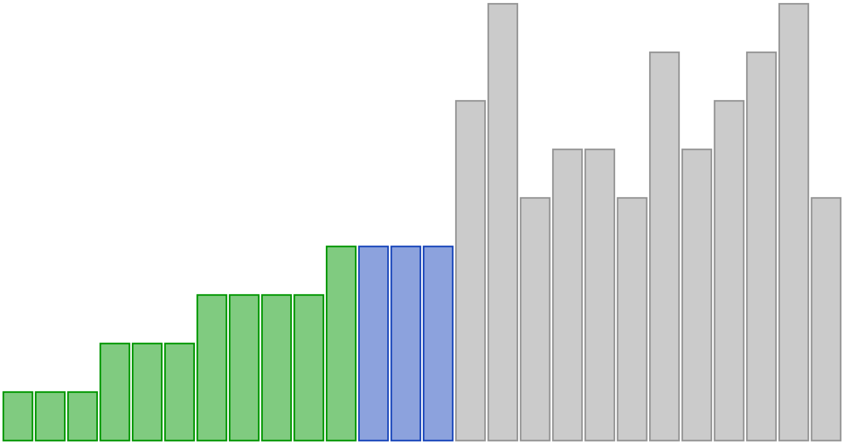


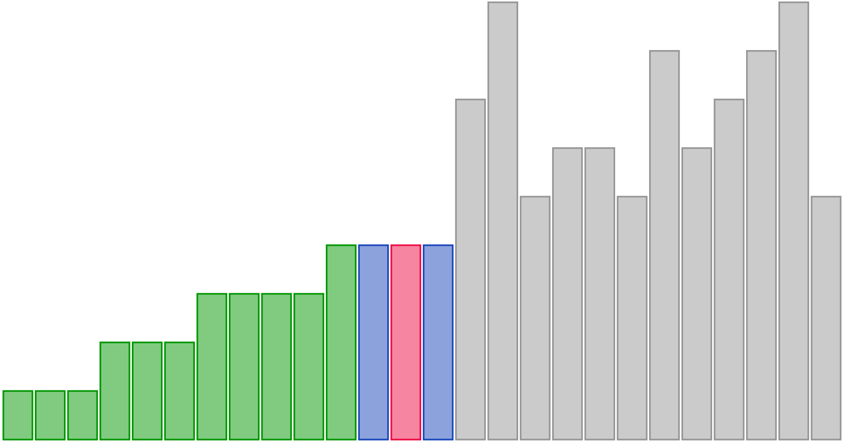


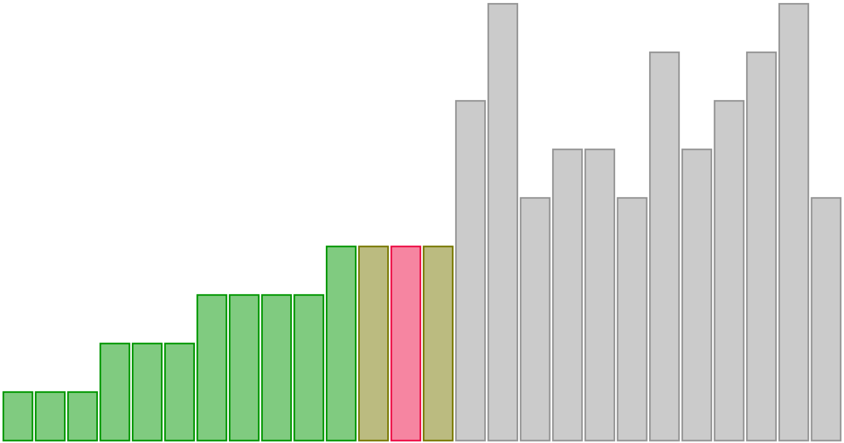


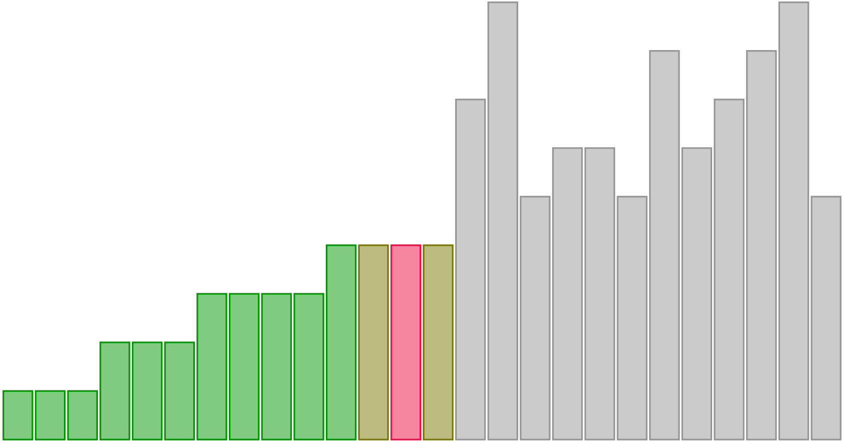


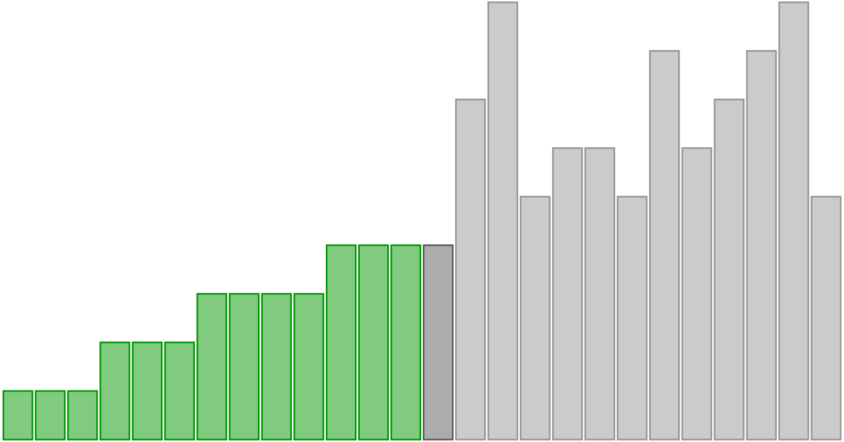


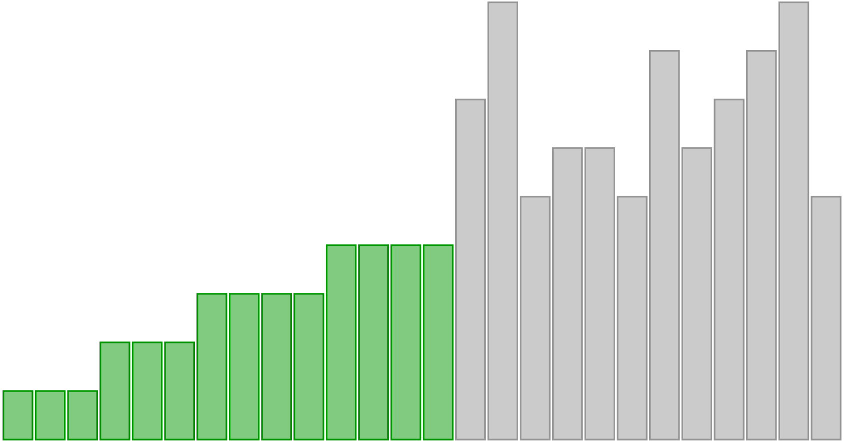


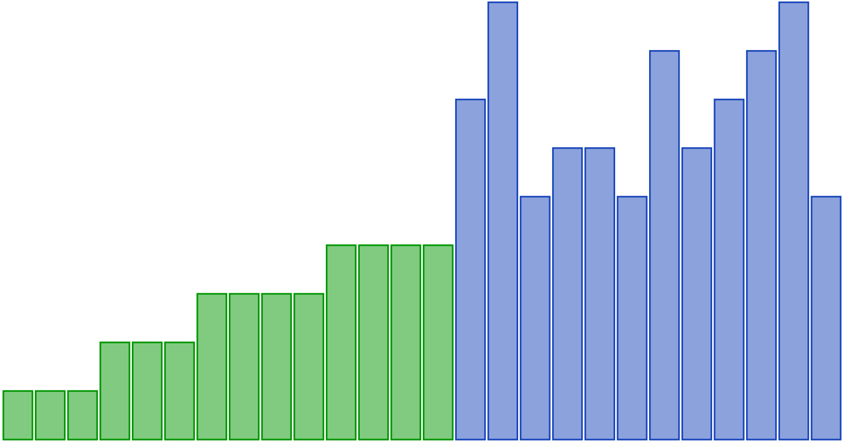


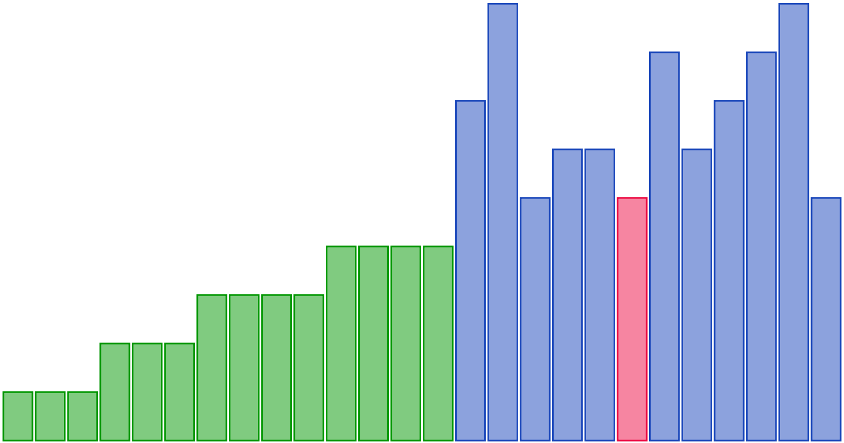


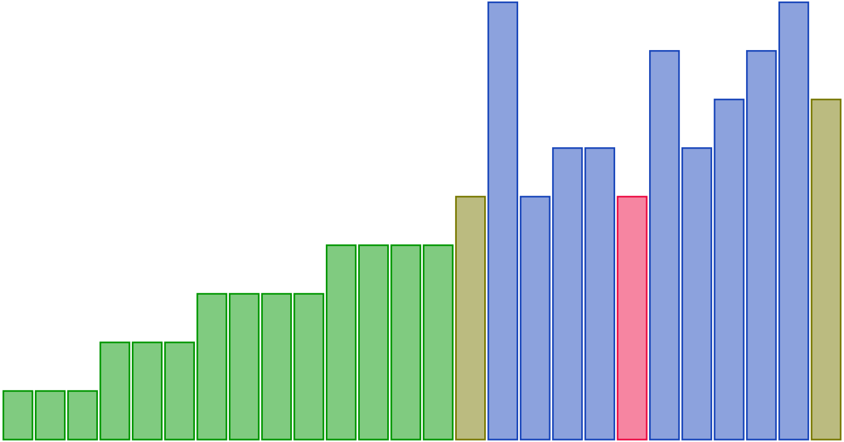


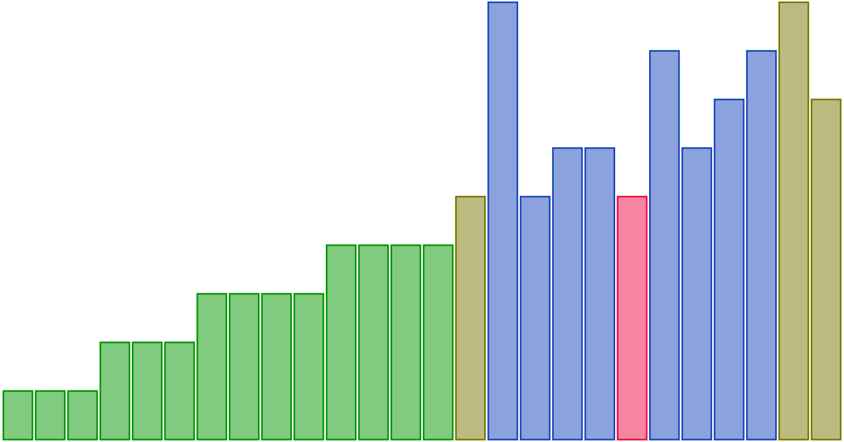


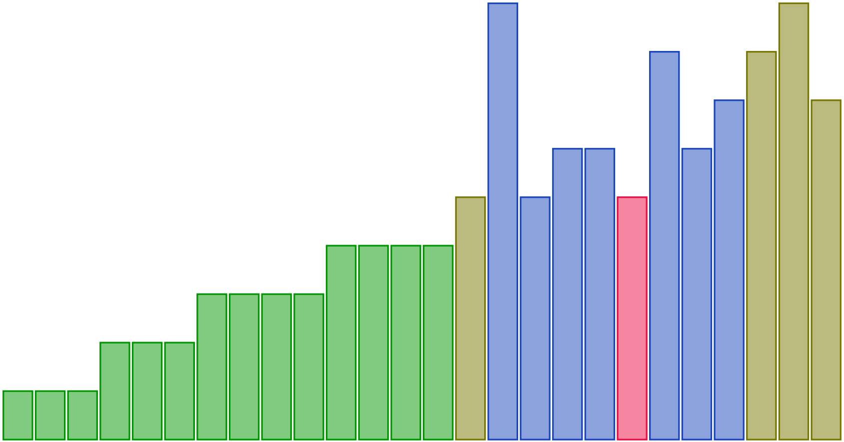


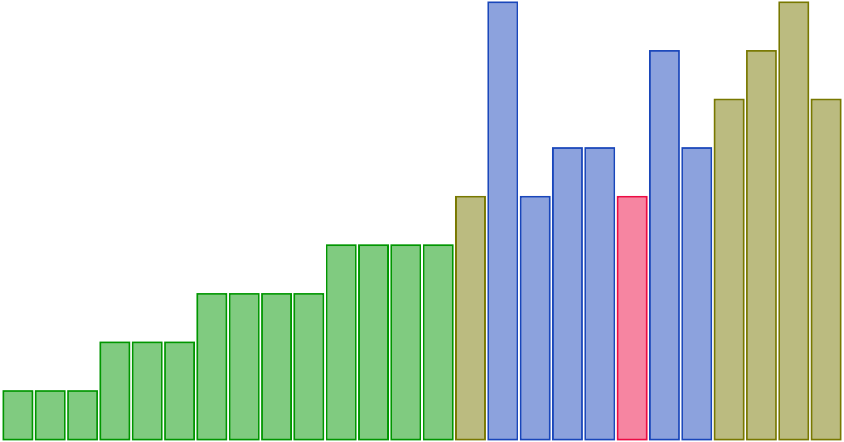


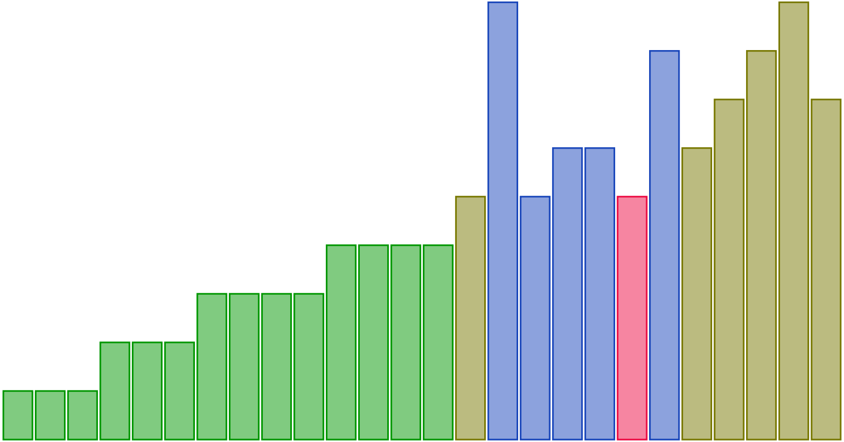


