Formal Program Derivation

Invariant Diagrams

Two Segment Problem

$\{\exists k: 0 \le k \le N: \forall j: 0 \le j < k: f.j = 0 \wedge \forall j: k \le j < N: f.j = 1\}$

Lecturer: Stephen Sheridan

Problem

Using an invariant diagram specify and, hence, derive an **O(N)** solution to the following: Given f[0..N) containing only 0's and 1's, sort it so that all 0's precede all 1's. Only swap operations are allowed on f.

|[**Con** N: int $\{N > 0\}$

Var

f: array [0..N) of int;

$\{\forall j: 0 \leq j < N: f.j = 0 \vee f.j = 1\}$

k: int;

S

$\{\exists k: 0 \leq k \leq N: \forall j: 0 \leq j < k: f.j = 0 \wedge \forall j: k \leq j < N: f.j = 1\}$

]|

Specification

|[**Con** N: int {N > 0}

Define the necessary constants. Most of this information will be specified in the problem description.

Var

f: array [0..N) of int;

$\{\forall j: 0 \le j < N: f.j = 0 \wedge f.j = 1\}$

Define the necessary variables. Notice that we have defined the array f as a variable because its contents will be changed as a result of the sorting.

Once we define the array we can write an assertion to say that it only contains 0's and 1's.

k: int;

S

Program label

$$\{\exists k: 0 \le k \le N: \forall j: 0 \le j < k: f.j = 0 \wedge \forall j: k \le j < N: f.j = 1\}$$

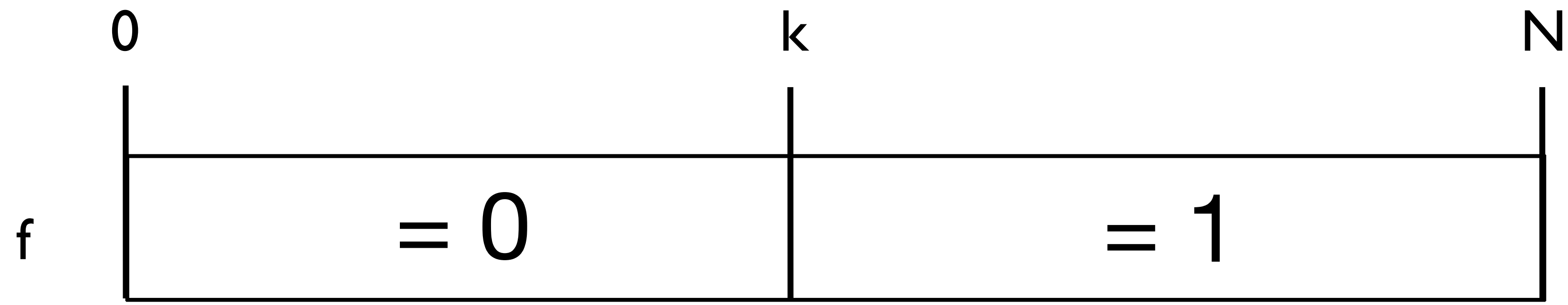
Write a postcondition that represents the problem

]|

Step 1: Postcondition Diagram

We need to draw a diagram that represents the state space after the program finishes. We do this so we can weaken the postcondition and add a unsorted/processed segment in the next step.

$\{\exists k: 0 \le k \le N: \forall j: 0 \le j < k: f.j = 0 \wedge \forall j: k \le j < N: f.j = 1\}$



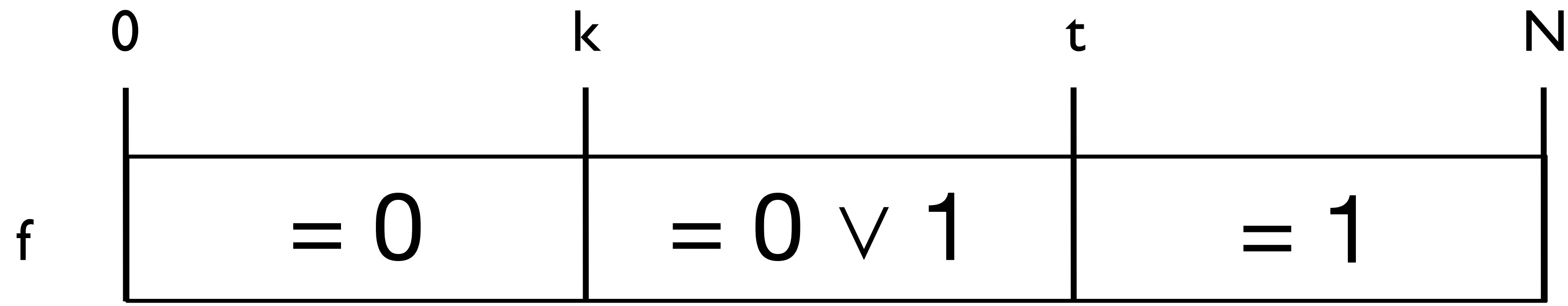
Notice that we have two distinct segments represented by **$0 \le j < k$** and **$k \le j < N$**.

NOTE: this diagram represents an array that has been fully processed. In other words the size of the unprocessed segment is zero.

Step 2: Invariant Diagram

Next we need to draw a diagram that represents a snap shot during the middle of processing. This is actually an invariant diagram as it introduces an unsorted segment and will hold true at the **start, middle** and **end** of processing.

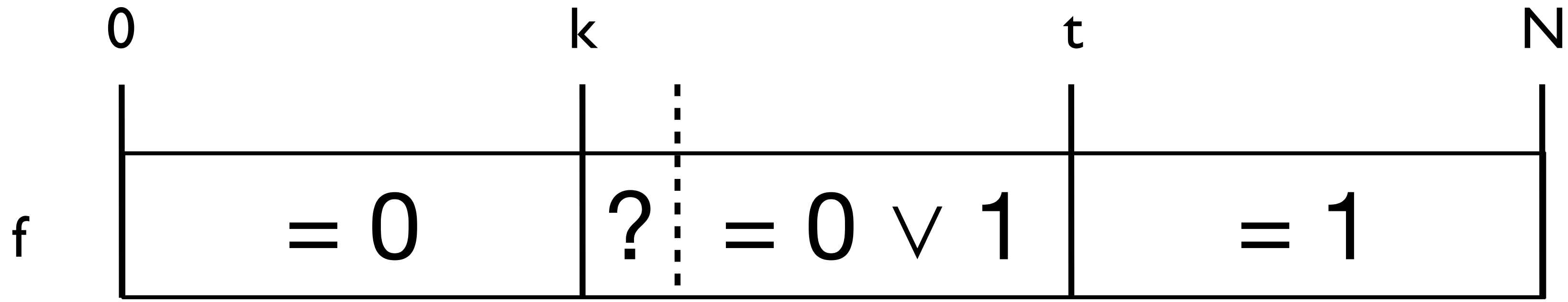
$\{\exists k: 0 \le k \le N: \forall j: 0 \le j < k: f.j = 0 \wedge \boldsymbol{\forall j: k \le j < t: f.j = 0} \vee \boldsymbol{f.j = 1} \wedge \forall j: t \le j < N: f.j = 1\}$



Notice that we now have a segment of unsorted array elements represented by
$\boldsymbol{\forall j: k \le j < t: f.j = 0} \vee \boldsymbol{f.j = 1}$.

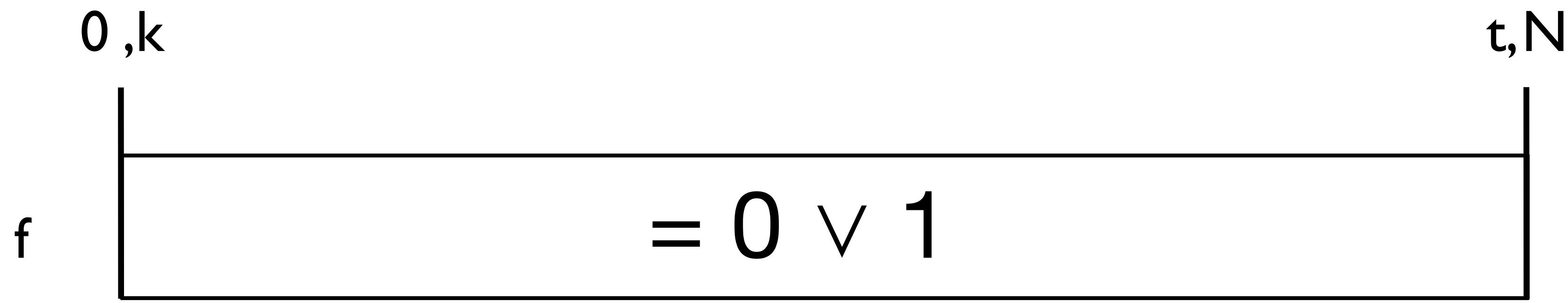
Step 3: Derive O(N) Solution

Using the invariant diagram, we consider the state space at the **start**, **middle** and **end** of processing. We consider the start and end states first because this gives us our variable initialisation and the guard for the loop.

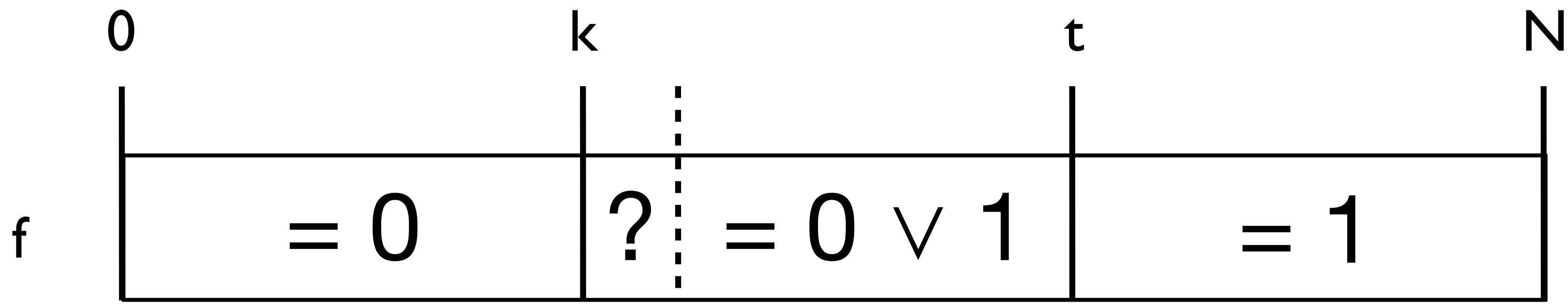


<u>**Start of processing**</u>

f[k..t) represents the whole array. f[0..k) and f[t..N) are both empty. In other words **k,t : = 0,N.**

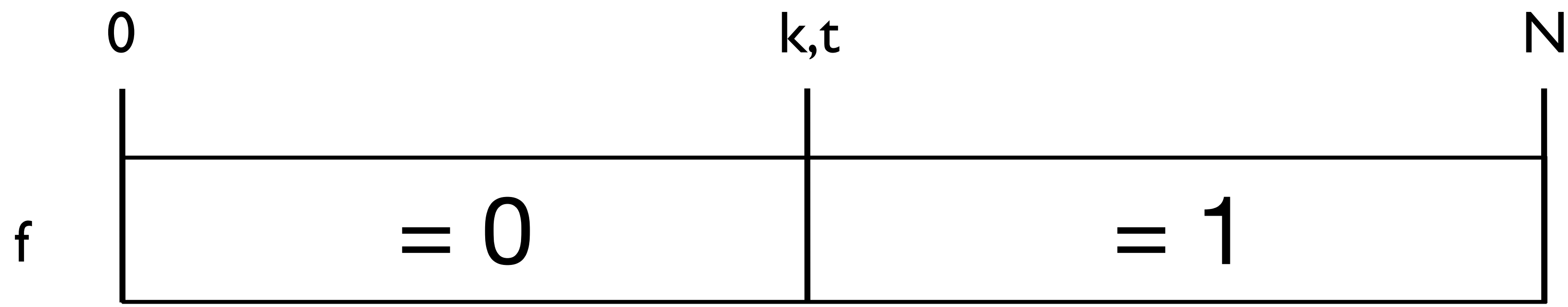


Step 3: Derive O(N) Solution



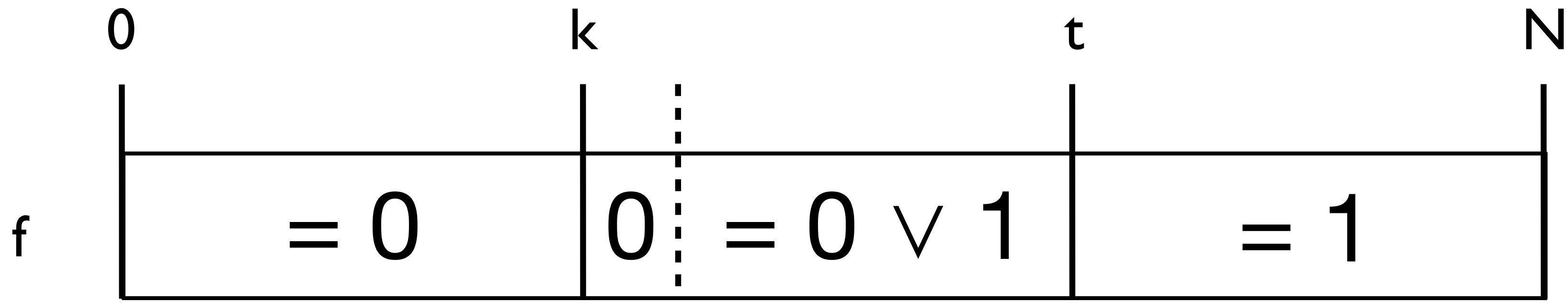
<u>End of processing</u>

f[k..t) will be empty, that is, k = t. Therefore, the guard on the loop is **k < t**.