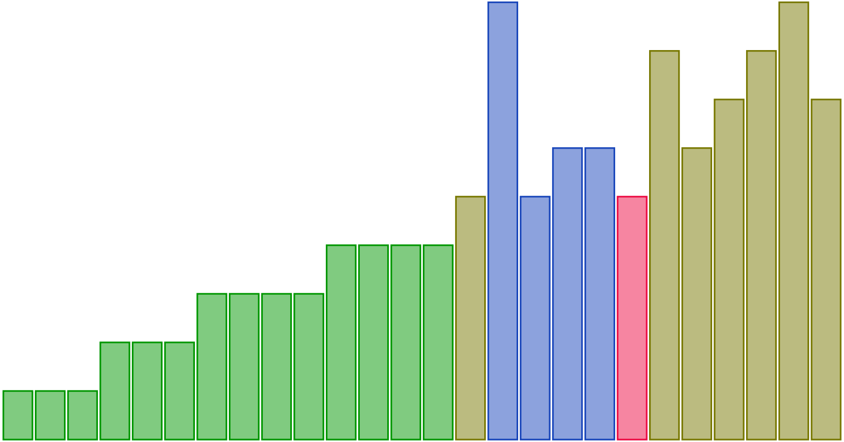


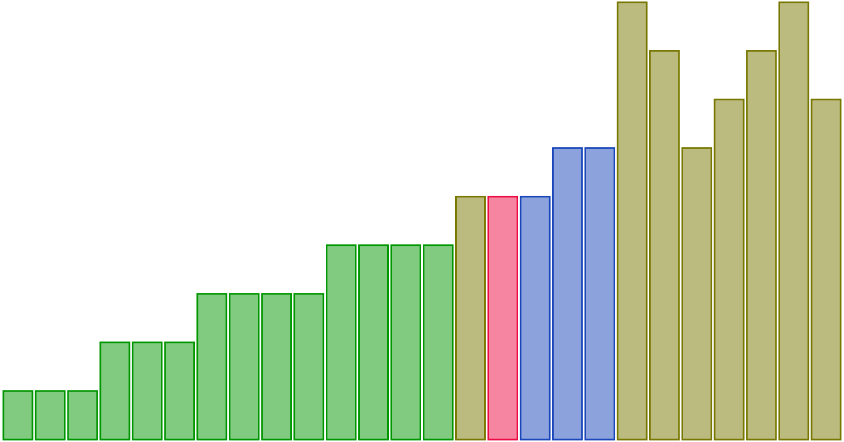


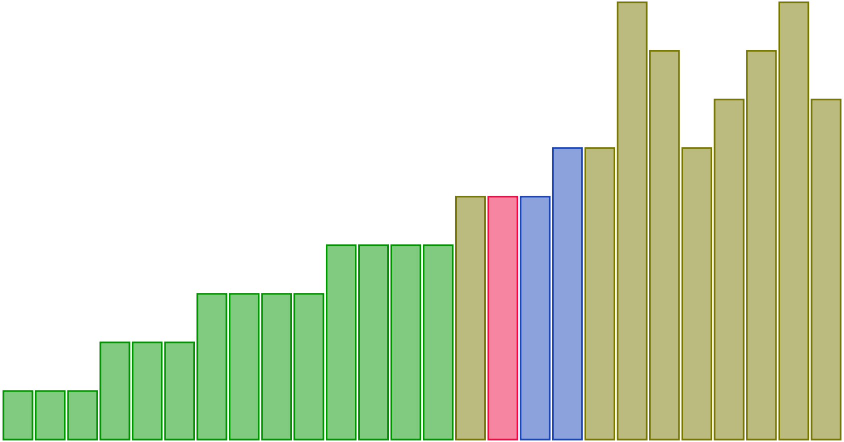


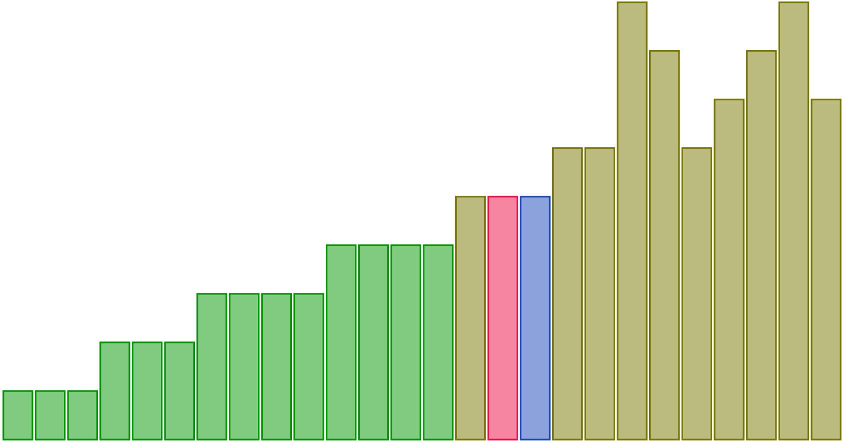


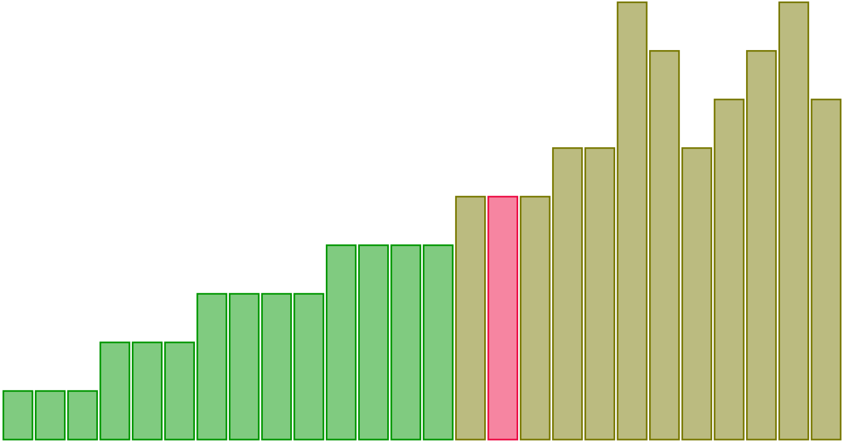


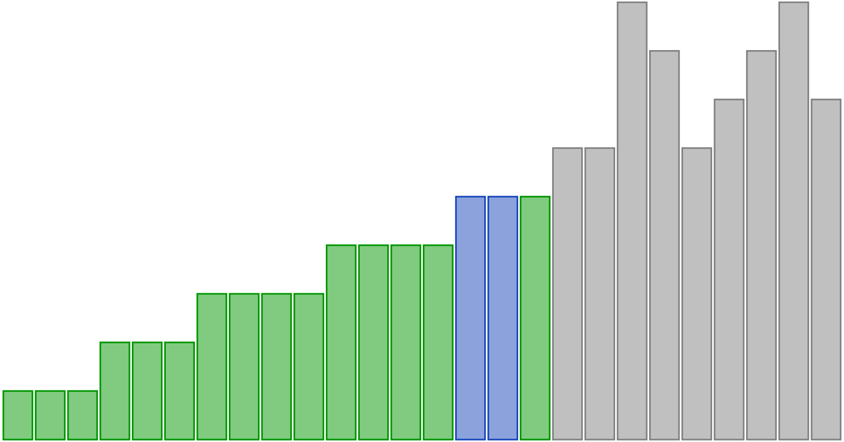


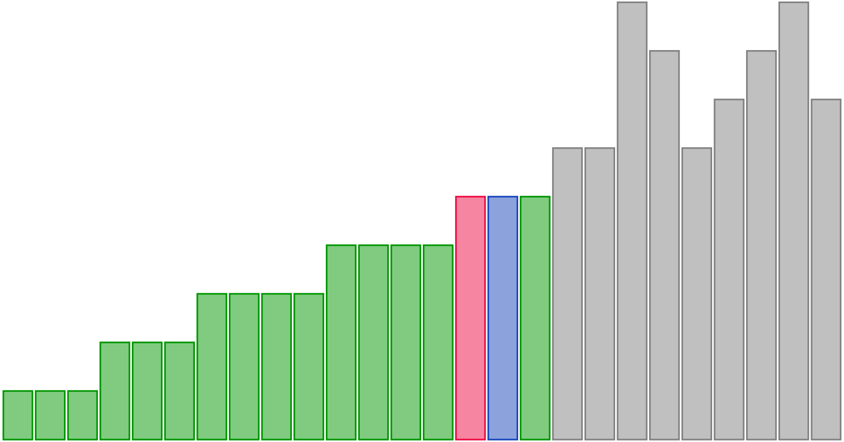


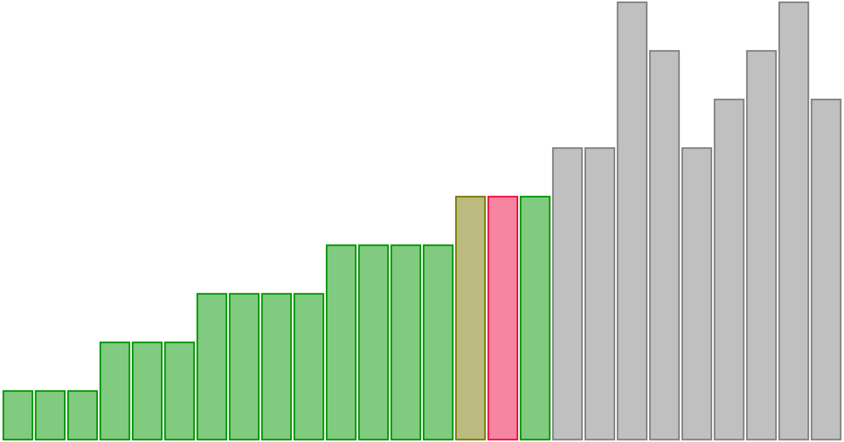


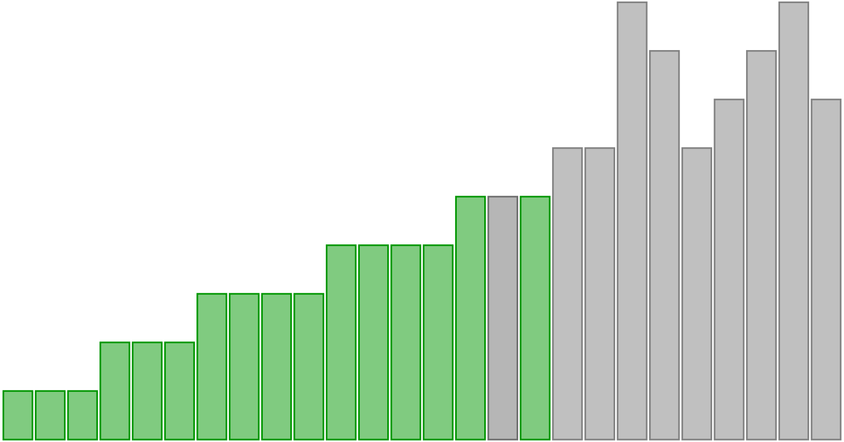


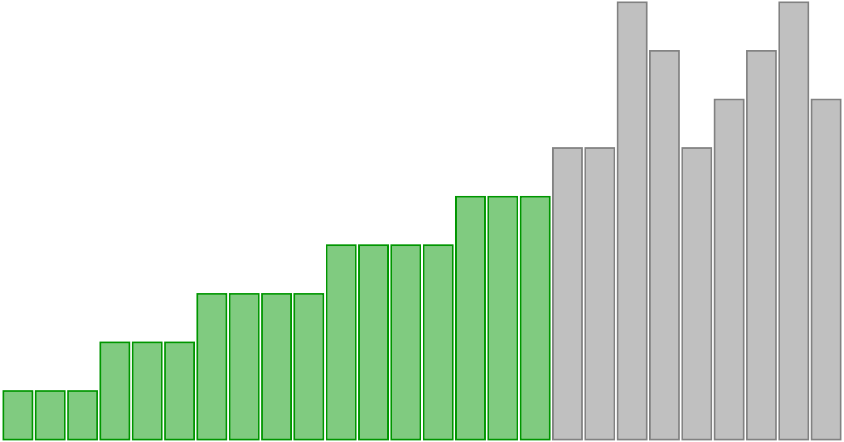


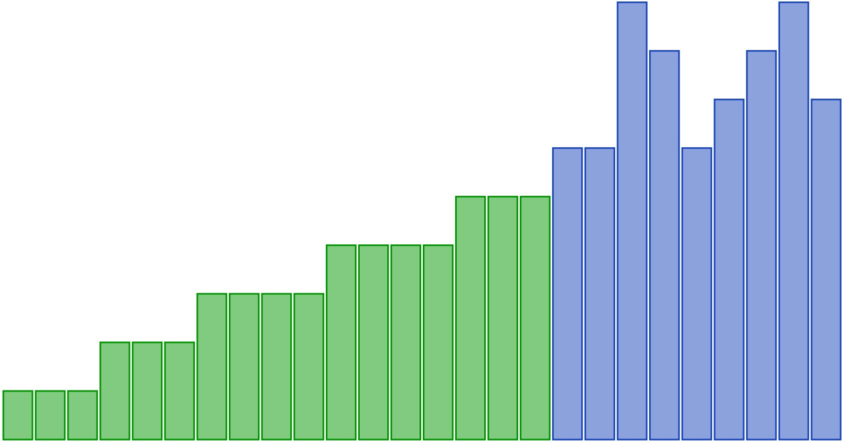


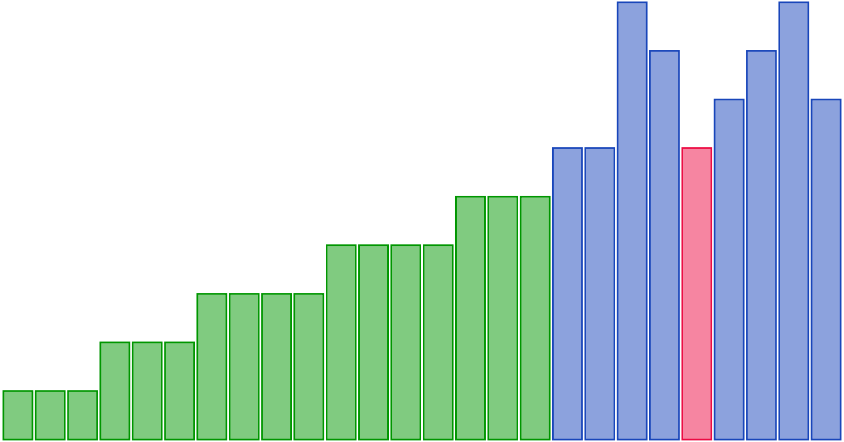


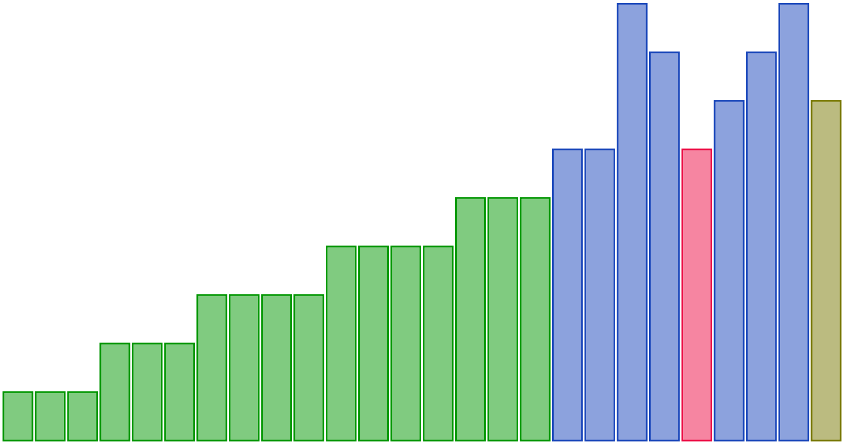


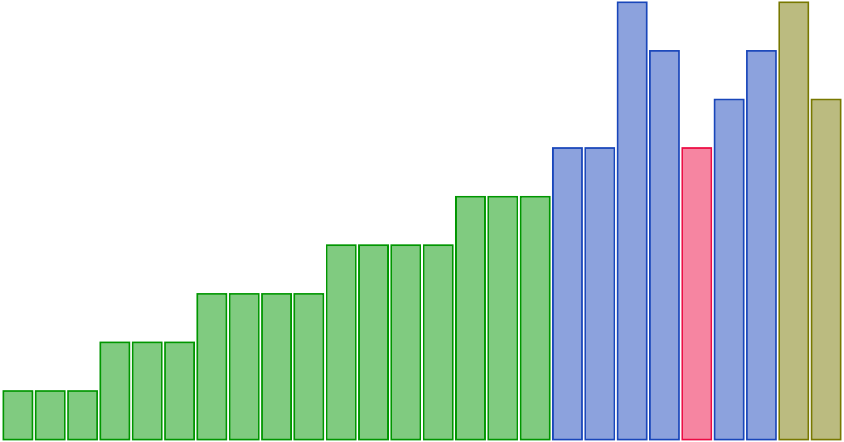


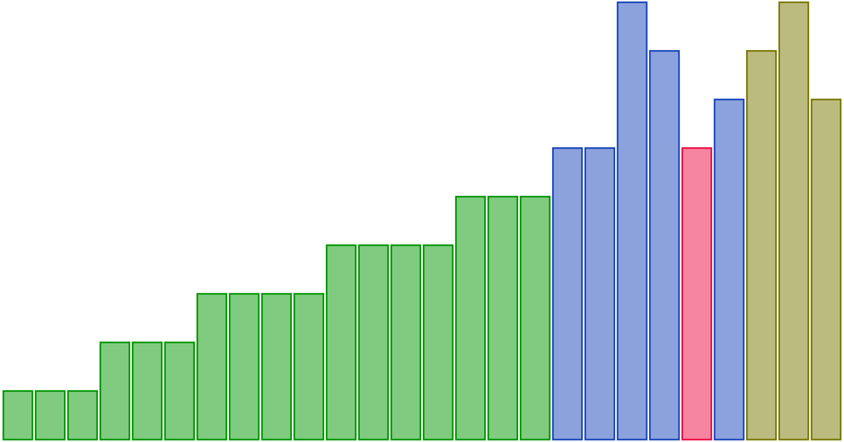


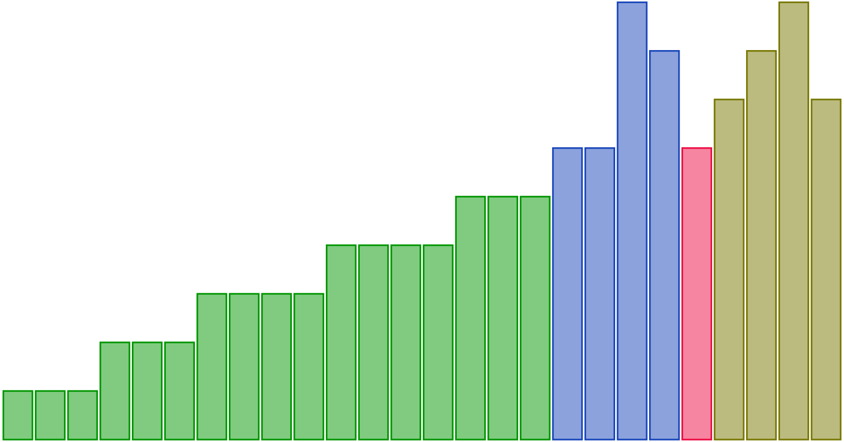


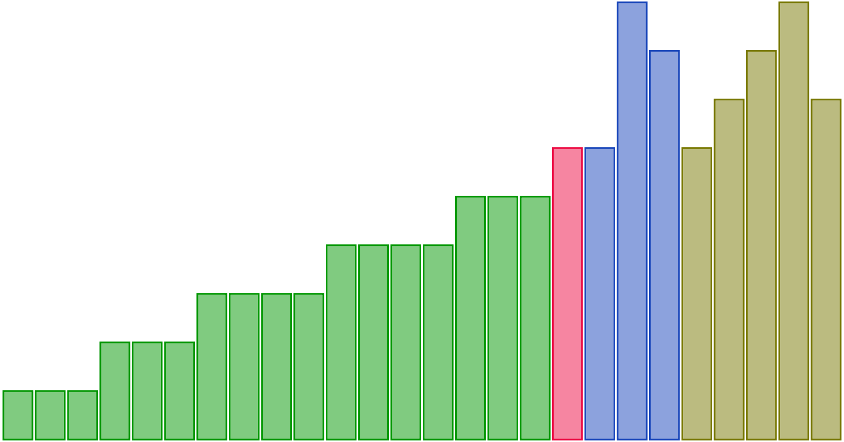


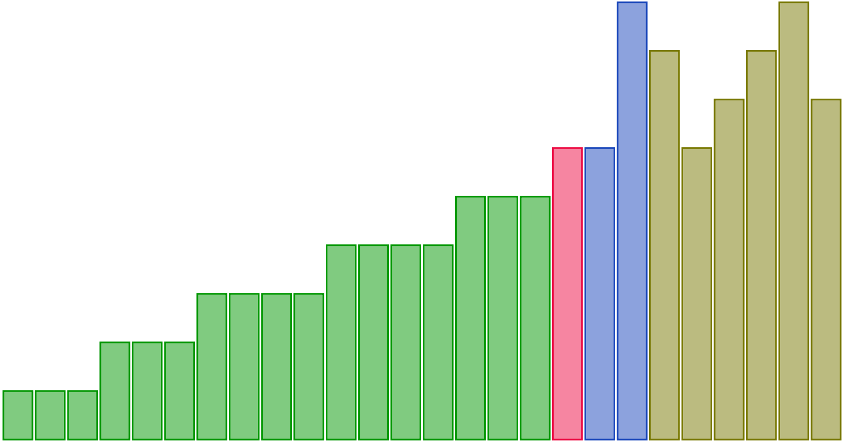


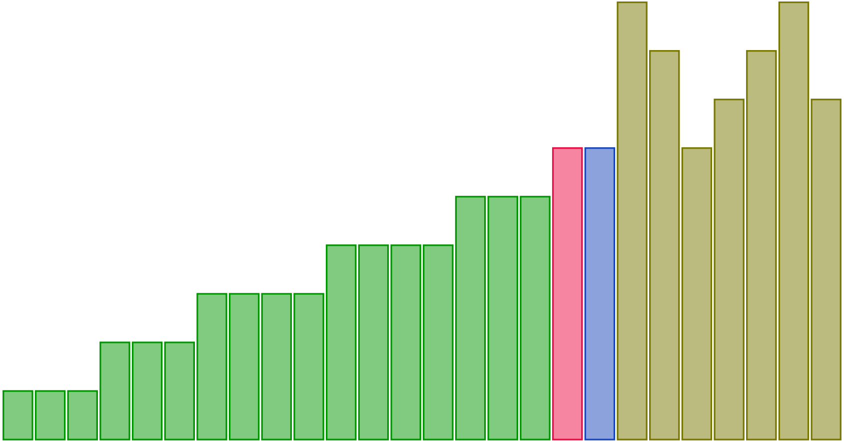


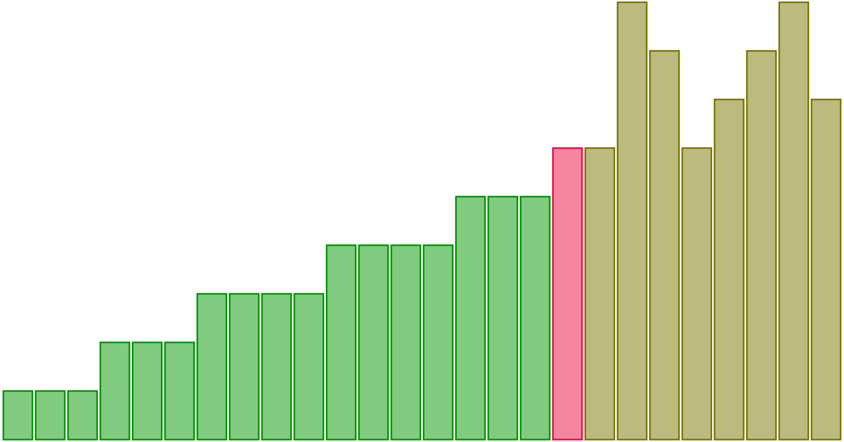


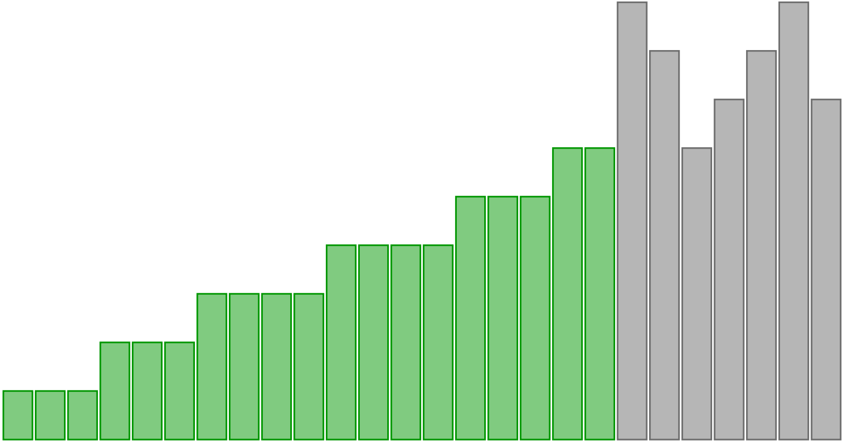


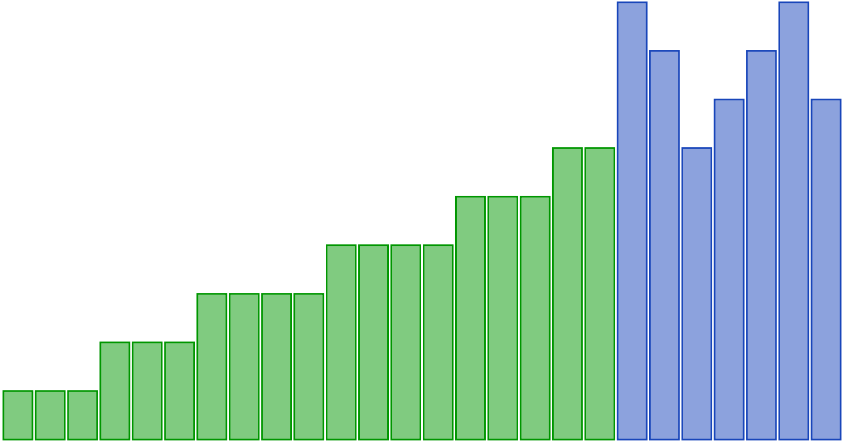


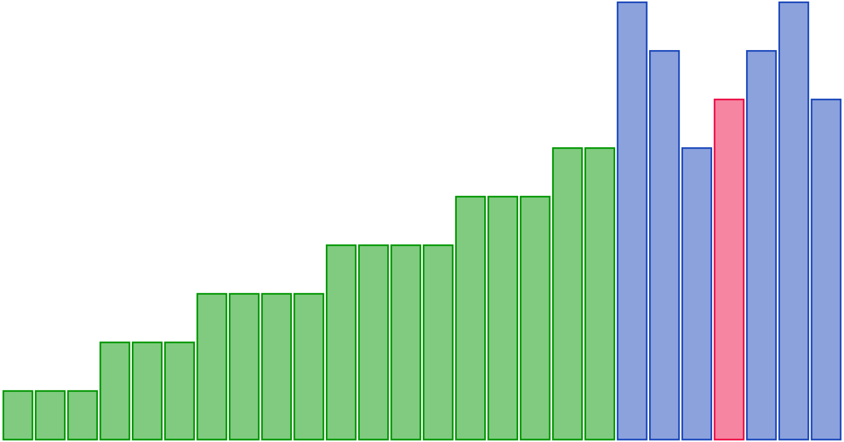


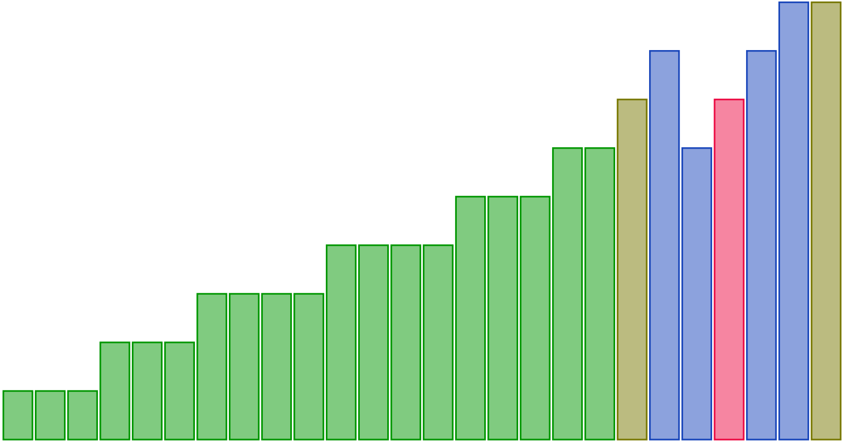


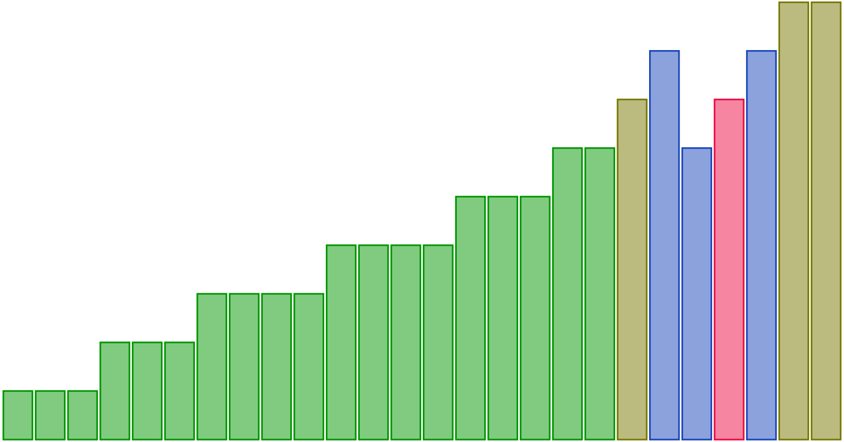


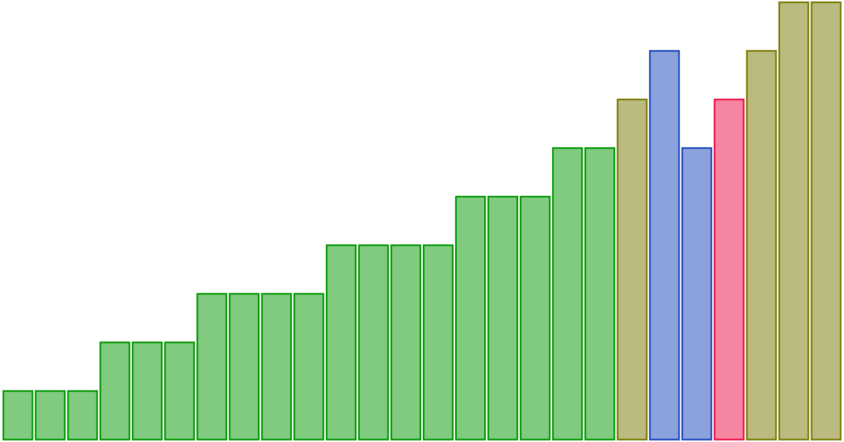