Step 3: Derive O(N) Solution

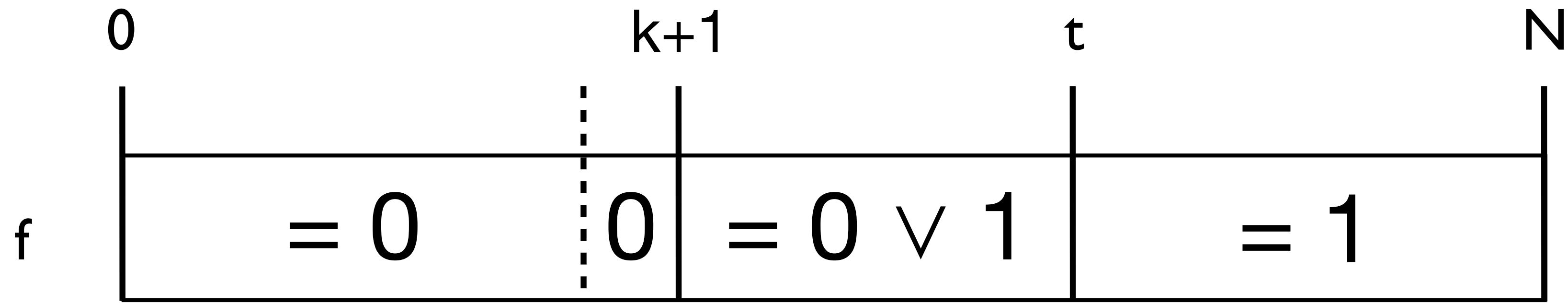
For the middle of processing, the loop body, we will focus on f.k and consider the possible cases. In this problem there are exactly two possibilities, **f.k = 0** $\vee$ **f.k = 1**.



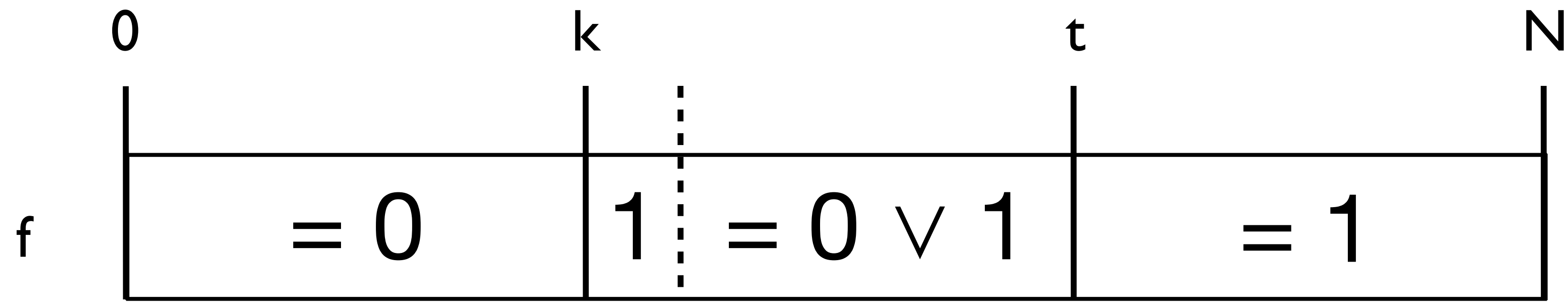
<u>Middle of processing</u>

f.k = 0 => Simply increase k by 1

k := k + 1

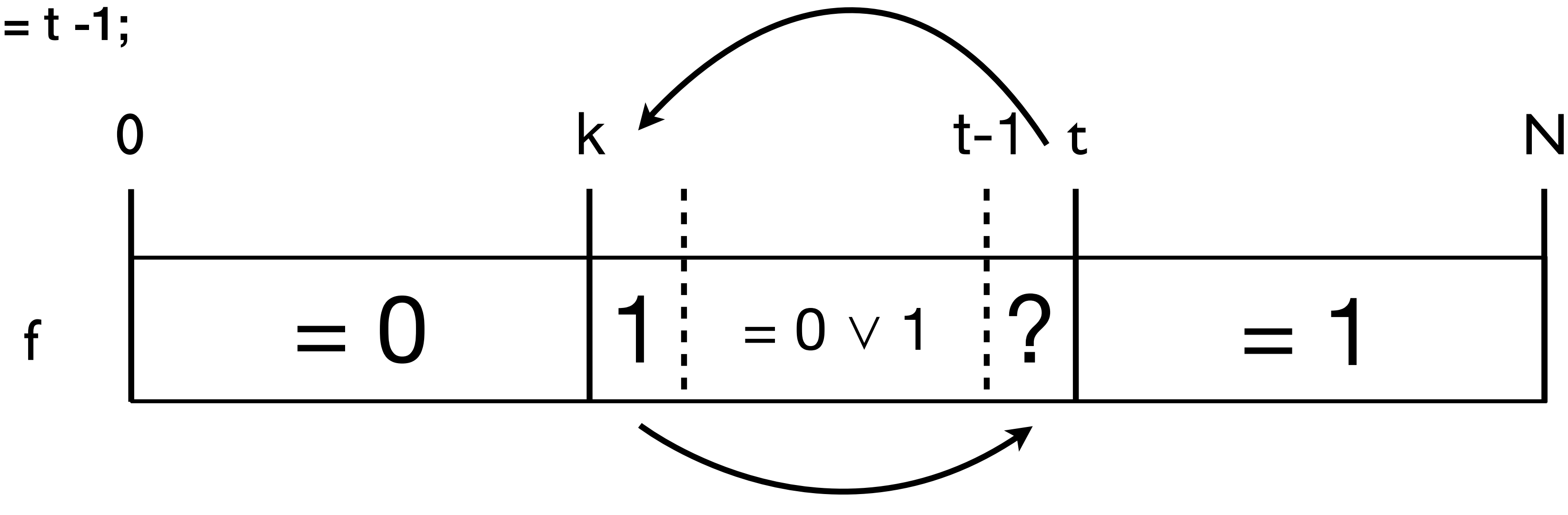


Step 3: Derive O(N) Solution



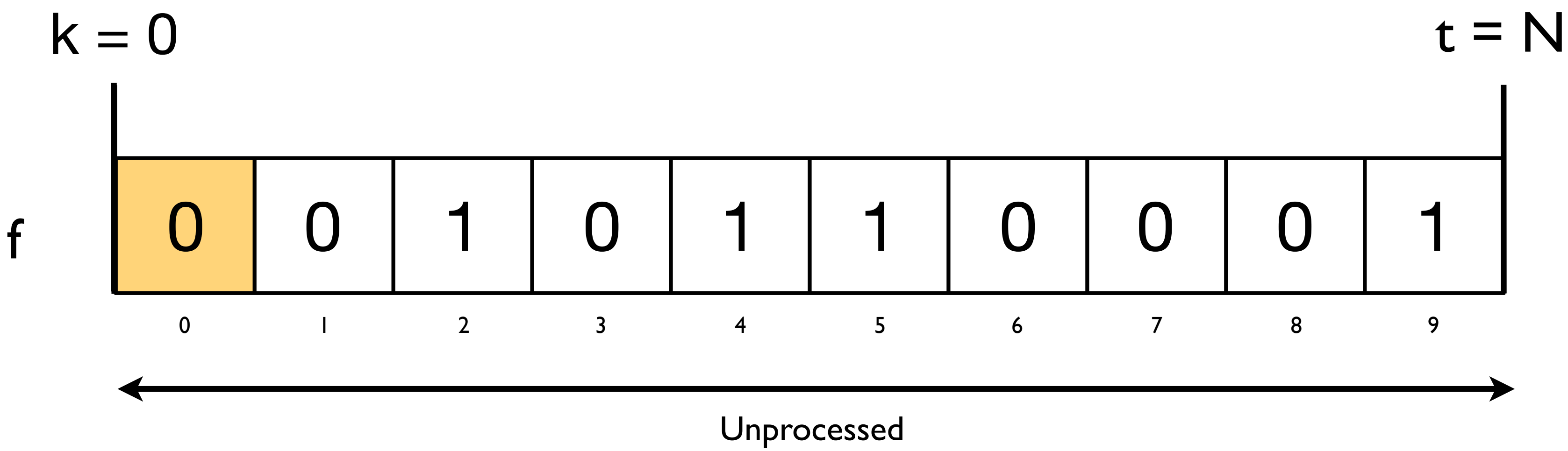
<u>**Middle of processing**</u>

f.k = 1 => Swap f.k with f.t – 1 and decrement t.
  f.k,f.t – 1 := f.t-1, f.k;
  t := t -1;



Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.k allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



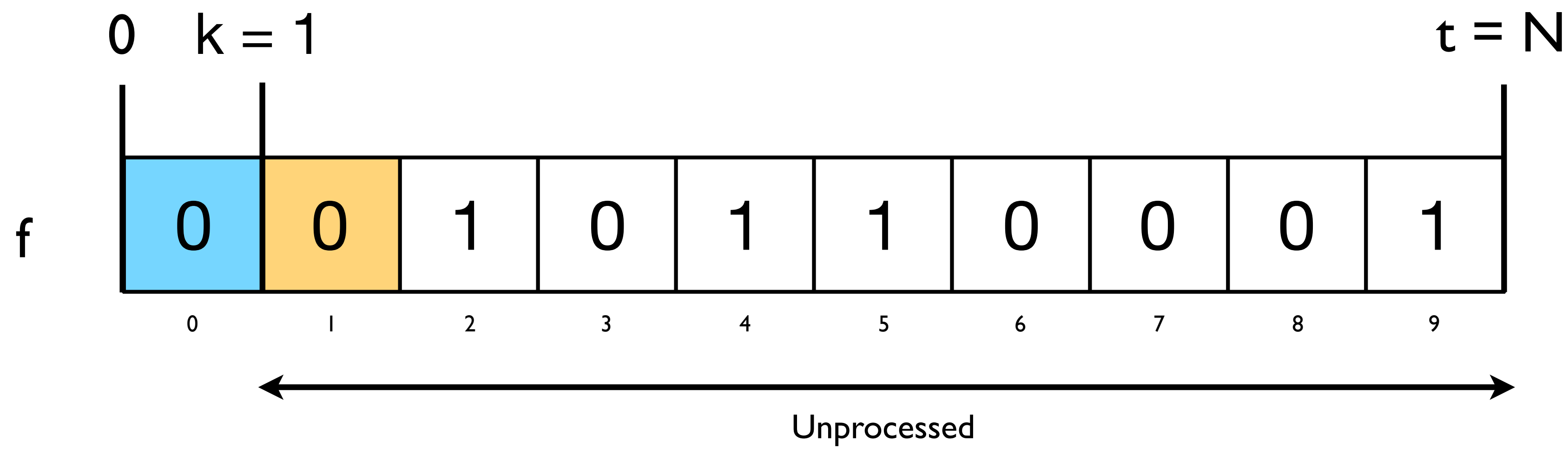
f.k = 0 => k:=k+1

Bound function t - k

t - 0 = 10

Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.k allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



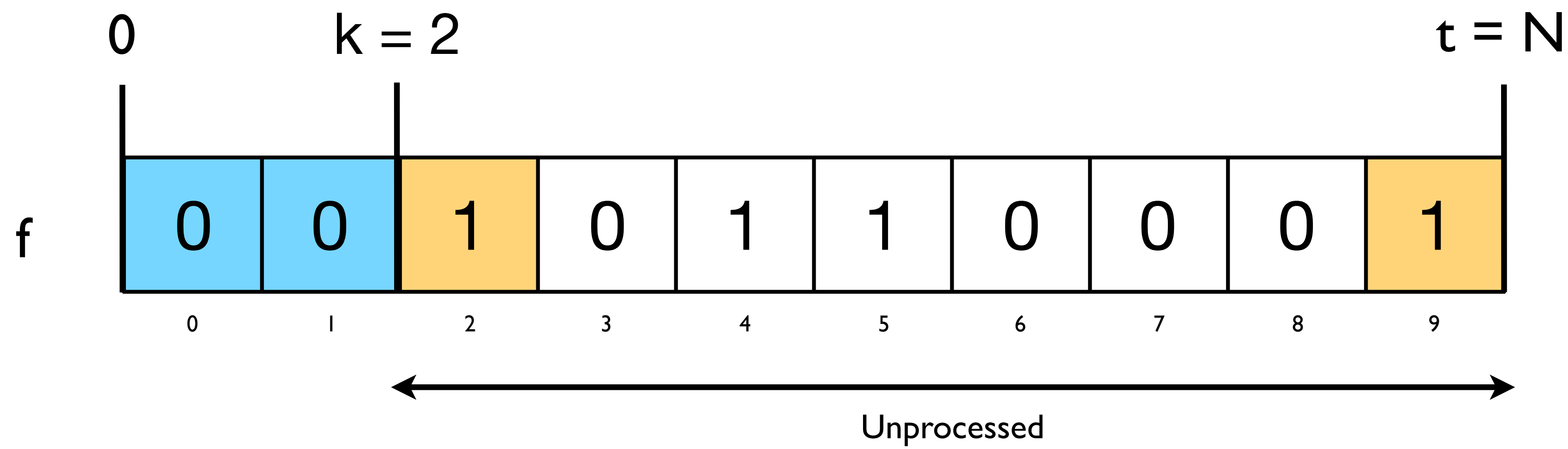
f.k = 0 => k:=k+1

Bound function t - k

t - 1 = 9

Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.k allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



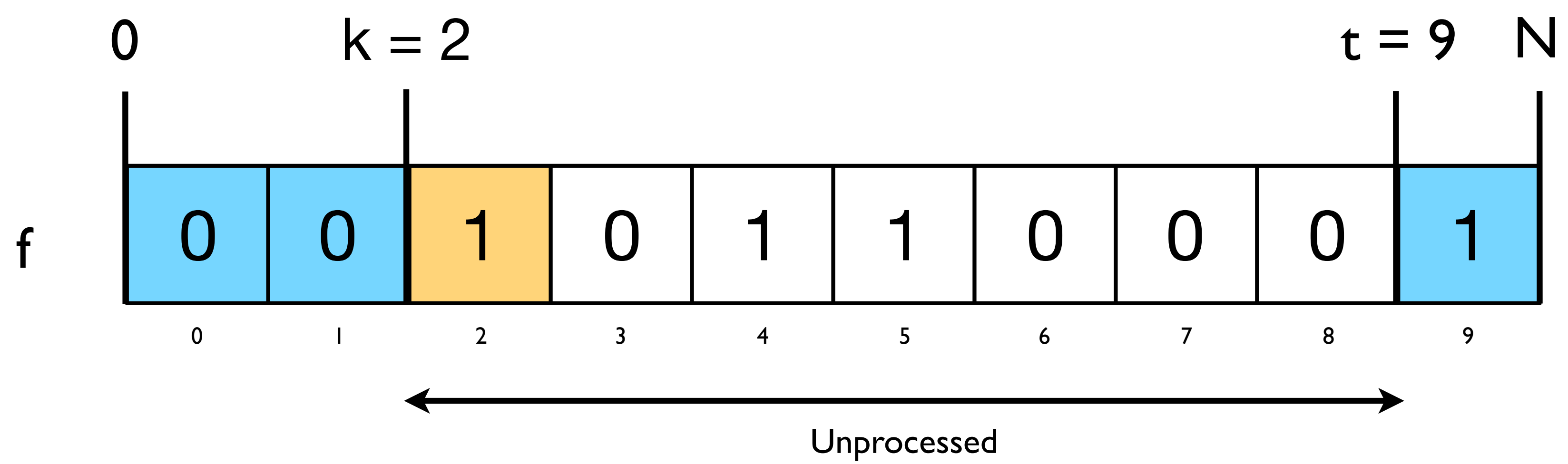
f.k = 1 => f.k, f.t-1 := f.t-1, f.k
t := t-1

Bound function t - k

t - 2 = 8

Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.k allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



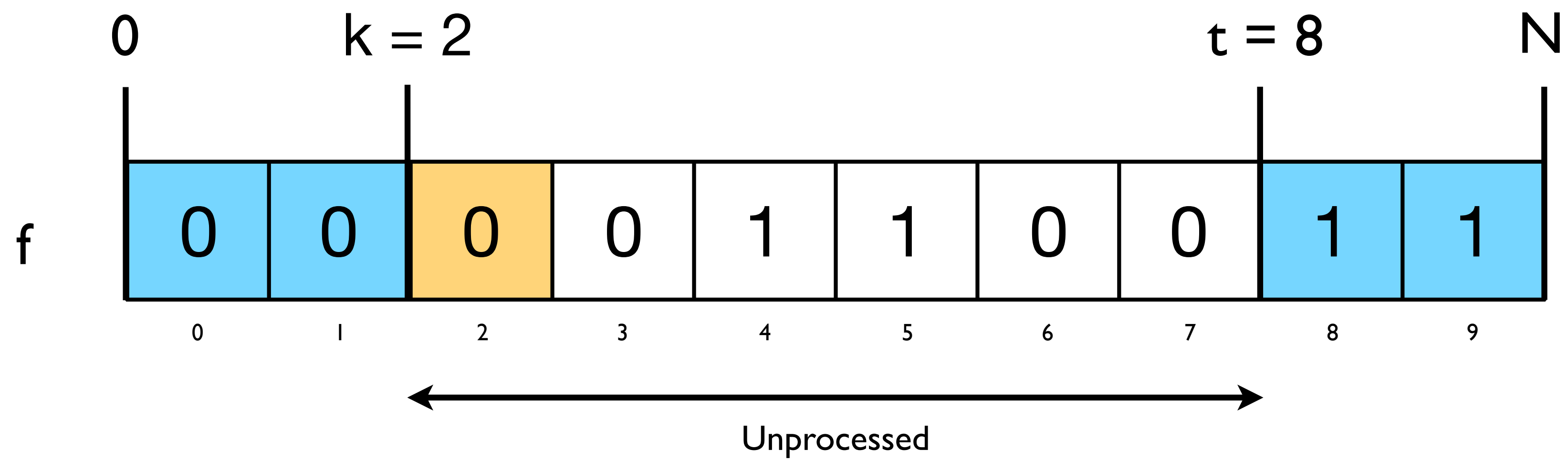
f.k = 1 => f.k, f.t-1 := f.t-1, f.k
t := t-1

Bound function t - k

9 - 2 = 7

Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.k allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



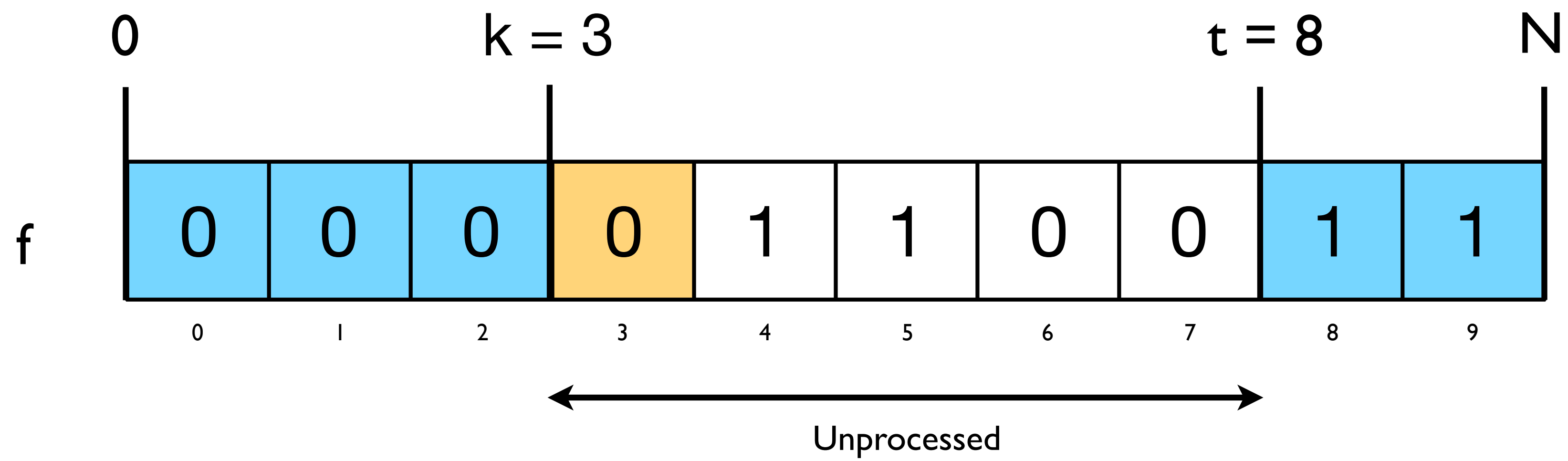
f.k = 0 => k:=k+1

Bound function t - k

8 - 2 = 6

Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.k allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



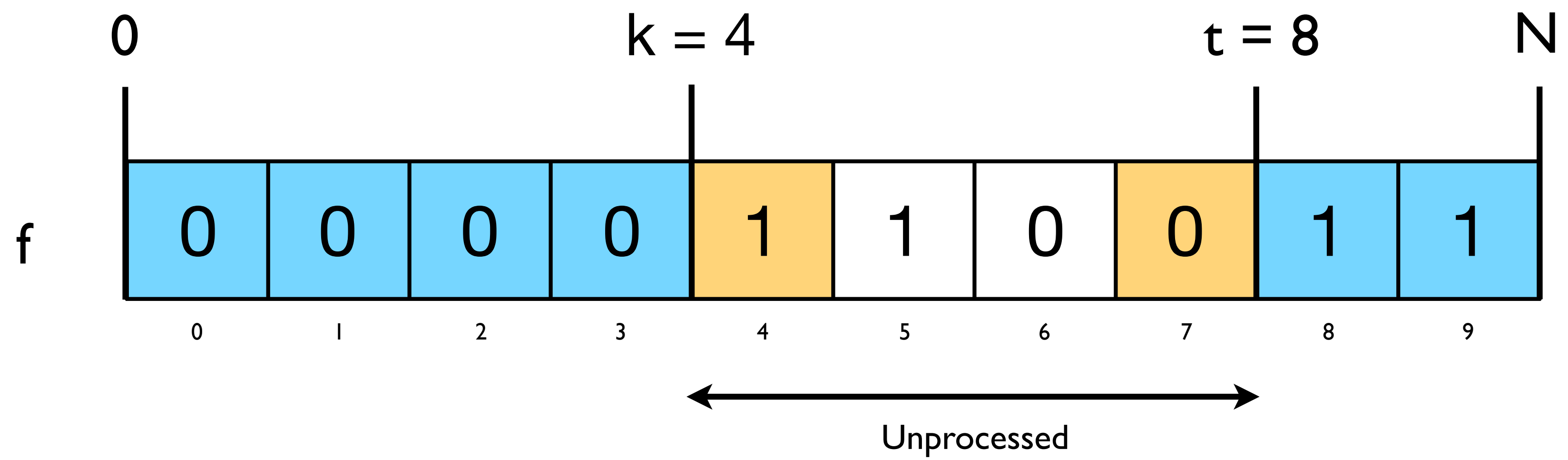
f.k = 0 => k:=k+1

Bound function t - k

8 - 3 = 5

Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.k allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