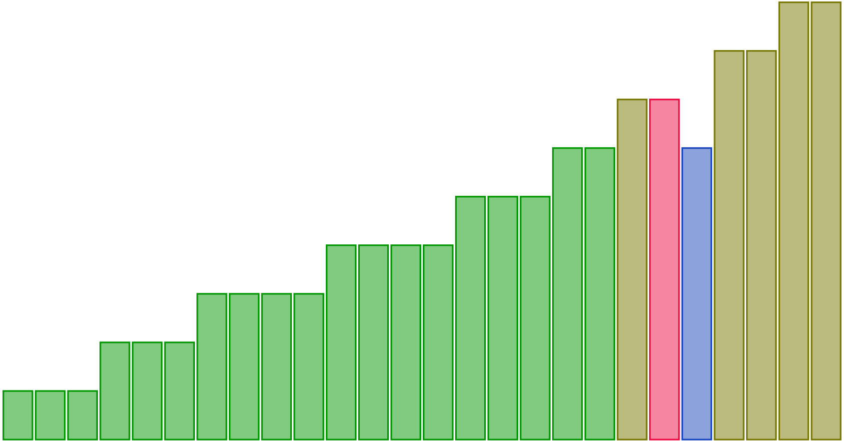


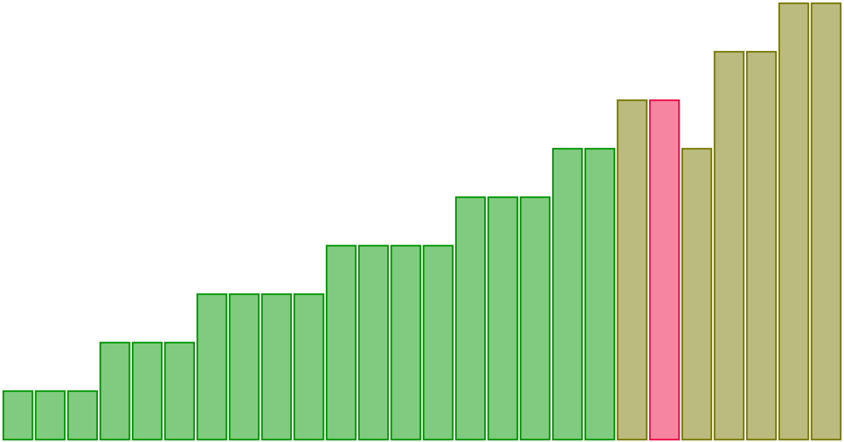


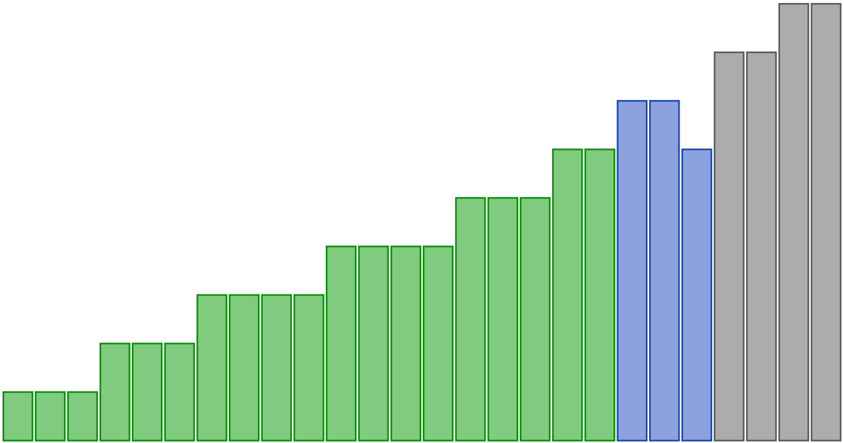


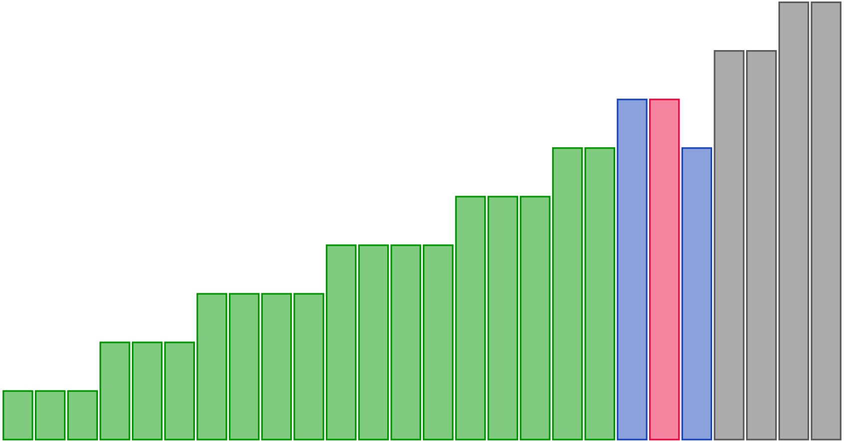


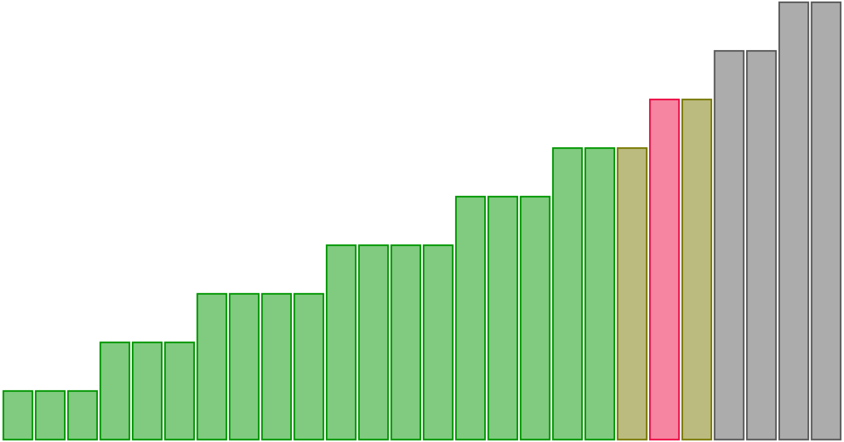


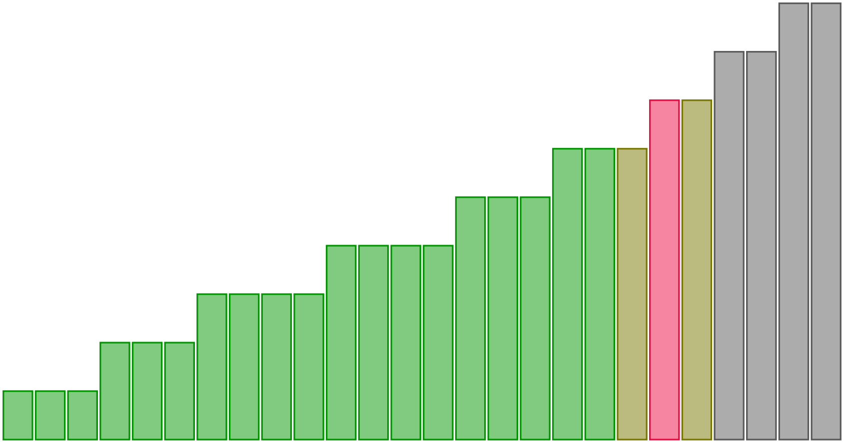


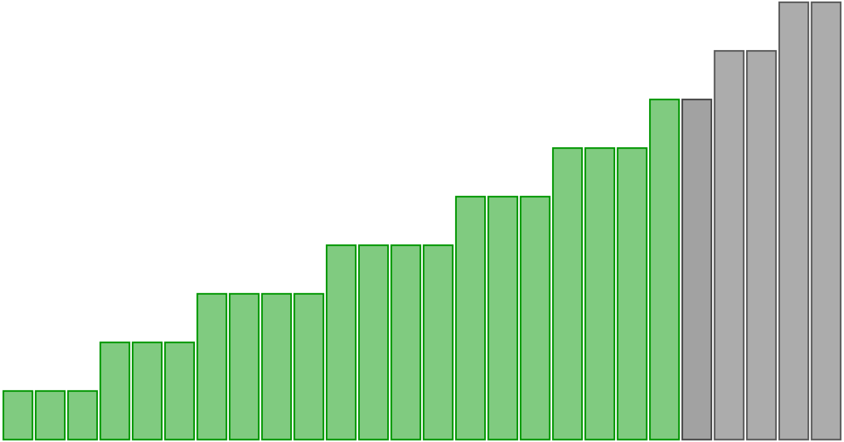


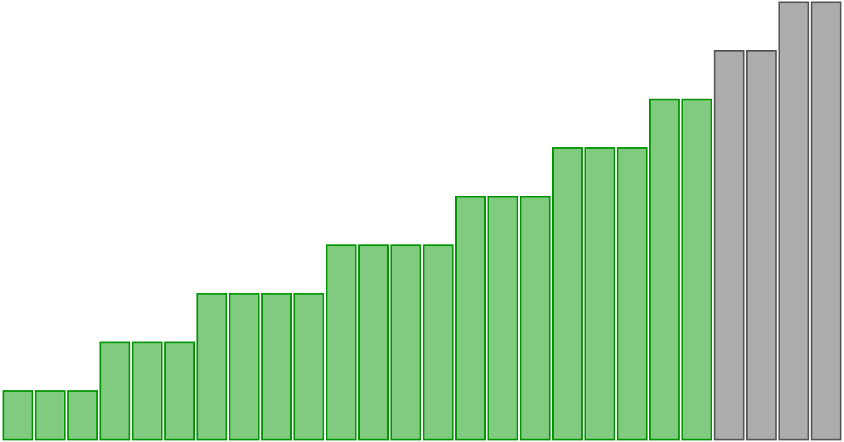


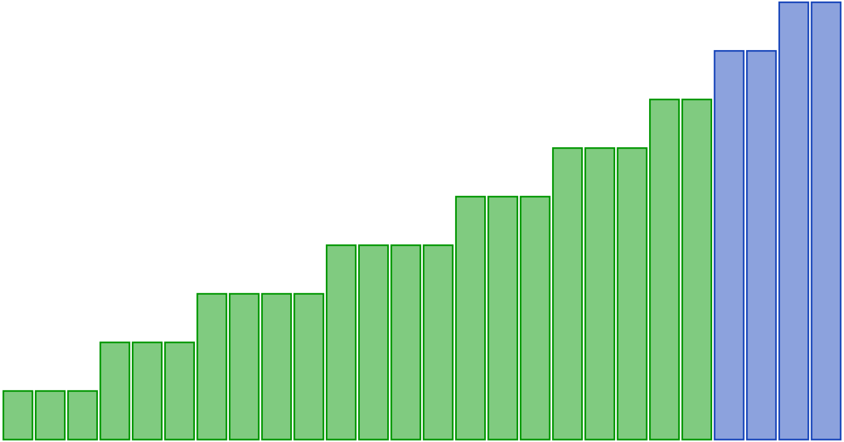


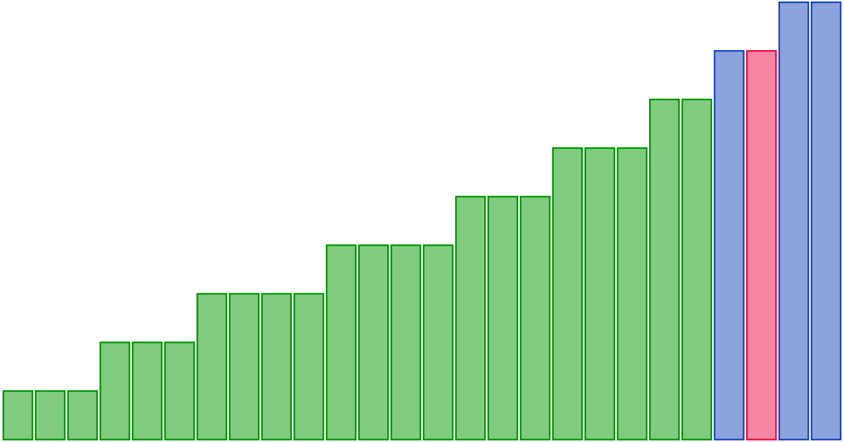


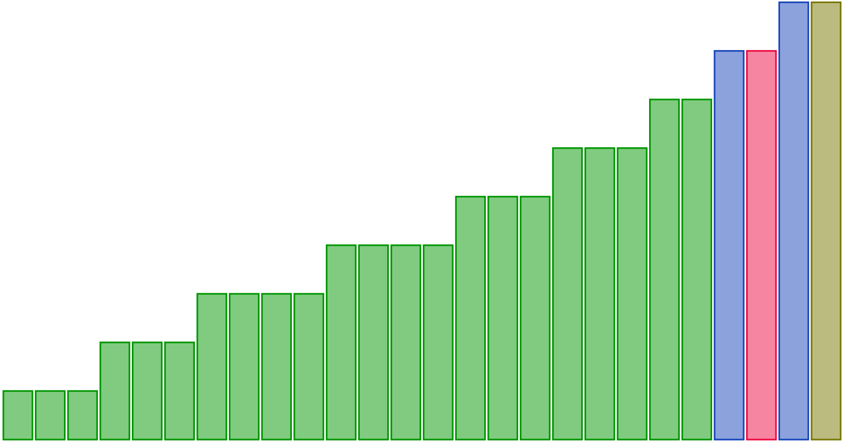


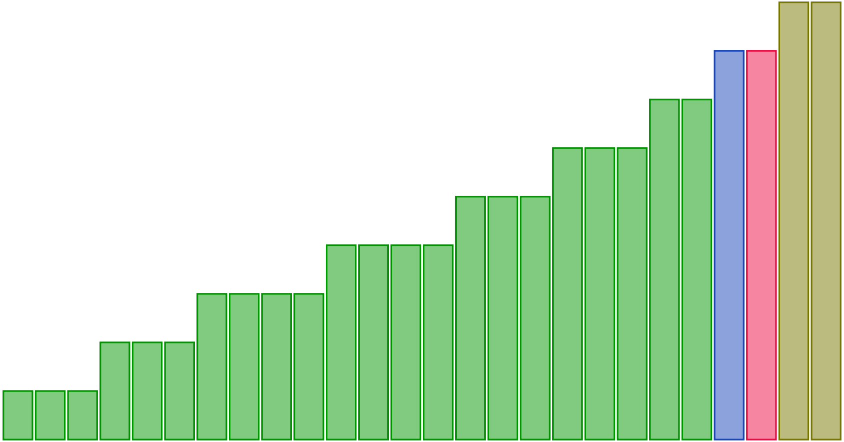


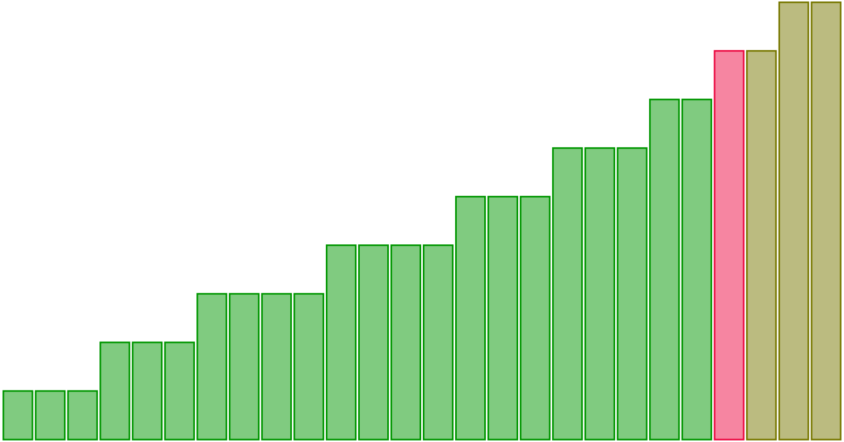