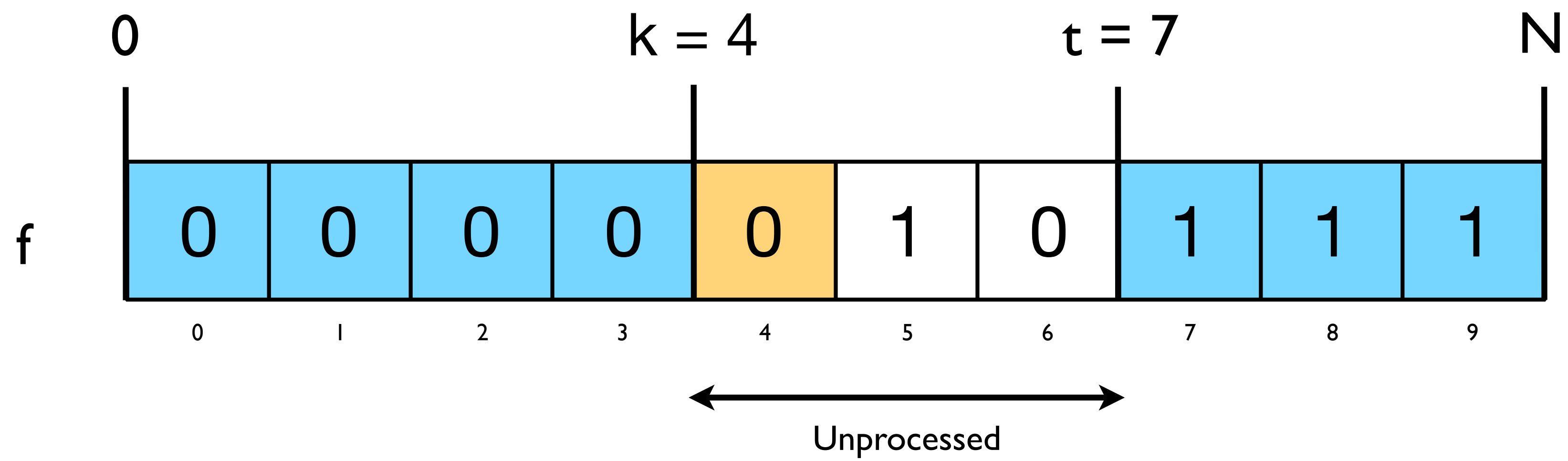
f.k = 1 => f.k, f.t-1 := f.t-1, f.k
t := t-1

Bound function t - k

8 - 4 = 4

Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.k allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



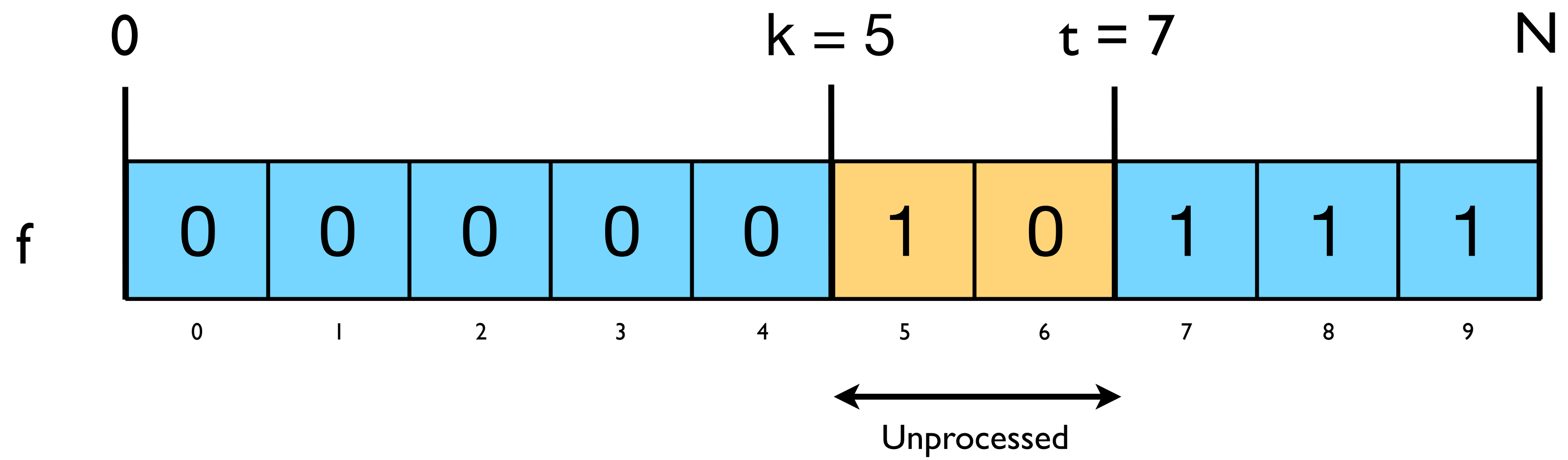
f.k = 0 => k:=k+1

Bound function t - k

7 - 4 = 3

Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.k allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



f.k = 1 => **f.k, f.t-1 := f.t-1, f.k**

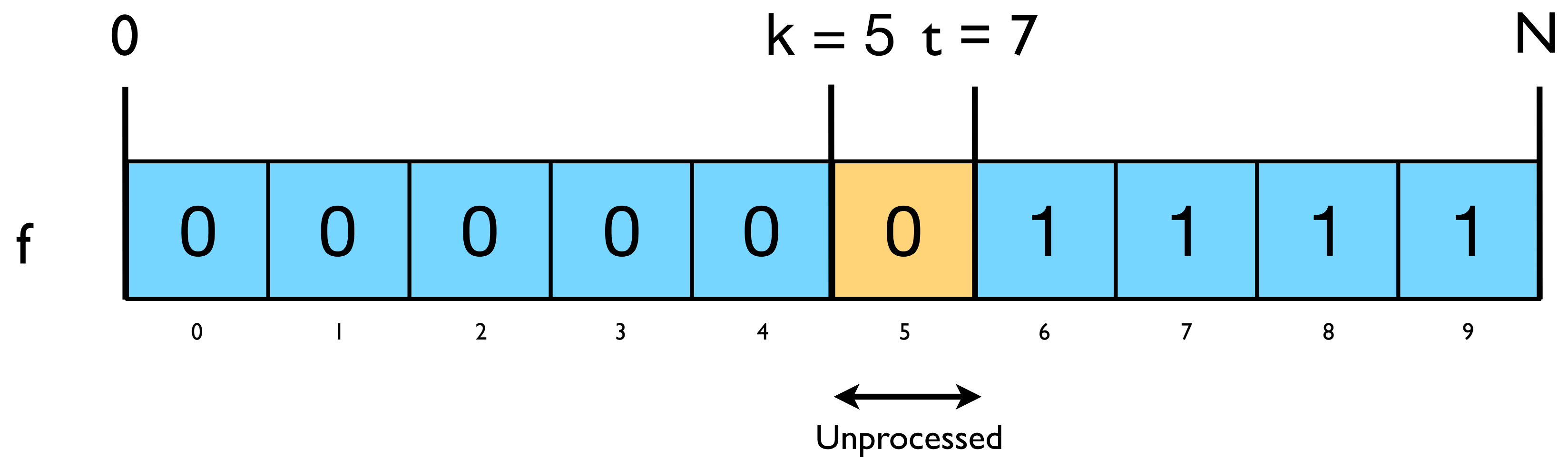
t := t-1

Bound function t - k

7 - 5 = 2

Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.k allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



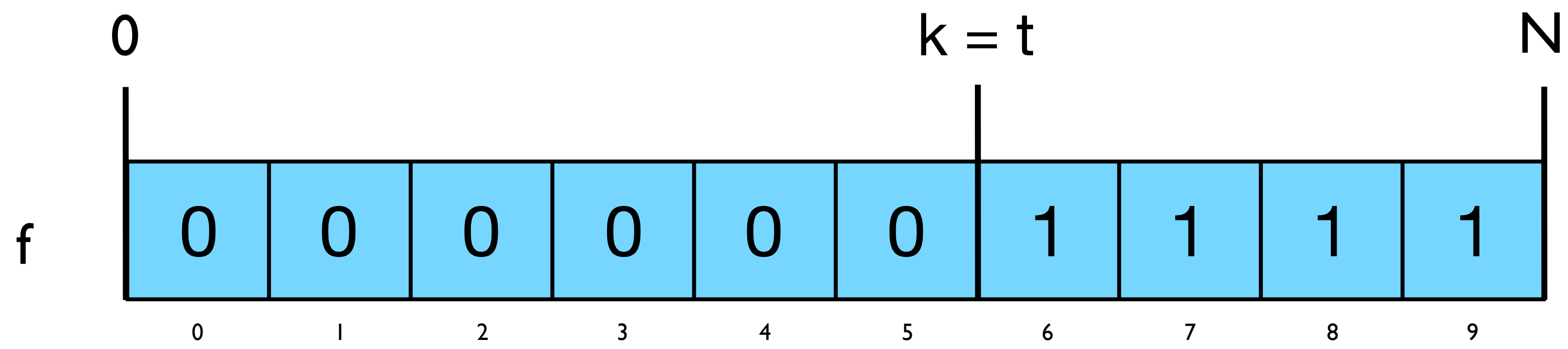
f.k = 0 => k:=k+1

Bound function t - k

6 - 5 = 1

Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.k allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



f.k = 0 => k:=k+1

Bound function t - k

6 - 6 = 0

Termination

For proof of loop termination we must take our bound function and show that our initial value for **k** and **t** will allow the loop to start and that our loop body operations will give us a decreasing function. This leads to three separate substitutions.

Initialisation

$(t - k > 0)\ (k, t := 0, N)$

$\equiv$ [Substitution]

$N - 0 > 0$

$\equiv$ [Arithmetic]

$N > 0$

$\equiv$ [$\Leftarrow$ Given $\{N > 0\}$]

TRUE

Initialisation will allow loop to execute.

k := k +1

$(t - k)\ (k := k + 1)$

$\equiv$ [Substitution]

$t - (k + 1)$

$\equiv$ [Arithmetic]

$t - k - 1$

$<$

$t - k$

Decreasing function. Unprocessed segment size will shrink.

t := t - 1

$(t - k)\ (t := t - 1)$

$\equiv$ [Substitution]

$(t - 1) - k$

$\equiv$ [Arithmetic]

$t - k - 1$

$<$

$t - k$

Decreasing function. Unprocessed segment size will shrink.

Complete Solution

```
|[ Con N: int {N ≥ 0}

  Var
       f: array [0..N) of int;
       {∀j: 0 ≤ j < N: f.j = 0 ∧ f.j = 1}


       k, t: int;
       k,t := 0, N;

        do k < t →
            if (f.k = 0) →
                k := k + 1;
            [] (f.k = 1) →
                f.k, f.t-1 := f.t-1, f.k;
                t := t -1;
            fi
        od

       {∃k: 0 ≤ k ≤ N: ∀j: 0 ≤ j < k: f.j = 0 ∧
                        ∀j: k ≤ j < N: f.j = 1}
]|
```

Three Segment Problem - Dutch National Flag

$$\{\exists k,t: 0 \leq k \leq t \leq N: \forall j: 0 \leq j < k: f.j = 0 \wedge$$
$$\forall j: k \leq j < t: f.j = 1 \wedge$$
$$\forall j: t \leq j < N: f.j = 2\}$$

Problem

Using an invariant diagram specify and, hence, derive an **O(N)** solution to the following: Given f[0..N) containing only 0's, 1's and 2's, sort it so that all 0's precede all 1's and all 1's precede all 2's. Only swap operations are allowed on f.

|[**Con** N: int $\{N > 0\}$

Var

f: array [0..N) of int;

$\{\forall j: 0 \leq j < N: f.j = 0 \vee f.j = 1 \vee f.j = 2\}$

k: int;

S

$$\{\exists k,t: 0 \leq k \leq t \leq N: \forall j: 0 \leq j < k: f.j = 0 \wedge$$
$$\forall j: k \leq j < t: f.j = 1 \wedge$$
$$\forall j: t \leq j < N: f.j = 2\}$$

]|

Specification

$|[$ **Con** $N: int \{N > 0\}$

Define the necessary constants. Most of this information will be specified in the problem description.

Var

f: array $[0..N)$ of int;

$\{\forall j: 0 \leq j < N: f.j = 0 \vee f.j = 1 \vee f.j = 2\}$

k: int;

Define the necessary variables. Notice that we have defined the array f as a variable because its contents will be changed as a result of the sorting.

Once we define the array we can write an assertion to say that it only contains 0's and 1's.