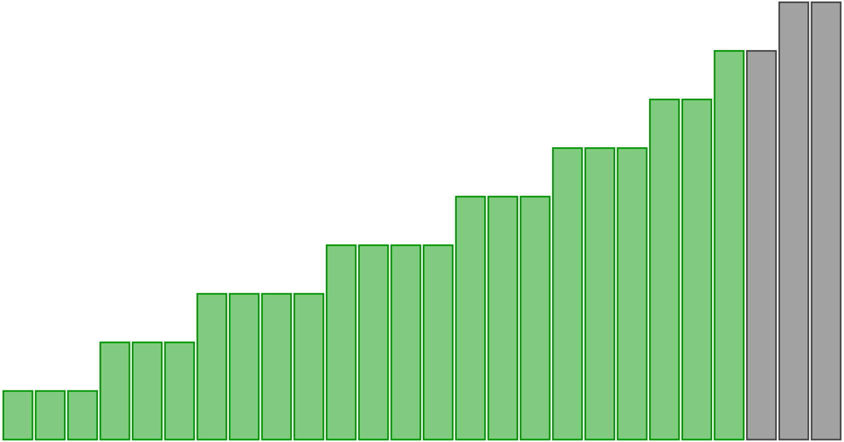


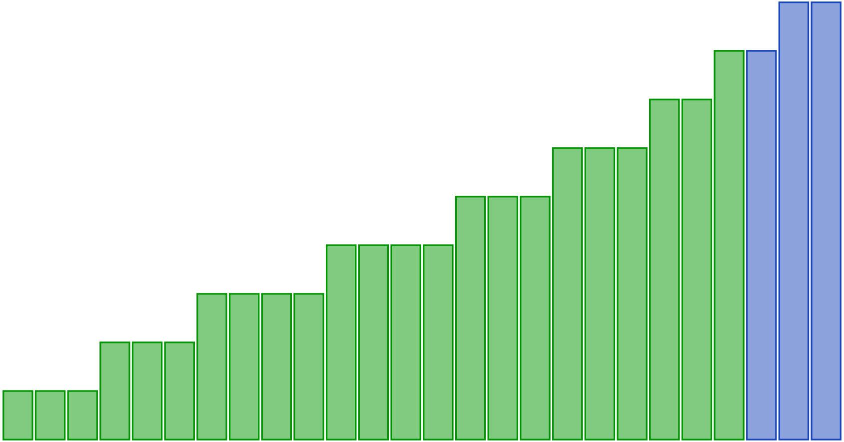


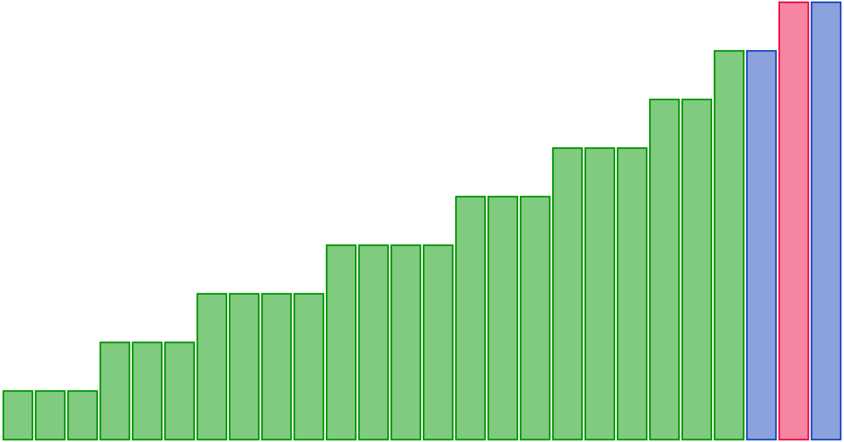


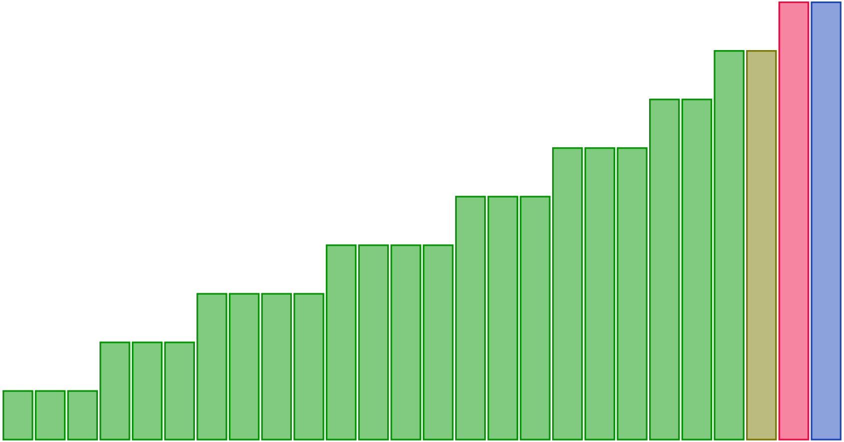


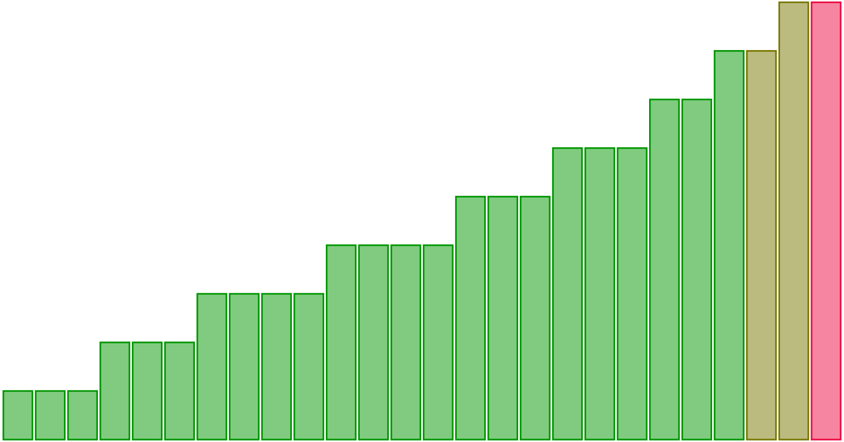


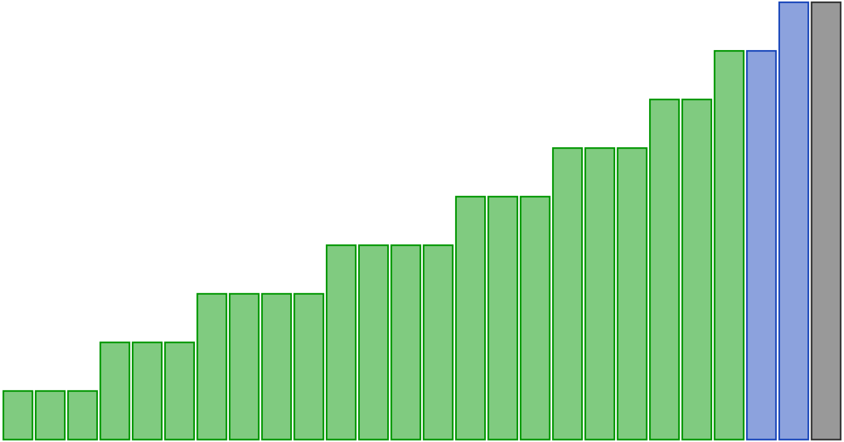


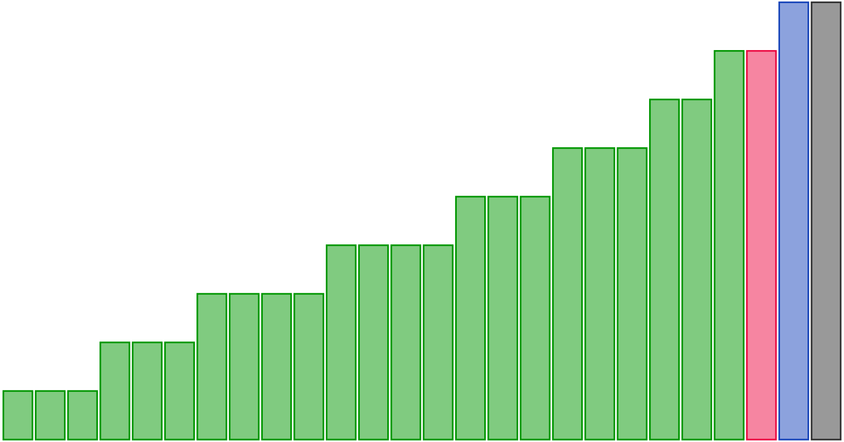


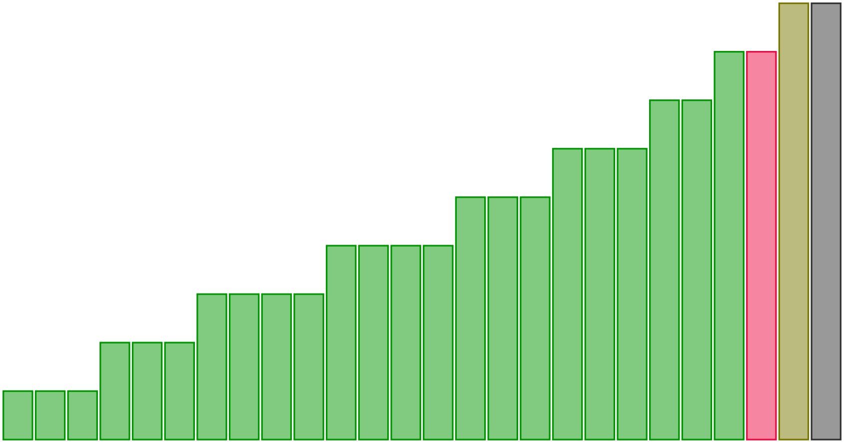


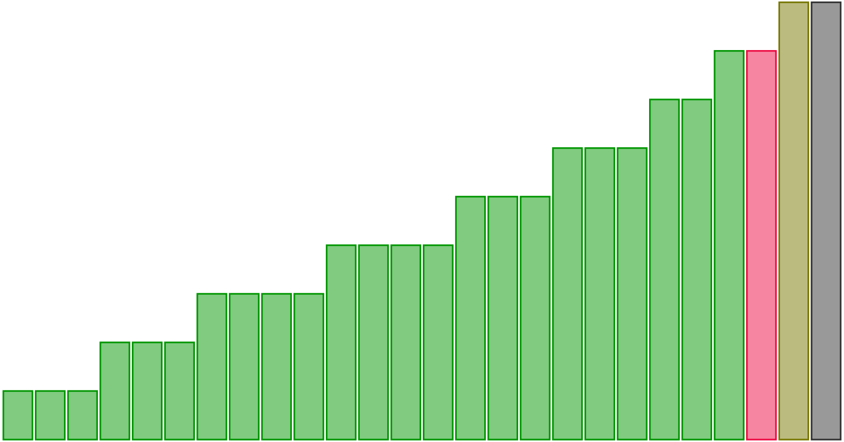


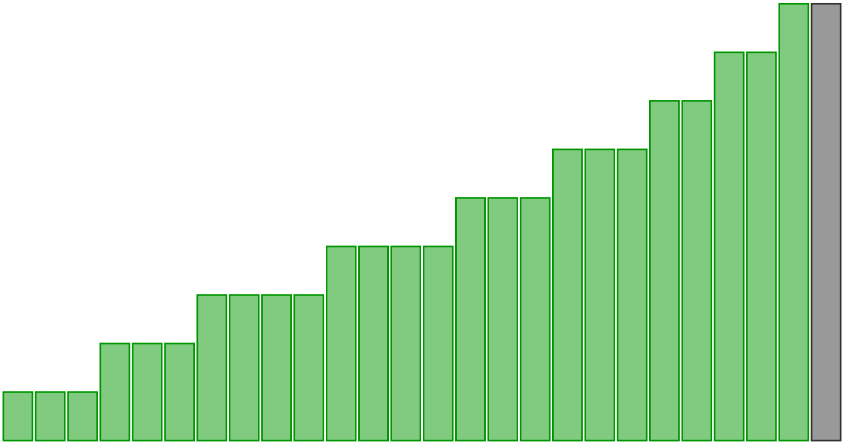


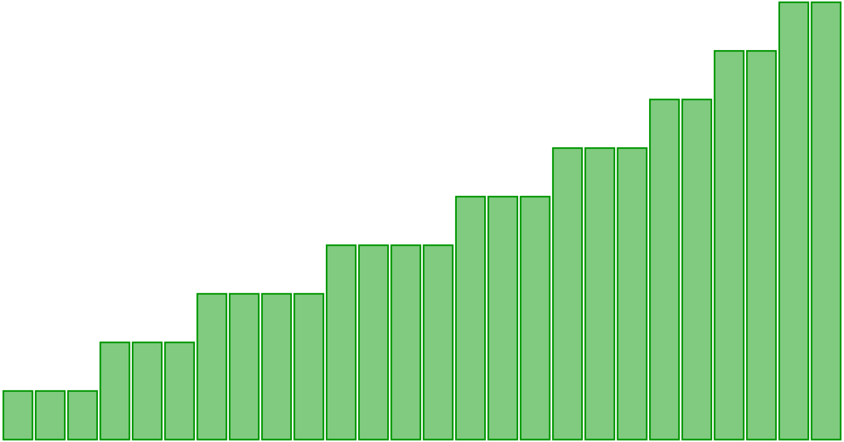












- Lemma: quicksort splits a non-single-element subarray $[s; e]$ into three possibly empty parts $[s; e']$, $(e'; s')$, $[s'; e]$, such that, for some M:
  - both $s' \neq s$ or $e' \neq e$
  - $A[i] \preceq M$ if $i \in [s; e']$
  - $A[i] = M$ if $i \in (e'; s')$
  - $M \preceq A[i]$ if $i \in [s'; e]$

- Lemma: quicksort splits a non-single-element subarray $[s; e]$ into three possibly empty parts $[s; e']$, $(e'; s')$, $[s'; e]$, such that, for some M:
  - both $s' \ne s$ or $e' \ne e$
  - $A[i] \preceq M$ if $i \in [s; e']$
  - $A[i] = M$ if $i \in (e'; s')$
  - $M \preceq A[i]$ if $i \in [s'; e]$