S

Program label

$$\{\exists k,t: 0 \leq k \leq t \leq N: \forall j: 0 \leq j < k: f.j = 0 \wedge$$
$$\forall j: k \leq j < t: f.j = 1 \wedge$$
$$\forall j: t \leq j < N: f.j = 2\}$$

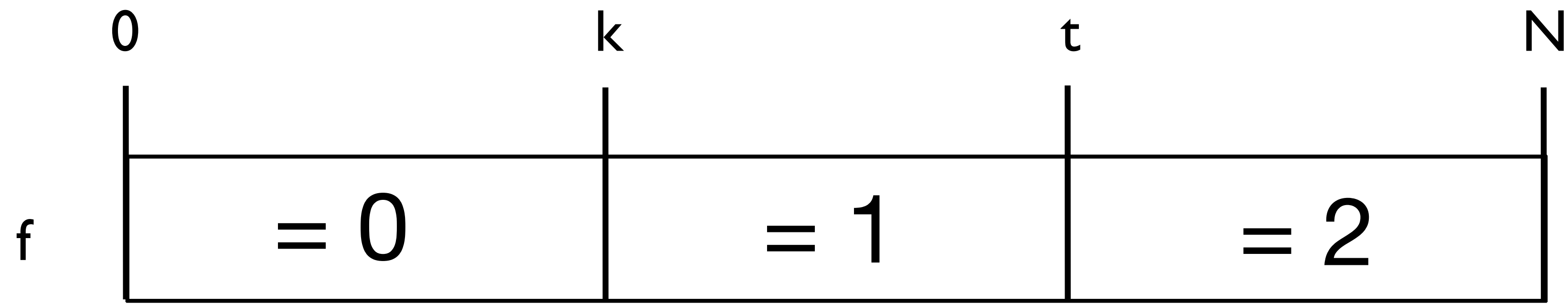
Write a postcondition that represents the problem

$]|$

Step 1: Postcondition Diagram

We need to draw a diagram that represents the state space after the program finishes. We do this so we can weaken the postcondition and add a unsorted/processed segment in the next step.

$\{\exists k,t: 0 \leq k \leq t \leq N: \forall j: 0 \leq j < k: f.j = 0 \wedge \forall j: k \leq j < t: f.j = 1 \wedge \forall j: t \leq j < N: f.j = 2\}$



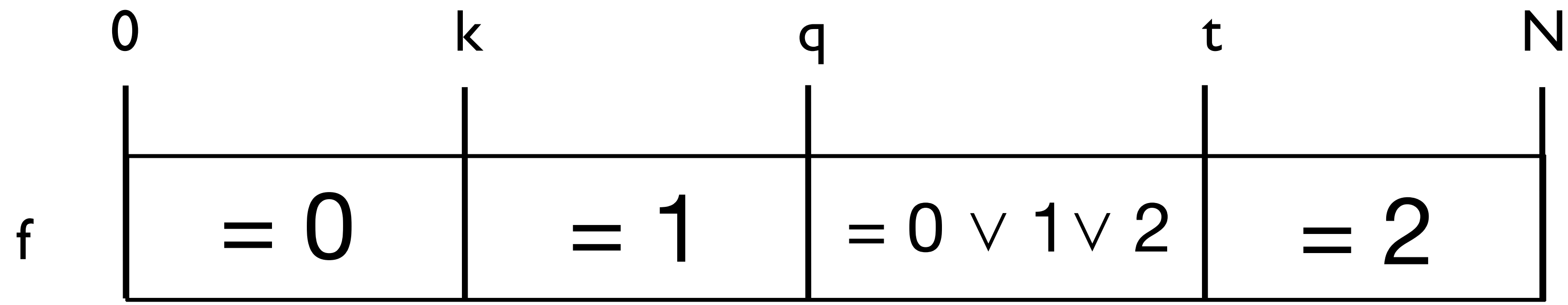
Notice that we have three distinct segments represented by **0 ≤ j < k** and **k ≤ j < t** and **t ≤ j < N**.

NOTE: this diagram represents an array that has been fully processed. In other words the size of the unprocessed segment is zero.

Step 2: Invariant Diagram

Next we need to draw a diagram that represents a snap shot during the middle of processing. This is actually an invariant diagram as it introduces an unsorted segment and will hold true at the **start, middle** and **end** of processing.

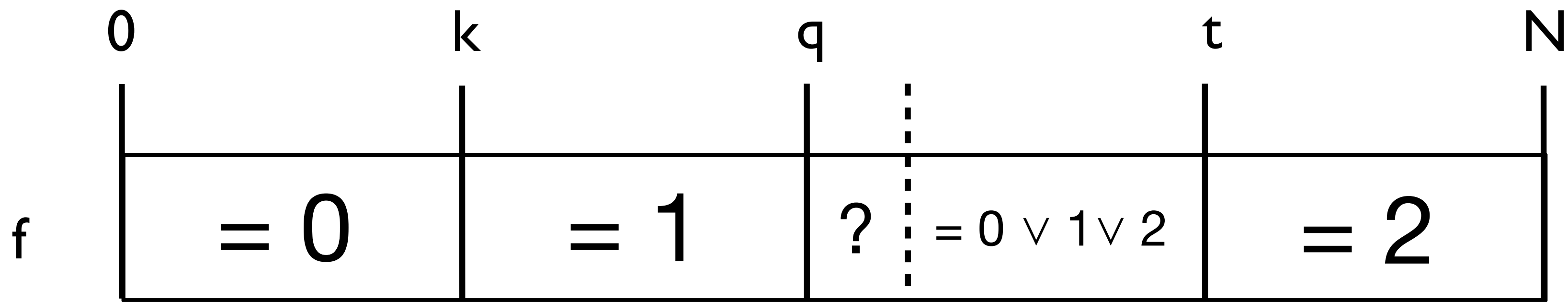
$$\{\exists k,t,q: 0 \le k \le q \le t \le N: \forall j: 0 \le j < k: f.j = 0 \wedge \forall j: k \le j < q: f.j = 1 \wedge$$
$$\boldsymbol{\forall j: q \le j < t: f.j = 0} \vee \boldsymbol{f.j = 1} \vee \boldsymbol{f.j = 2} \wedge \forall j: t \le j < N: f.j = 2\}$$



Notice that we now have a segment of unsorted array elements represented by
$\boldsymbol{\forall j: q \le j < t: f.j = 0} \vee \boldsymbol{f.j = 1} \vee \boldsymbol{f.j = 2}$.

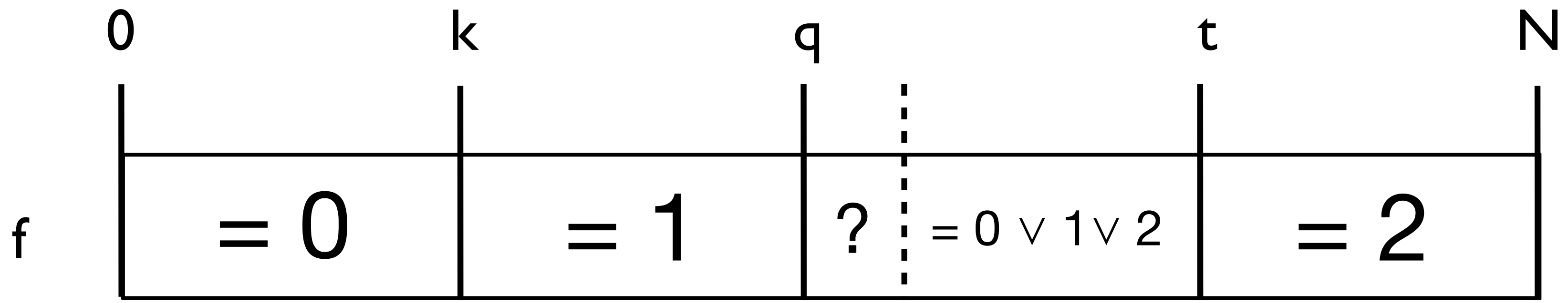
Step 3: Derive O(N) Solution

Using the invariant diagram, we consider the state space at the **start**, **middle** and **end** of processing. We consider the start and end states first because this gives us our variable initialisation and the guard for the loop.

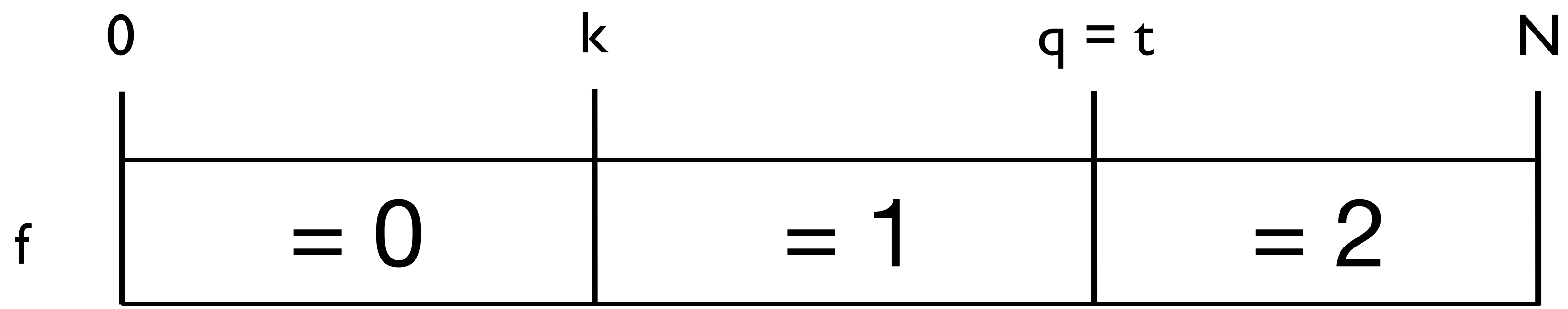


__Start of processing__
f[q..t) represents the whole array. f[0..q) and f[t..N) are both empty. In other words **k,q,t : = 0,0,N.**

Step 3: Derive O(N) Solution

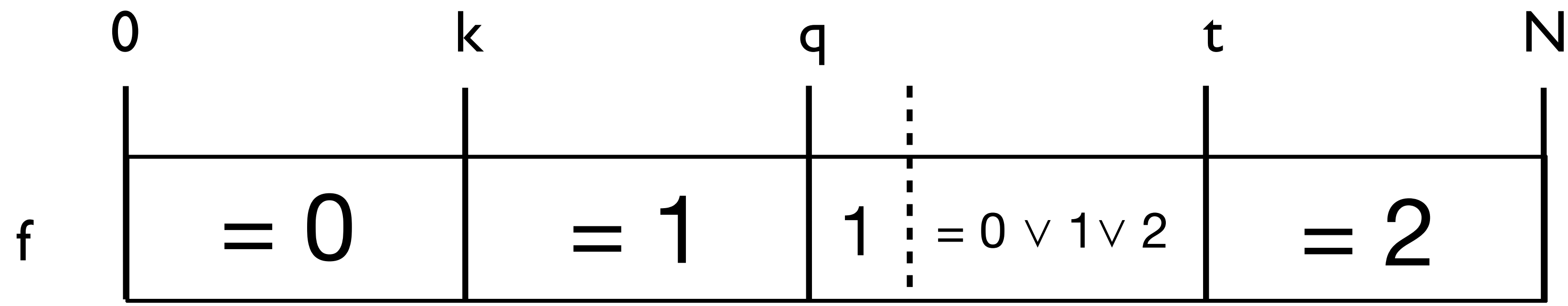


__End of processing__
f[q..t) will be empty, that is, q = t. Therefore, the guard on the loop is **q < t**.



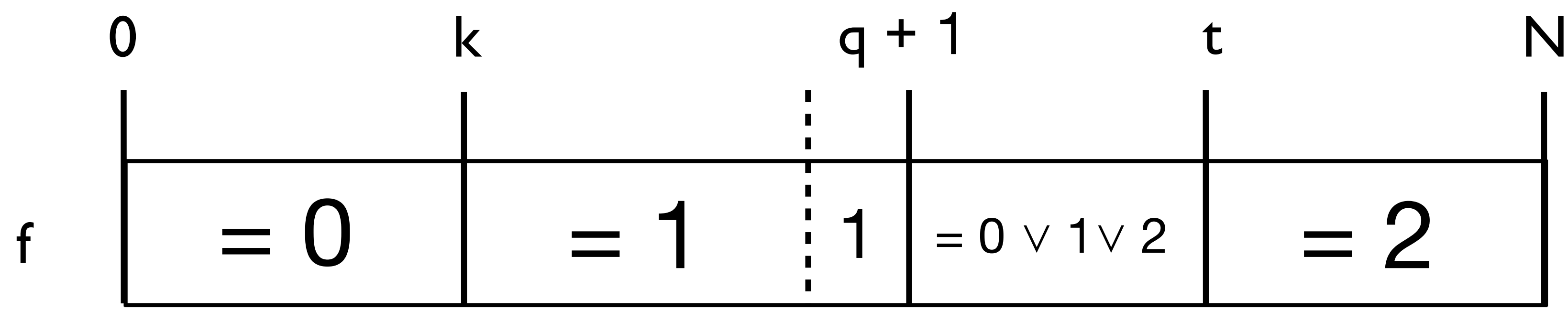
Step 3: Derive O(N) Solution

For the middle of processing, the loop body, we will focus on f.q and consider the possible cases. In this problem there are exactly three possibilities, **f.q = 0** $\vee$ **f.q = 1** $\vee$ **f.q = 2**.