- Proof (1/2):
  - Recall invariants:
    - $[s; s')$ contains elements $\preceq M$
    - $(e'; e]$ contains elements $\succeq M$

```
procedure Quicksort(A, ≺, s, e)
    s' ← s, e' ← e, M ← A[(s + e)/2]
    while s' ≤ e' do
        while A[s'] ≺ M do s' ← s' + 1 end while
        while M ≺ A[e'] do e' ← e' − 1 end while
        if s' ≤ e' then
            A[s'] ⇔ A[e']
            s' ← s' + 1, e' ← e' − 1
        end if
    end while
    if s ≤ e' then Quicksort(A, ≺, s, e') end if
    if s' ≤ e then Quicksort(A, ≺, s', e) end if
end procedure
```

- Lemma: quicksort splits a non-single-element subarray $[s; e]$ into three possibly empty parts $[s; e']$, $(e'; s')$, $[s'; e]$, such that, for some M:
  - both $s' \neq s$ or $e' \neq e$
  - $A[i] \preceq M$ if $i \in [s; e']$
  - $A[i] = M$ if $i \in (e'; s')$
  - $M \preceq A[i]$ if $i \in [s'; e]$
- Proof (1/2):
  - Recall invariants:
    - $[s; s')$ contains elements $\preceq M$
    - $(e'; e]$ contains elements $\succeq M$
  - At the end of the outer **while** $s' > e'$, so every element is either:
    - in $[s; e'] \rightarrow \quad \preceq M$
    - in $[s'; e] \rightarrow \quad \succeq M$
    - in $(e'; s') \rightarrow \preceq M$ and $\succeq M$

```
procedure QUICKSORT(A, ≺, s, e)
    s′ ← s, e′ ← e, M ← A[(s + e)/2]
    while s′ ≤ e′ do
        while A[s′] ≺ M do s′ ← s′ + 1 end while
        while M ≺ A[e′] do e′ ← e′ − 1 end while
        if s′ ≤ e′ then
            A[s′] ⇔ A[e′]
            s′ ← s′ + 1, e′ ← e′ − 1
        end if
    end while
    if s ≤ e′ then QUICKSORT(A, ≺, s, e′) end if
    if s′ ≤ e then QUICKSORT(A, ≺, s′, e) end if
end procedure
```

- Lemma: quicksort splits a non-single-element subarray $[s; e]$ into three possibly empty parts $[s; e']$, $(e'; s')$, $[s'; e]$, such that, for some M:
  - both $s' \neq s$ or $e' \neq e$
  - $A[i] \preceq M$ if $i \in [s; e']$ proven
  - $A[i] = M$ if $i \in (e'; s')$ proven
  - $M \preceq A[i]$ if $i \in [s'; e]$ proven

- Proof (1/2):
  - Recall invariants:
    - $[s; s')$ contains elements $\preceq M$
    - $(e'; e]$ contains elements $\succeq M$
  - At the end of the outer **while** $s' > e'$, so every element is either:
    - in $[s; e'] \rightarrow \ \preceq M$
    - in $[s'; e] \rightarrow \ \succeq M$
    - in $(e'; s') \rightarrow \preceq M$ and $\succeq M$

procedure QUICKSORT(A, $\prec$, s, e)
  $s' \leftarrow s$, $e' \leftarrow e$, $M \leftarrow A[(s+e)/2]$
  while $s' \leq e'$ **do**
    while $A[s'] \prec M$ **do** $s' \leftarrow s' + 1$ **end while**
    while $M \prec A[e']$ **do** $e' \leftarrow e' - 1$ **end while**
    if $s' \leq e'$ **then**
      $A[s'] \Leftrightarrow A[e']$
      $s' \leftarrow s' + 1$, $e' \leftarrow e' - 1$
    end if
  end while
  if $s \leq e'$ **then** QUICKSORT(A, $\prec$, s, e') **end if**
  if $s' \leq e$ **then** QUICKSORT(A, $\prec$, s', e) **end if**
end procedure

- Lemma: quicksort splits a non-single-element subarray $[s; e]$ into three possibly empty parts $[s; e']$, $(e'; s')$, $[s'; e]$, such that, for some M:
  - both $s' \ne s$ or $e' \ne e$
  - $A[i] \preceq M$ if $i \in [s; e']$ proven
  - $A[i] = M$ if $i \in (e'; s')$ proven
  - $M \preceq A[i]$ if $i \in [s'; e]$ proven
- Proof (2/2): Assume $s' = s$. Then $e' < s$. How can that be?

procedure $\textsc{Quicksort}(A, \prec, s, e)$
  $s' \leftarrow s$, $e' \leftarrow e$, $M \leftarrow A[(s+e)/2]$
  while $s' \le e'$ **do**
    while $A[s'] \prec M$ **do** $s' \leftarrow s' + 1$ **end while**
    while $M \prec A[e']$ **do** $e' \leftarrow e' - 1$ **end while**
    if $s' \le e'$ **then**
      $A[s'] \Leftrightarrow A[e']$
      $s' \leftarrow s' + 1$, $e' \leftarrow e' - 1$
    end if
  end while
  if $s \le e'$ **then** $\textsc{Quicksort}(A, \prec, s, e')$ **end if**
  if $s' \le e$ **then** $\textsc{Quicksort}(A, \prec, s', e)$ **end if**
end procedure

- Lemma: quicksort splits a non-single-element subarray $[s; e]$ into three possibly empty parts $[s; e']$, $(e'; s')$, $[s'; e]$, such that, for some M:
  - both $s' \ne s$ or $e' \ne e$
  - $A[i] \preceq M$ if $i \in [s; e']$ proven
  - $A[i] = M$ if $i \in (e'; s')$ proven
  - $M \preceq A[i]$ if $i \in [s'; e]$ proven

- Proof (2/2): Assume $s' = s$. Then $e' < s$. How can that be?
  - $A[s'] \prec M$ loop body never executed

```
procedure Quicksort(A, ≺, s, e)
    s' ← s, e' ← e, M ← A[(s + e)/2]
    while s' ≤ e' do
        while A[s'] ≺ M do s' ← s' + 1 end while
        while M ≺ A[e'] do e' ← e' − 1 end while
        if s' ≤ e' then
            A[s'] ⇔ A[e']
            s' ← s' + 1, e' ← e' − 1
        end if
    end while
    if s ≤ e' then Quicksort(A, ≺, s, e') end if
    if s' ≤ e then Quicksort(A, ≺, s', e) end if
end procedure
```

- Lemma: quicksort splits a non-single-element subarray $[s; e]$ into three possibly empty parts $[s; e']$, $(e'; s')$, $[s'; e]$, such that, for some M:
  - both $s' \ne s$ or $e' \ne e$
  - $A[i] \preceq M$ if $i \in [s; e']$ proven
  - $A[i] = M$ if $i \in (e'; s')$ proven
  - $M \preceq A[i]$ if $i \in [s'; e]$ proven
- Proof (2/2): Assume $s' = s$. Then $e' < s$. How can that be?
  - $A[s'] \prec M$ loop body never executed
  - Inner $s' \le e'$ never happened

procedure QUICKSORT(A, $\prec$, s, e)
  $s' \leftarrow s$, $e' \leftarrow e$, $M \leftarrow A[(s+e)/2]$
  while $s' \le e'$ **do**
    while $A[s'] \prec M$ **do** $s' \leftarrow s' + 1$ **end while**
    while $M \prec A[e']$ **do** $e' \leftarrow e' - 1$ **end while**
    if $s' \le e'$ **then**
      $A[s'] \Leftrightarrow A[e']$
      $s' \leftarrow s' + 1$, $e' \leftarrow e' - 1$
    end if
  end while
  if $s \le e'$ **then** QUICKSORT(A, $\prec$, s, e') **end if**
  if $s' \le e$ **then** QUICKSORT(A, $\prec$, s', e) **end if**
end procedure

- Lemma: quicksort splits a non-single-element subarray $[s; e]$ into three possibly empty parts $[s; e']$, $(e'; s')$, $[s'; e]$, such that, for some M:
  - both $s' \ne s$ or $e' \ne e$
  - $A[i] \preceq M$ if $i \in [s; e']$ proven
  - $A[i] = M$ if $i \in (e'; s')$ proven
  - $M \preceq A[i]$ if $i \in [s'; e]$ proven

- Proof (2/2): Assume $s' = s$. Then $e' < s$. How can that be?
  - $A[s'] \prec M$ loop body never executed
  - Inner $s' \le e'$ never happened
  - Thus, $M \prec A[e']$ loop condition is always true

```
procedure QUICKSORT(A, ≺, s, e)
    s′ ← s, e′ ← e, M ← A[(s + e)/2]
    while s′ ≤ e′ do
        while A[s′] ≺ M do s′ ← s′ + 1 end while
        while M ≺ A[e′] do e′ ← e′ − 1 end while
        if s′ ≤ e′ then
            A[s′] ⇔ A[e′]
            s′ ← s′ + 1, e′ ← e′ − 1
        end if
    end while
    if s ≤ e′ then QUICKSORT(A, ≺, s, e′) end if
    if s′ ≤ e then QUICKSORT(A, ≺, s′, e) end if
end procedure
```

- Lemma: quicksort splits a non-single-element subarray $[s; e]$ into three possibly empty parts $[s; e']$, $(e'; s')$, $[s'; e]$, such that, for some M:
  - both $s' \neq s$ or $e' \neq e$
  - $A[i] \preceq M$ if $i \in [s; e']$ proven
  - $A[i] = M$ if $i \in (e'; s')$ proven
  - $M \preceq A[i]$ if $i \in [s'; e]$ proven

- Proof (2/2): Assume $s' = s$. Then $e' < s$. How can that be?
  - $A[s'] \prec M$ loop body never executed
  - Inner $s' \leq e'$ never happened
  - Thus, $M \prec A[e']$ loop condition is always true
  - But it cannot happen, as M is taken from the array

procedure $\textsc{Quicksort}(A, \prec, s, e)$
 $s' \leftarrow s$, $e' \leftarrow e$, $M \leftarrow A[(s+e)/2]$
 while $s' \leq e'$ **do**
  while $A[s'] \prec M$ **do** $s' \leftarrow s' + 1$ **end while**
  while $M \prec A[e']$ **do** $e' \leftarrow e' - 1$ **end while**
  if $s' \leq e'$ **then**
   $A[s'] \Leftrightarrow A[e']$
   $s' \leftarrow s' + 1$, $e' \leftarrow e' - 1$
  end if
 end while
 if $s \leq e'$ **then** $\textsc{Quicksort}(A, \prec, s, e')$ **end if**
 if $s' \leq e$ **then** $\textsc{Quicksort}(A, \prec, s', e)$ **end if**
end procedure

- Lemma: quicksort splits a non-single-element subarray $[s; e]$ into three possibly empty parts $[s; e']$, $(e'; s')$, $[s'; e]$, such that, for some M:
  - both $s' \neq s$ or $e' \neq e$
  - $A[i] \preceq M$ if $i \in [s; e']$ proven
  - $A[i] = M$ if $i \in (e'; s')$ proven
  - $M \preceq A[i]$ if $i \in [s'; e]$ proven
- Proof (2/2): Assume $s' = s$. Then $e' < s$. How can that be?
  - $A[s'] \prec M$ loop body never executed
  - Inner $s' \leq e'$ never happened
  - Thus, $M \prec A[e']$ loop condition is always true
  - But it cannot happen, as M is taken from the array
  - So, $s' \neq s$. $e' \neq e$ by symmetry.

procedure $\textsc{Quicksort}(A, \prec, s, e)$
  $s' \leftarrow s$, $e' \leftarrow e$, $M \leftarrow A[(s+e)/2]$
  while $s' \leq e'$ **do**
    while $A[s'] \prec M$ **do** $s' \leftarrow s' + 1$ **end while**
    while $M \prec A[e']$ **do** $e' \leftarrow e' - 1$ **end while**
    if $s' \leq e'$ **then**
      $A[s'] \Leftrightarrow A[e']$
      $s' \leftarrow s' + 1$, $e' \leftarrow e' - 1$
    end if
  end while
  if $s \leq e'$ **then** $\textsc{Quicksort}(A, \prec, s, e')$ **end if**
  if $s' \leq e$ **then** $\textsc{Quicksort}(A, \prec, s', e)$ **end if**
end procedure

- Lemma: quicksort splits a non-single-element subarray $[s; e]$ into three possibly empty parts $[s; e']$, $(e'; s')$, $[s'; e]$, such that, for some M:
  - both $s' \neq s$ or $e' \neq e$ proven
  - $A[i] \preceq M$ if $i \in [s; e']$ proven
  - $A[i] = M$ if $i \in (e'; s')$ proven
  - $M \preceq A[i]$ if $i \in [s'; e]$ proven

- Proof (2/2): Assume $s' = s$. Then $e' < s$. How can that be?
  - $A[s'] \prec M$ loop body never executed
  - Inner $s' \leq e'$ never happened
  - Thus, $M \prec A[e']$ loop condition is always true
  - But it cannot happen, as M is taken from the array
  - So, $s' \neq s$. $e' \neq e$ by symmetry.