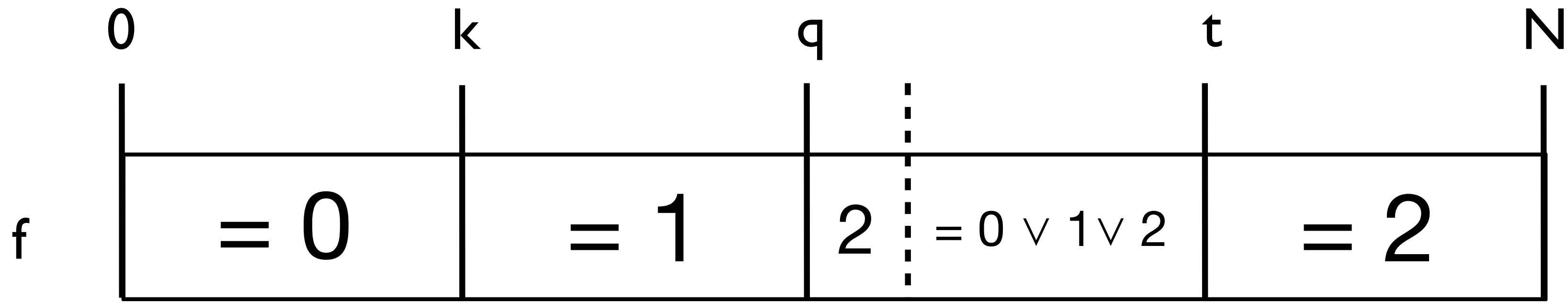


<u>Middle of processing</u>

f.q = 1 => Simply increase q by 1
q := q + 1

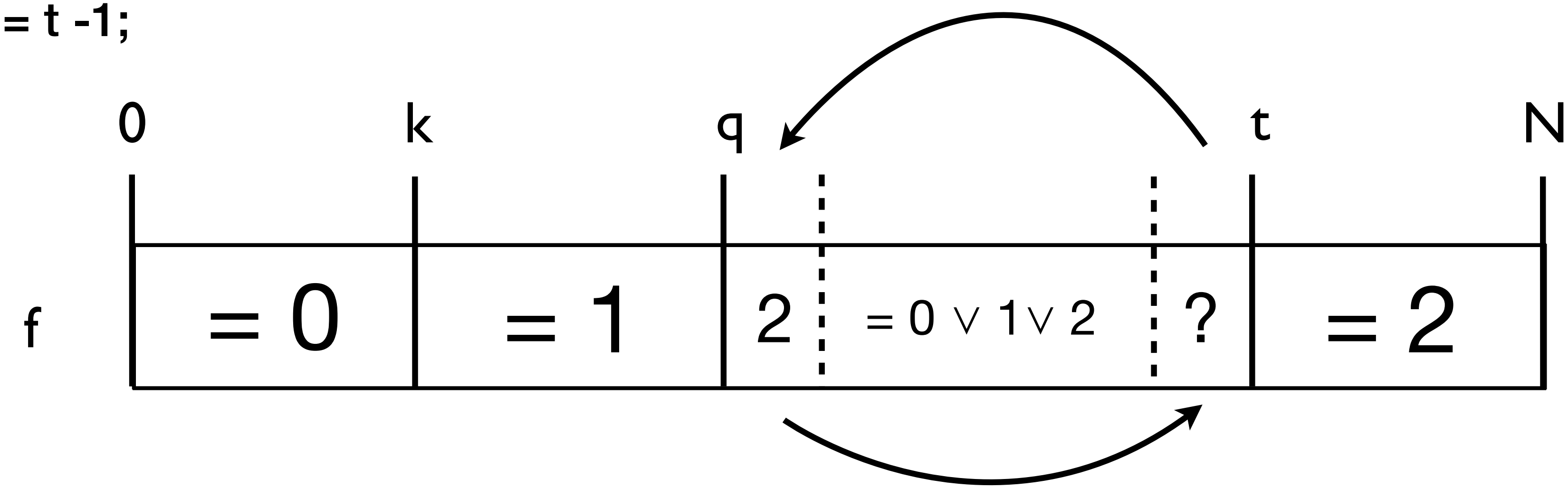


Step 3: Derive O(N) Solution

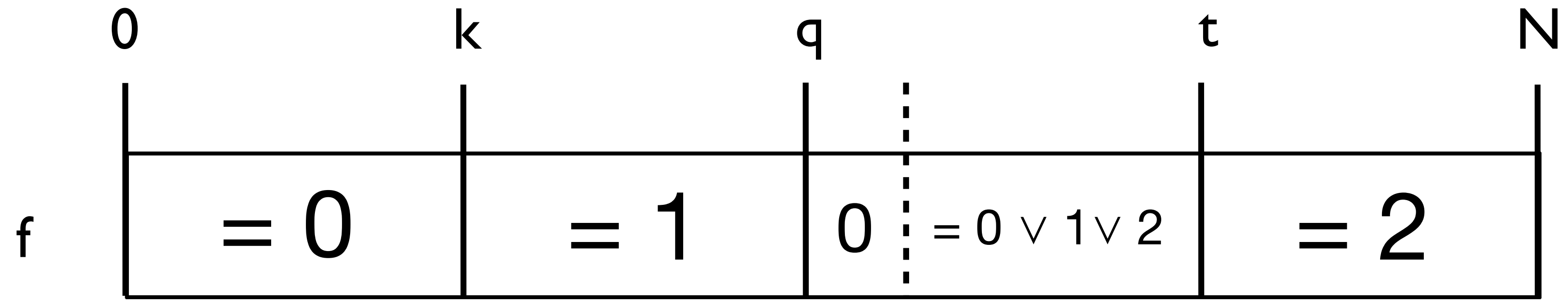


Middle of processing

f.q = 2 => Swap f.q with f.t – 1 and decrement t.
  f.q,f.t – 1 := f.t-1, f.q;
  t := t -1;



Step 3: Derive O(N) Solution

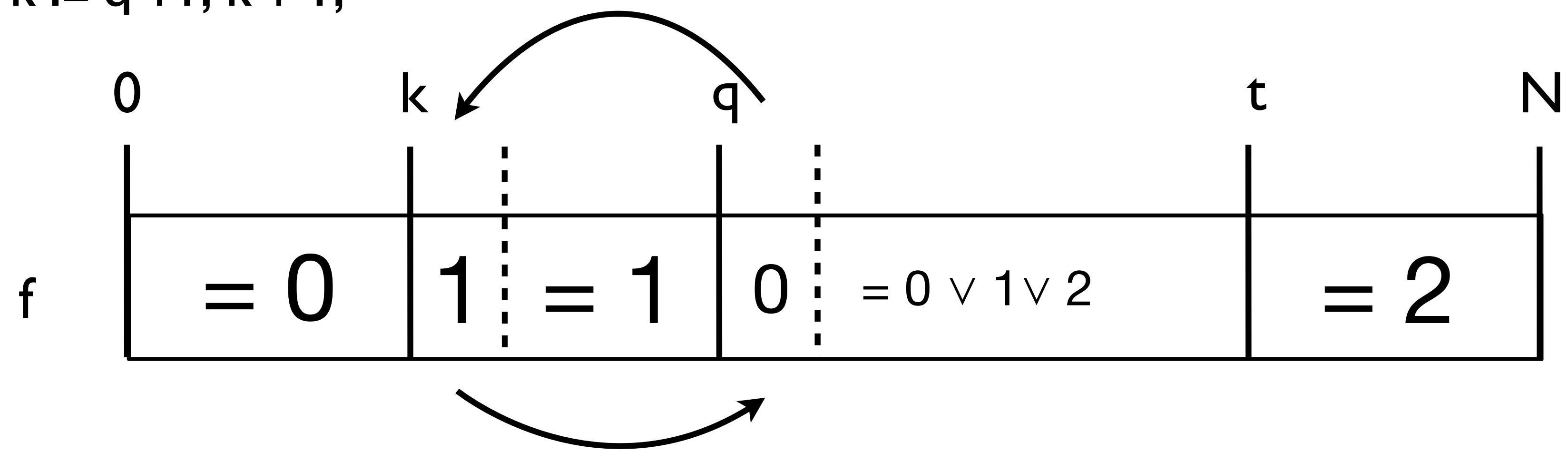


Middle of processing

f.q = 0 => Swap f.q with f.k and increment k and q.

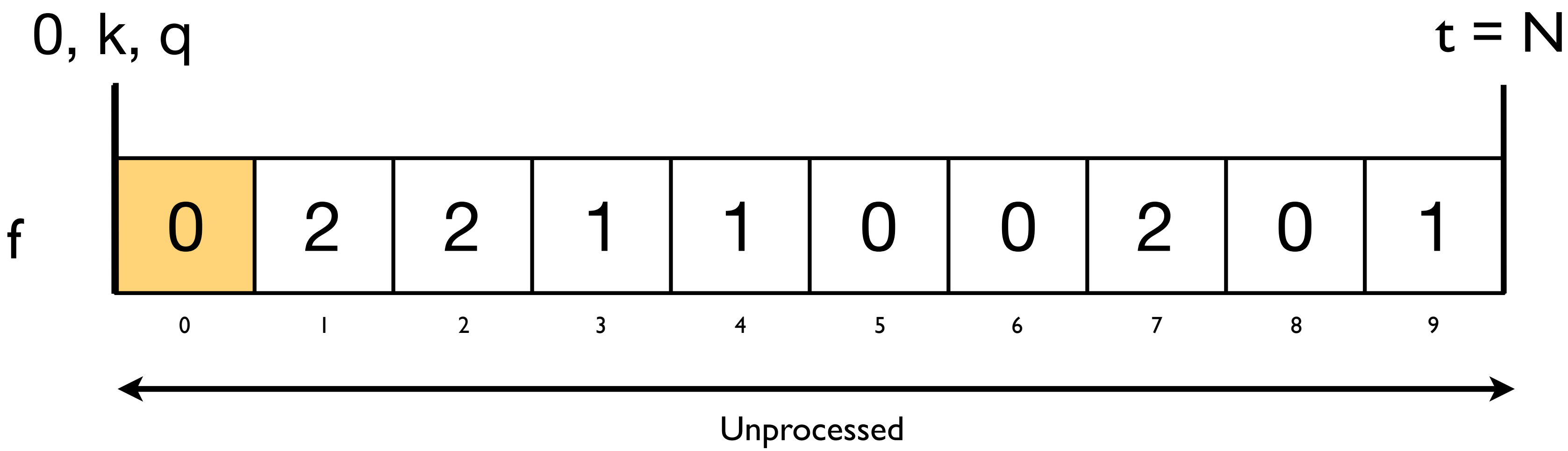
f.q,f.k := f.k, f.q;

q, k := q +1, k + 1;



Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.q allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



f.q = 0 => f.q, f.k := f.k, f.q

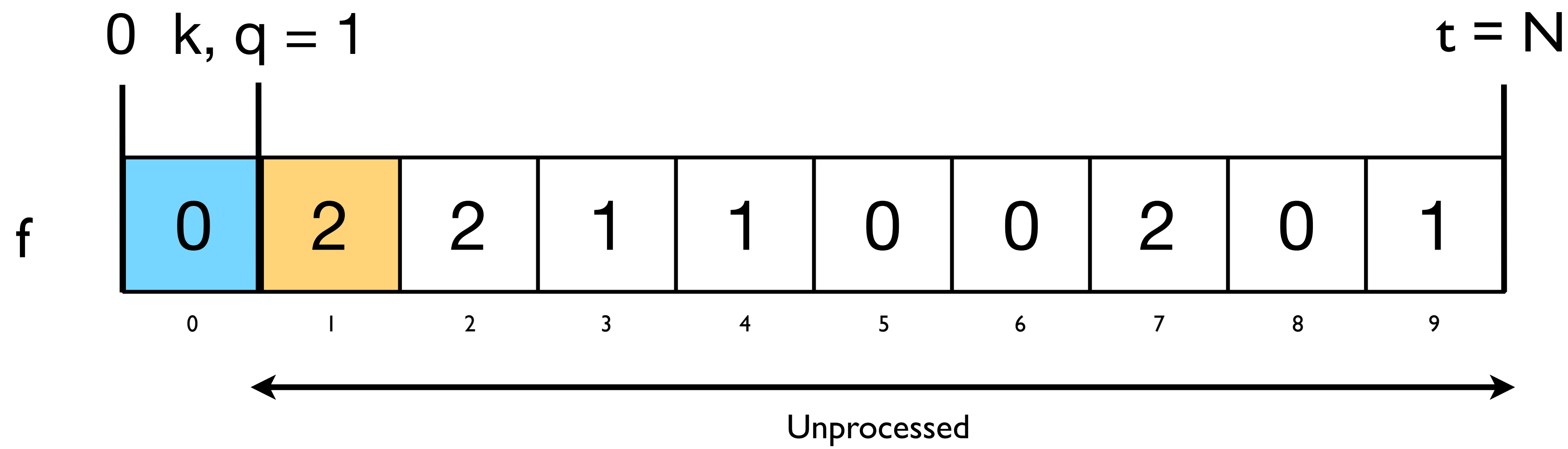
q, k := q+1, k+1

Bound function t - q

t - 0 = 10

Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.q allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



f.q = 2 => f.q, f.t-1 := f.t-1, f.q

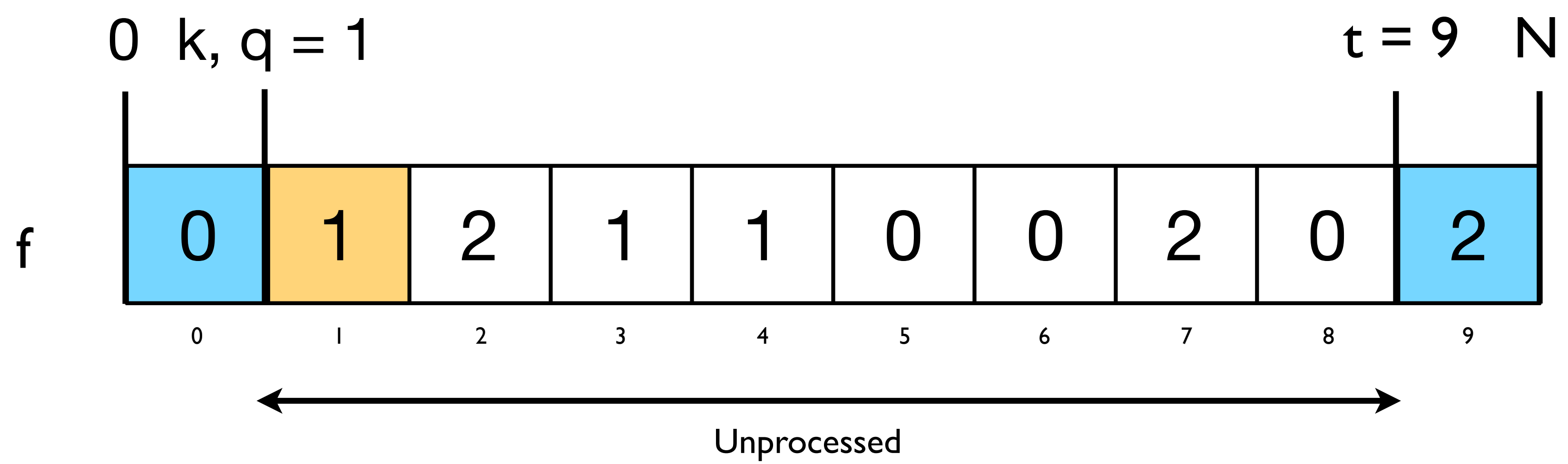
t := t -1

Bound function t - q

t - 1 = 9

Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.q allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



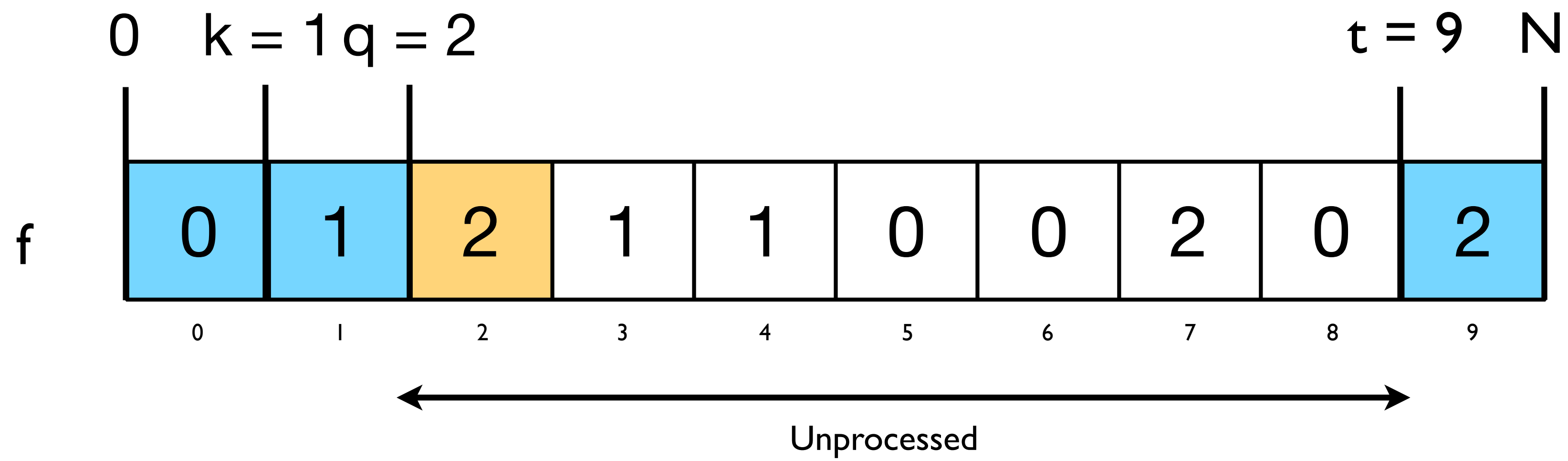
f.q = 1 => q := q + 1

Bound function t - q

t - 1 = 8

Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.q allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



f.q = 2 => f.q, f.t-1 := f.t-1, f.q

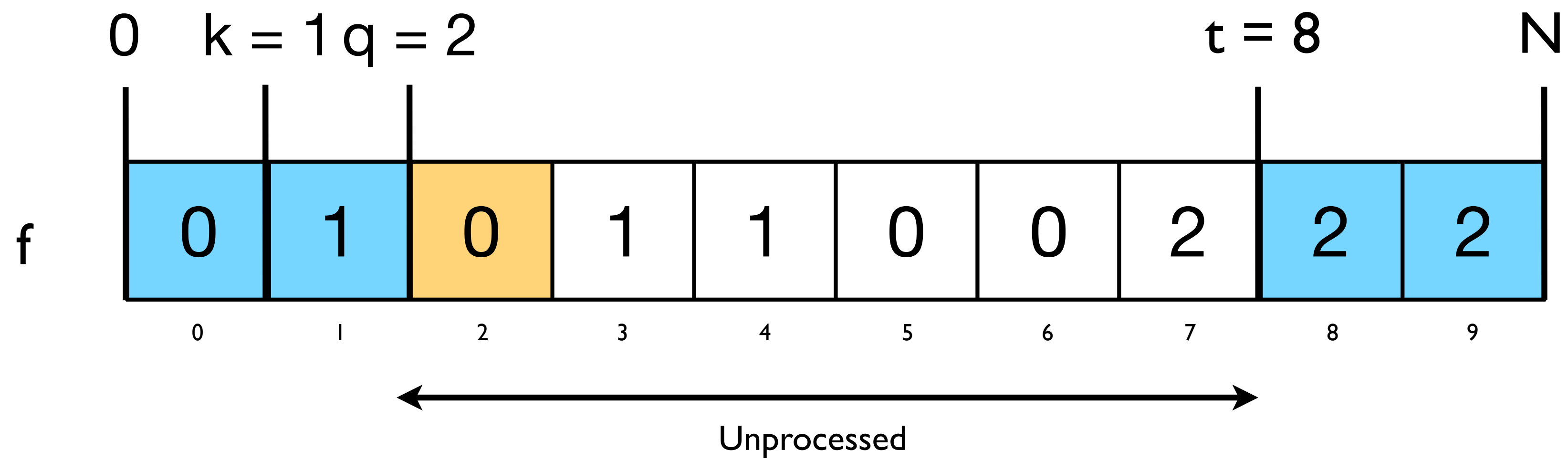
t := t -1

Bound function t - q

t - 2 = 7

Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.q allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



f.q = 0 => f.q, f.k := f.k, f.q

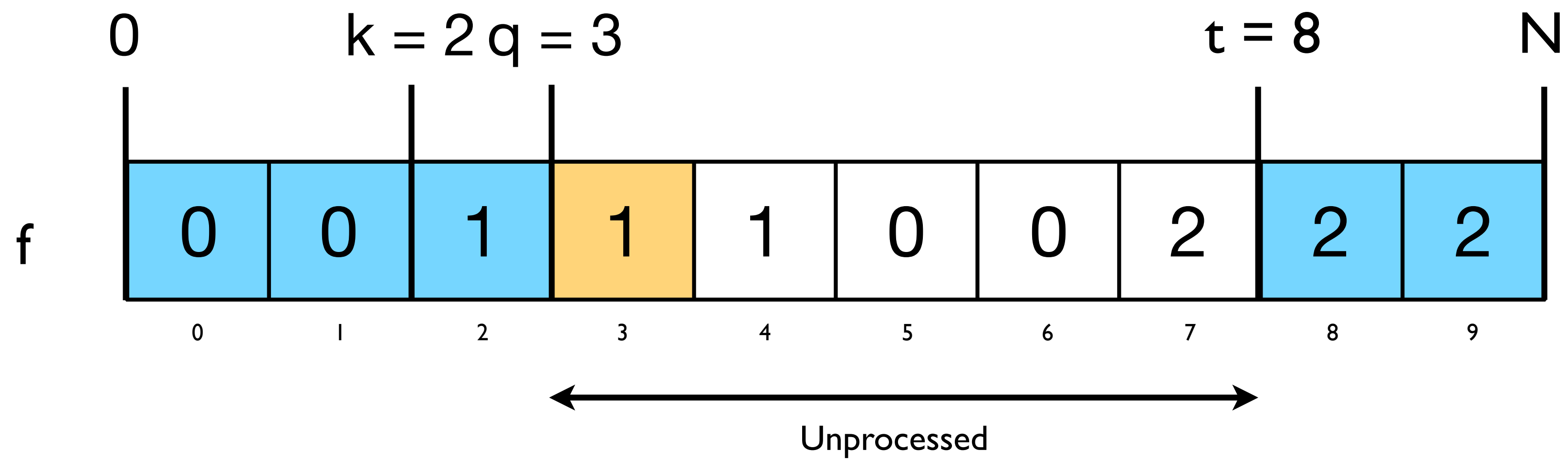
k, q := k + 1, q + 1

Bound function t - q

t - 2 = 6

Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.q allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



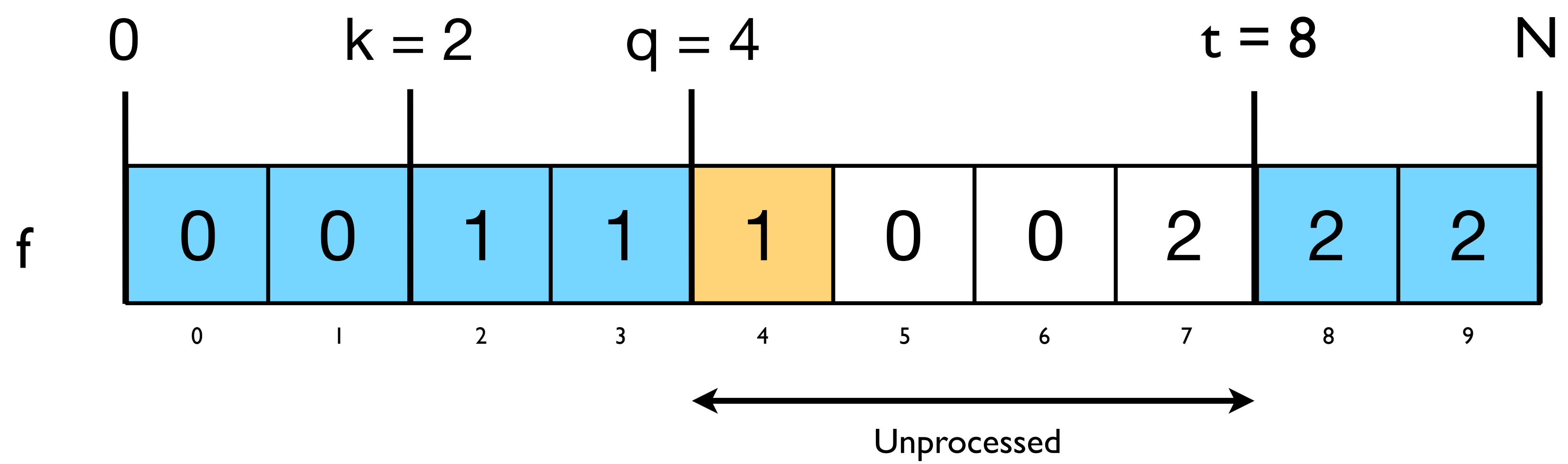
f.q = 1 => q := q + 1

Bound function t - q

t - 3 = 5

Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.q allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