procedure $\textsc{Quicksort}(A, \prec, s, e)$
  $s' \leftarrow s$, $e' \leftarrow e$, $M \leftarrow A[(s+e)/2]$
  while $s' \leq e'$ **do**
    while $A[s'] \prec M$ **do** $s' \leftarrow s' + 1$ **end while**
    while $M \prec A[e']$ **do** $e' \leftarrow e' - 1$ **end while**
    if $s' \leq e'$ **then**
      $A[s'] \Leftrightarrow A[e']$
      $s' \leftarrow s' + 1$, $e' \leftarrow e' - 1$
    end if
  end while
  if $s \leq e'$ **then** $\textsc{Quicksort}(A, \prec, s, e')$ **end if**
  if $s' \leq e$ **then** $\textsc{Quicksort}(A, \prec, s', e)$ **end if**
end procedure

- Top level: quicksort is correct if it:
  - Does nothing on a single-element subarray

- Top level: quicksort is correct if it:
  - Does nothing on a single-element subarray
  - Splits a non-single-element subarray $[s; e]$ into three possibly empty parts $[s; e']$, $(e'; s')$, $[s'; e]$, such that, for some M:
    - both $s' \neq s$ or $e' \neq e$
    - $A[i] \preceq M$ if $i \in [s; e']$
    - $A[i] = M$ if $i \in (e'; s')$
    - $M \preceq A[i]$ if $i \in [s'; e]$

- Top level: quicksort is correct if it:
  - Does nothing on a single-element subarray
  - Splits a non-single-element subarray $[s; e]$ into three possibly empty parts $[s; e']$, $(e'; s')$, $[s'; e]$, such that, for some M:
    - both $s' \neq s$ or $e' \neq e$
    - $A[i] \preceq M$ if $i \in [s; e']$
    - $A[i] = M$ if $i \in (e'; s')$
    - $M \preceq A[i]$ if $i \in [s'; e]$
  - Calls itself recursively on $[s; e']$ and $[s'; e]$

- Top level: quicksort is correct if it:
  - Does nothing on a single-element subarray
  - Splits a non-single-element subarray $[s; e]$ into three possibly empty parts $[s; e']$, $(e'; s')$, $[s'; e]$, such that, for some M:
    - both $s' \neq s$ or $e' \neq e$
    - $A[i] \preceq M$ if $i \in [s; e']$
    - $A[i] = M$ if $i \in (e'; s')$
    - $M \preceq A[i]$ if $i \in [s'; e]$
  - Calls itself recursively on $[s; e']$ and $[s'; e]$
- Proof:
  - Quicksort terminates, because recursive calls work with strictly smaller array parts

- Top level: quicksort is correct if it:
  - Does nothing on a single-element subarray
  - Splits a non-single-element subarray $[s; e]$ into three possibly empty parts $[s; e']$, $(e'; s')$, $[s'; e]$, such that, for some M:
    - both $s' \neq s$ or $e' \neq e$
    - $A[i] \preceq M$ if $i \in [s; e']$
    - $A[i] = M$ if $i \in (e'; s')$
    - $M \preceq A[i]$ if $i \in [s'; e]$
  - Calls itself recursively on $[s; e']$ and $[s'; e]$
- Proof:
  - Quicksort terminates, because recursive calls work with strictly smaller array parts
  - Any single-element subarray is sorted by definition
  - After recursive calls are done, the subarrays $[s; e']$ and $[s'; e]$ are sorted, and the subarray $(e'; s')$ consists of equal elements, thus also sorted
  - Left part $\preceq$ middle part $\preceq$ right part $\rightarrow$ result is sorted

Running time inside each stack frame: $\Theta(e - s + 1)$

- Each position is visited at least once, at most twice

Running time inside each stack frame: $\Theta(e - s + 1)$

- Each position is visited at least once, at most twice

Best running time: all splits done evenly

Running time inside each stack frame: $\Theta(e - s + 1)$

- Each position is visited at least once, at most twice

Best running time: all splits done evenly

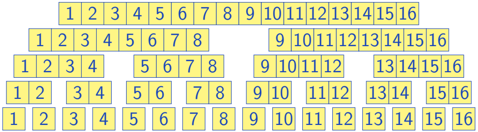
- Example: a sorted array

Running time inside each stack frame: $\Theta(e - s + 1)$

- Each position is visited at least once, at most twice

Best running time: all splits done evenly

- Example: a sorted array
- Look at the call tree to the right

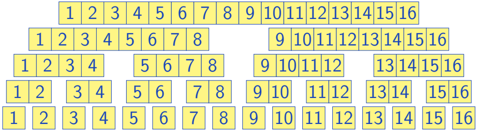


Running time inside each stack frame: $\Theta(e - s + 1)$

- Each position is visited at least once, at most twice

Best running time: all splits done evenly

- Example: a sorted array
- Look at the call tree to the right
- Maximum depth: $\Theta(\log N)$, as every subarray size is at most a half of its parent's size

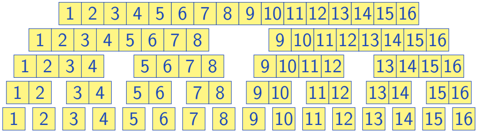


Running time inside each stack frame: $\Theta(e - s + 1)$

- Each position is visited at least once, at most twice

Best running time: all splits done evenly

- Example: a sorted array
- Look at the call tree to the right
- Maximum depth: $\Theta(\log N)$, as every subarray size is at most a half of its parent's size
- Running time: $\Theta(N \log N)$



Running time inside each stack frame: $\Theta(e - s + 1)$

- Each position is visited at least once, at most twice

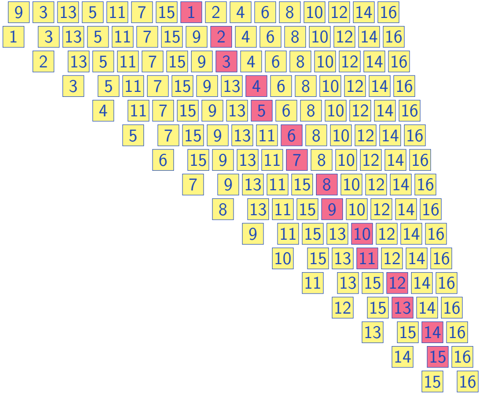
Worst running time: all splits are $1 : K - 1$

Running time inside each stack frame: $\Theta(e - s + 1)$

- Each position is visited at least once, at most twice

Worst running time: all splits are $1 : K - 1$

- Look at the call tree to the right

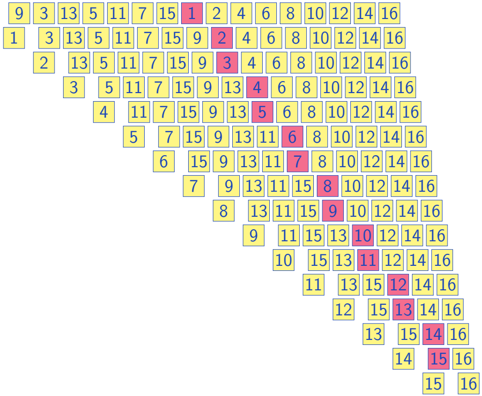


Running time inside each stack frame: $\Theta(e - s + 1)$

- Each position is visited at least once, at most twice

Worst running time: all splits are $1 : K - 1$

- Look at the call tree to the right
- Running time:
  $\Theta\left(\sum_{i=2}^{N} i + N - 1\right) = \Theta(N^2)$

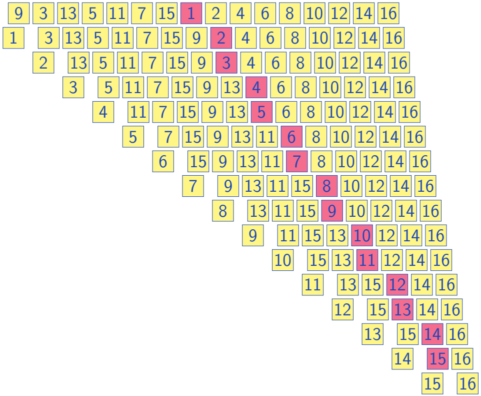


Running time inside each stack frame: $\Theta(e - s + 1)$

- Each position is visited at least once, at most twice

Worst running time: all splits are $1 : K - 1$

- Look at the call tree to the right
- Running time:
  $\Theta\left(\sum_{i=2}^{N} i + N - 1\right) = \Theta(N^2)$
- This is called "quicksort degradation"

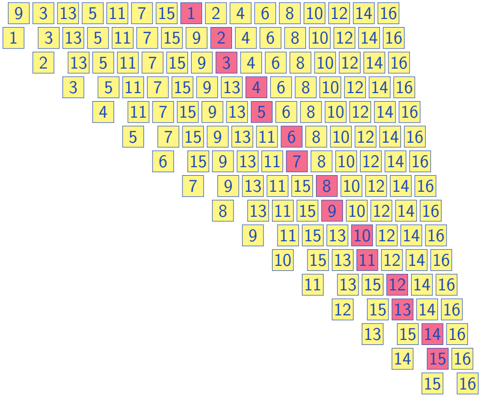


Running time inside each stack frame: $\Theta(e - s + 1)$

- Each position is visited at least once, at most twice

Worst running time: all splits are $1 : K - 1$

- Look at the call tree to the right
- Running time:
  $\Theta\left(\sum_{i=2}^{N} i + N - 1\right) = \Theta(N^2)$
- This is called “quicksort degradation”
- But why is it “quick”?



What is the average running time of quicksort (on all arrays)?

What is the average running time of quicksort (on all arrays)?

- Consider sorting permutations: same as sorting arrays with all elements distinct

What is the average running time of quicksort (on all arrays)?

- Consider sorting permutations: same as sorting arrays with all elements distinct
- Among all permutations, with probability 0.5 the rank of the pivot value will be within $[N/4; 3N/4]$
  - This means that the maximum of subarray sizes is at most $3N/4$
  - This means that the logarithm of the size decreases by at least $\log 4/3$

What is the average running time of quicksort (on all arrays)?

- Consider sorting permutations: same as sorting arrays with all elements distinct
- Among all permutations, with probability 0.5 the rank of the pivot value will be within $[N/4; 3N/4]$
  - This means that the maximum of subarray sizes is at most $3N/4$
  - This means that the logarithm of the size decreases by at least $\log 4/3$
- After splitting, the subarrays are again equivalent to random permutations

What is the average running time of quicksort (on all arrays)?

- Consider sorting permutations: same as sorting arrays with all elements distinct
- Among all permutations, with probability 0.5 the rank of the pivot value will be within $[N/4; 3N/4]$
  - This means that the maximum of subarray sizes is at most $3N/4$
  - This means that the logarithm of the size decreases by at least $\log 4/3$
- After splitting, the subarrays are again equivalent to random permutations

What follows?

What is the average running time of quicksort (on all arrays)?

- Consider sorting permutations: same as sorting arrays with all elements distinct
- Among all permutations, with probability 0.5 the rank of the pivot value will be within $[N/4; 3N/4]$
  - This means that the maximum of subarray sizes is at most $3N/4$
  - This means that the logarithm of the size decreases by at least $\log 4/3$
- After splitting, the subarrays are again equivalent to random permutations

What follows?

- With probability 0.5 the logarithm of the size decreases by at least $\log 4/3$

What is the average running time of quicksort (on all arrays)?

- Consider sorting permutations: same as sorting arrays with all elements distinct
- Among all permutations, with probability 0.5 the rank of the pivot value will be within $[N/4; 3N/4]$
  - This means that the maximum of subarray sizes is at most $3N/4$
  - This means that the logarithm of the size decreases by at least $\log 4/3$
- After splitting, the subarrays are again equivalent to random permutations

What follows?

- With probability 0.5 the logarithm of the size decreases by at least $\log 4/3$
- Expected logarithm decrease: at least $0.5 \log 4/3$

What is the average running time of quicksort (on all arrays)?

- Consider sorting permutations: same as sorting arrays with all elements distinct
- Among all permutations, with probability 0.5 the rank of the pivot value will be within $[N/4; 3N/4]$
  - This means that the maximum of subarray sizes is at most $3N/4$
  - This means that the logarithm of the size decreases by at least $\log 4/3$
- After splitting, the subarrays are again equivalent to random permutations

What follows?

- With probability 0.5 the logarithm of the size decreases by at least $\log 4/3$
- Expected logarithm decrease: at least $0.5 \log 4/3$
- Expected depth: at most $\log N/(0.5 \log 4/3) = O(\log N)$

What is the average running time of quicksort (on all arrays)?

- Consider sorting permutations: same as sorting arrays with all elements distinct
- Among all permutations, with probability 0.5 the rank of the pivot value will be within $[N/4; 3N/4]$
  - This means that the maximum of subarray sizes is at most $3N/4$
  - This means that the logarithm of the size decreases by at least $\log 4/3$
- After splitting, the subarrays are again equivalent to random permutations

What follows?

- With probability 0.5 the logarithm of the size decreases by at least $\log 4/3$
- Expected logarithm decrease: at least $0.5 \log 4/3$
- Expected depth: at most $\log N/(0.5 \log 4/3) = O(\log N)$
- Total work at each depth: $O(N) \to$ average runtime is $O(N \log N)$