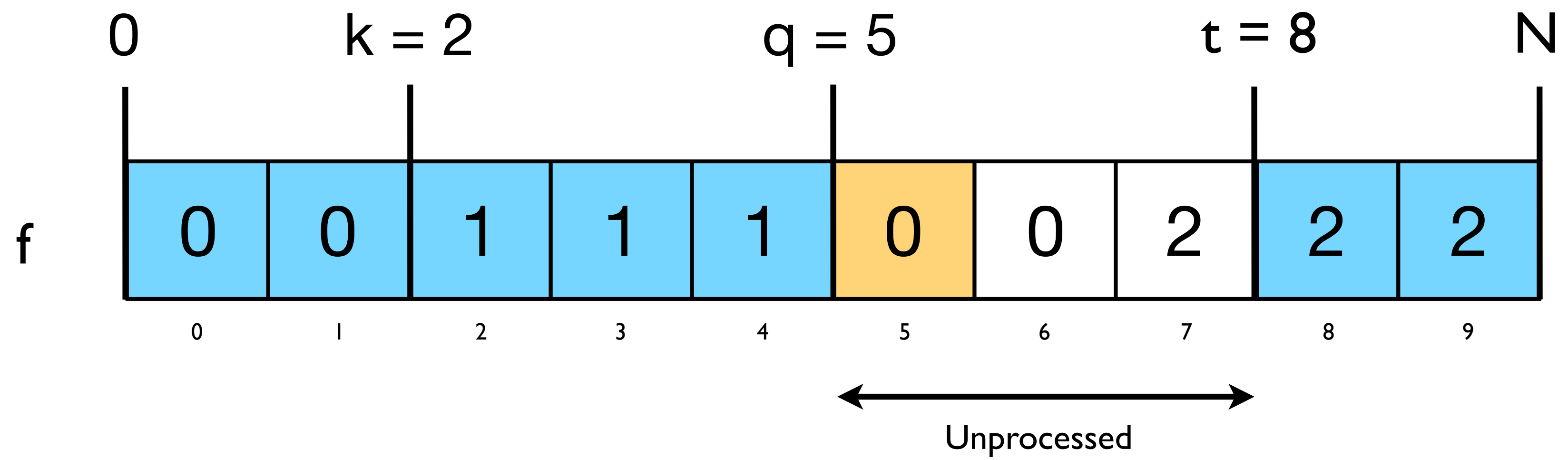
f.q = 1 => q := q + 1

Bound function t - q

t - 4 = 4

Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.q allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



f.q = 0 => f.q, f.k := f.k, f.q

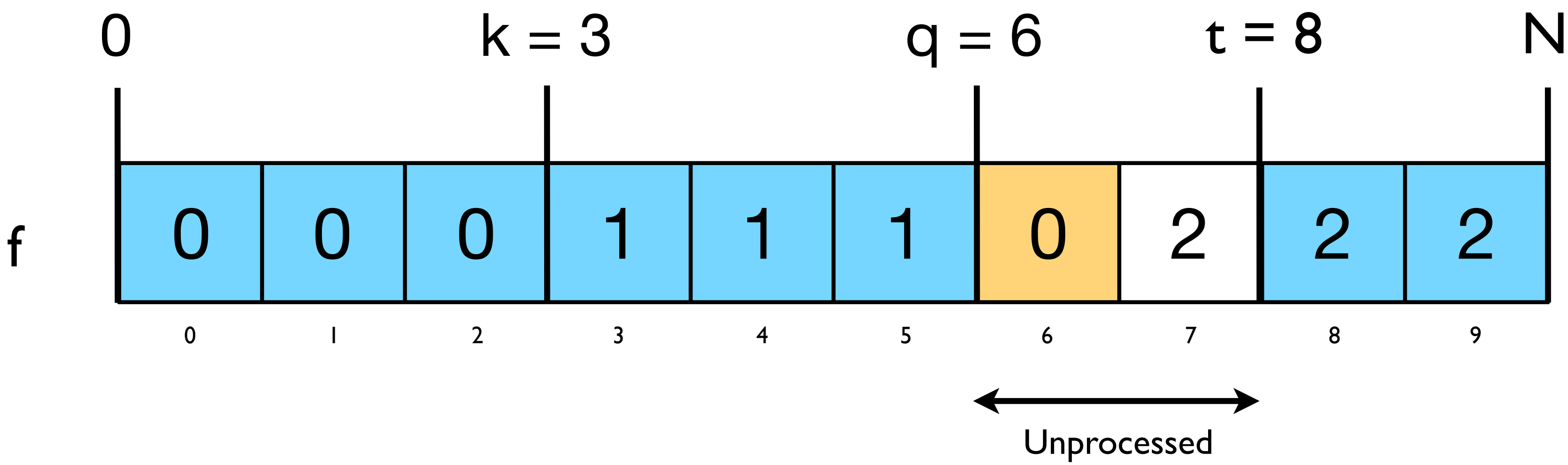
k, q := k + 1, q + 1

Bound function t - q

t - 5 = 3

Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.q allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



f.q = 0 => f.q, f.k := f.k, f.q

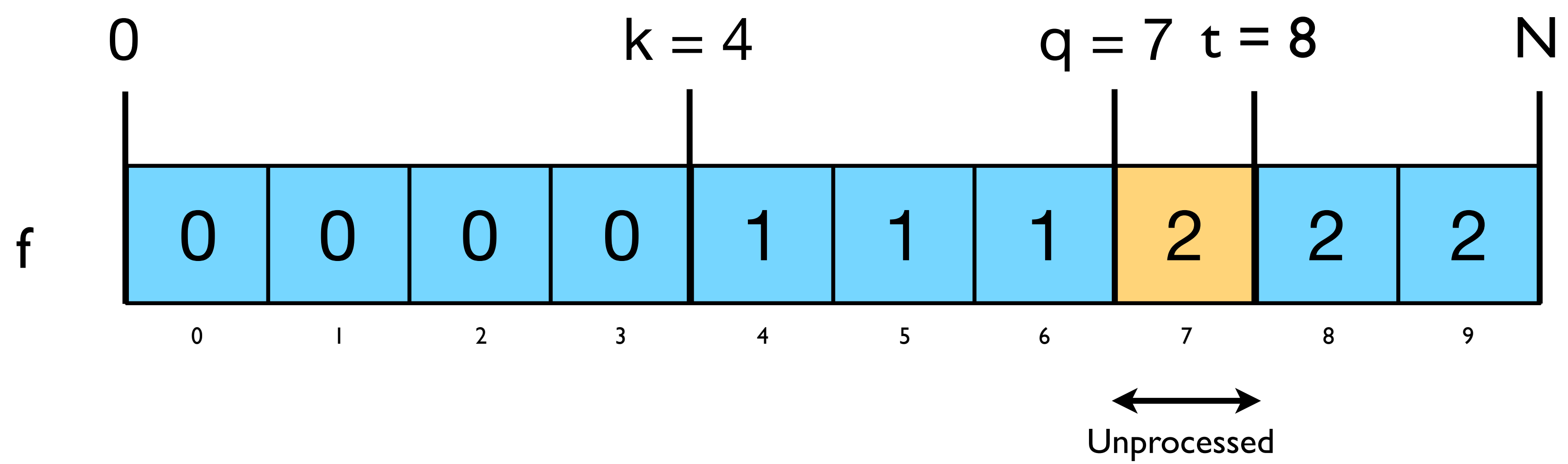
k, q := k + 1, q + 1

Bound function t - q

t - 6 = 2

Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.q allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



f.q = 2 => **f.q, f.t-1 := f.t-1, f.q**

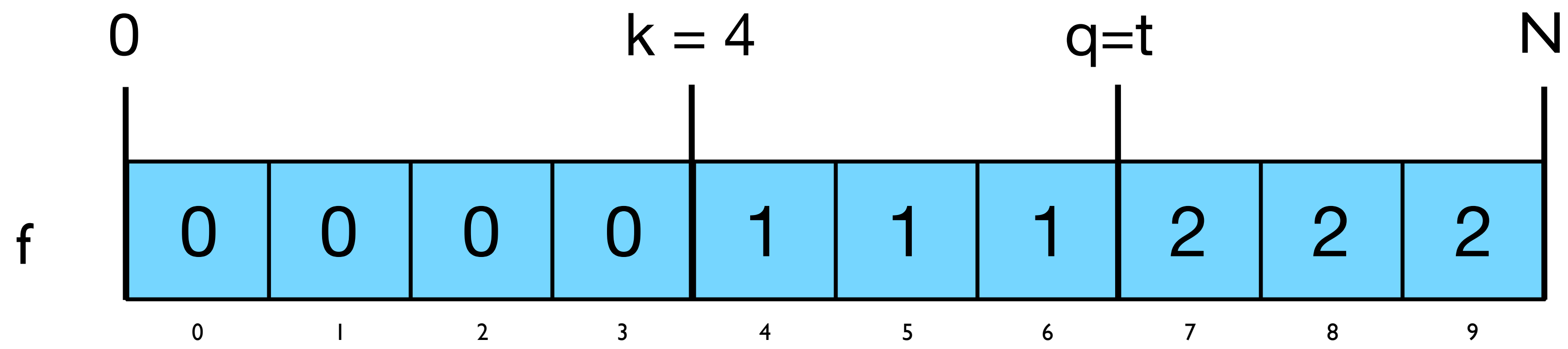
t := t - 1

Bound function t - q

t - 7 = 1

Middle of processing; a closer look

Stepping through the program we can see how the choices we make when processing f.q allow us to sort the array and to ensure we make progress by reducing the unsorted segment.



Bound function t - q

t - 7 = 0

Termination

For proof of loop termination we must take our bound function and show that our initial value for **k** and **t** will allow the loop to start and that our loop body operations will give us a decreasing function. This leads to three separate substitutions.

Initialisation

$(t - q > 0)\ (q\ t := 0, N)$

≡ [Substitution]

$N - 0 > 0$

≡ [Arithmetic]

$N > 0$

≡ [⇐ Given {N > 0}]

TRUE

Initialisation will allow loop to execute.

q := q +1

$(t - q)\ (q := q + 1)$

≡ [Substitution]

$t - (q + 1)$

≡ [Arithmetic]

$t - q - 1$

$<$

$t - q$

q := q +1

$(t - q)\ (q := q + 1)$

≡ [Substitution]

$t - (q + 1)$

≡ [Arithmetic]

$t - q - 1$

$<$

$t - q$

Decreasing function. Unprocessed segment size will shrink.

t := t - 1

$(t - k)\ (t := t - 1)$

≡ [Substitution]

$(t - 1) - k$

≡ [Arithmetic]

$t - k - 1$

$<$

$t - k$

Decreasing function. Unprocessed segment size will shrink.

Complete Solution

```
|[ Con N: int {N > 0}

  Var
     f: array [0..N) of int;
     {∀j: 0 ≤ j < N: f.j = 0 ∨ f.j = 1 ∨ f.j = 2}
     k,q,t: int;

     k,q,t := 0,0,N;

    do q < t →
        if f.q = 0 →
            f.q, f.k := f.k, f.q;
            q,k := q+1, k+1;
        [] f.q = 1 →
            q := q+1;
        [] f.q = 2 →
            f.q, ft-1 := f.t-1, f.q;
            t := t-1;
    od

    {∃k,t: 0 ≤ k ≤  t ≤ N: ∀j: 0 ≤ j < k: f.j = 0 ∧
                           ∀j: k ≤ j < t: f.j = 1 ∧
                           ∀j: t ≤ j< N: f.j = 2}
]|
```