Terminaison et correction des algorithmes

MPSI/PCSI

Lycée Jean Perrin, Lyon

2022–2023

Logarithme en base 10

```
def entlog10(x):
    n=0
    while 10**n<=x:
        n+=1
    return n-1
```

Logarithme en base 10 : terminaison et correction

- Le programme termine :
  Si n_0 est le plus petit entier tel que $10^{n_0} > x$, $n_0 - n$ est un variant de boucle

- Le programme n'est pas faiblement correct :
  Si $x < 1$:

Logarithme en base 10 : terminaison et correction

- **Préconditions :** $x > 0$

- **Post-conditions :**
  $(x \geq 1,\ 10^{n-1} \leq x$ et $x < 10^n)$ ou
  $(x < 1,\ 10^n \leq x$ et $10^{n+1} > x)$.

Logarithme en base 10 : nouvelle version

```
def entlog10v2(x):
    n=0
    assert x>0
    if x>=1:
            while 10**n<=x:
                    n+=1
            return n-1
    else:
            while 10**n>x:
                    n-=1
            return n
```

Logarithme en base 10 : terminaison et correction

- Si $x \geq 1$:

- Sinon : si n_0 désigne le plus grand entier relatif tel que $10^{n_0} \leq x$...

Recherche d'un élément dans une liste

```
def recherche1(L,elt):
    trouve=False
    for k in range(len(L)):
        if L[k]==elt:
            trouve=True
    return trouve
```

Terminaison et correction

- Terminaison :

- Correction :
  Invariant : $P(k)$: $Trouve_k = True \Leftrightarrow \exists i \in [\![0; k-1]\!]$ tel que $T[i] = elt$

Recherche d'un élément dans une liste, version 2

```
def recherche2(L,elt):
    trouve=False
    k=0
    while trouve==False and k<len(L):
        if L[k]==elt:
            trouve=True
        k+=1
    return trouve
```

```
def recherche2b(L,elt):
    k=0
    while k<len(L) and L[k]!=elt:
        k+=1
    trouve = (k<len(L))
    return trouve
```

Recherche d'un élément dans une liste, version 2

```
def recherche2t(L,elt):
    trouve = False
    k=0
    while k<len(L) and not(trouve):
        trouve = (L[k]=elt)
        k+=1
    return trouve
```

Terminaison et correction

- Terminaison :

- Correction :
  Invariant : $P(k)$: $Trouve_k = True \Leftrightarrow \exists i \in [\![0; k-1]\!]$ tel que $T[i] = elt$

Problème du drapeau hollandais

```
def drapeau(L,p):
    n=len(L)
    i,j,k=0,0,n # Valeurs initiales de i, j et k
    while j<k:
        elt=L[j] # Compare le premier element non teste a elt
        if elt<p: # S'il est plus petit, on classe dans la deuxi
            L[j],L[i]= L[i],elt
            j+=1
            i+=1
        elif elt==p: # S'il est egal, deuxieme partie
            j+=1
        else:
            L[j],L[k-1]=L[k-1],elt # Sinon, partie finale
            k-=1
    return L
```

Drapeau hollandais : terminaison et correction

- Terminaison :
  Variant :

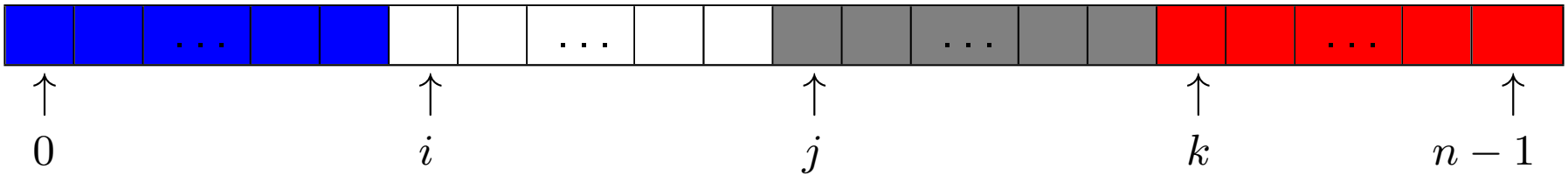
- Correction :
  Post-condition : il existe des entiers i et j tels que :
  - $0 \leq i \leq j \leq n-1$
  - Pour tout entier l compris entre 0 et $i-1$, $L[l] < p$.
  - Pour tout entier l compris entre i et $j-1$, $L[l] = p$.
  - Pour tout entier l compris entre j et $n-1$, $L[l] > p$.

Drapeau hollandais : terminaison et correction

- Correction : **Invariant :** il existe des entiers i, j et k tels que :
  - $0 \leq i \leq j \leq k \leq n-1$
  - Pour tout entier l compris entre 0 et $i-1$, $L[l] < p$.
  - Pour tout entier l compris entre i et $j-1$, $L[l] = p$.
  - Pour tout entier l compris entre k et $n-1$, $L[l] > p$.

Drapeau hollandais : terminaison et correction

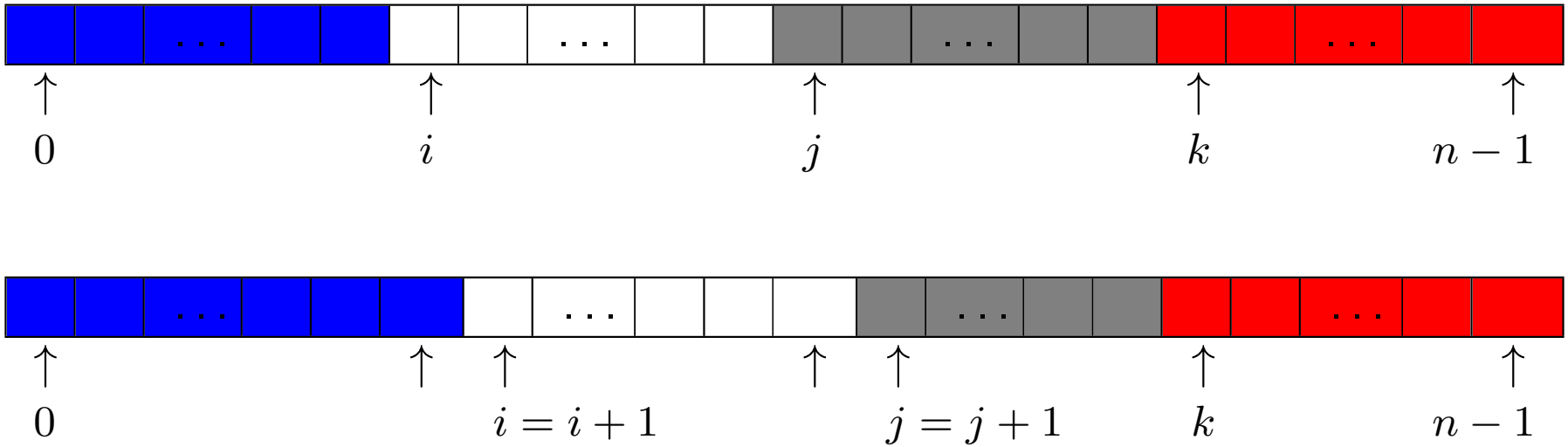
Avant le premier passage dans la boucle, on a $i = 0$, $j = 0$ et $k = n$, donc toutes les conditions sont vérifiées.



Drapeau hollandais : terminaison et correction

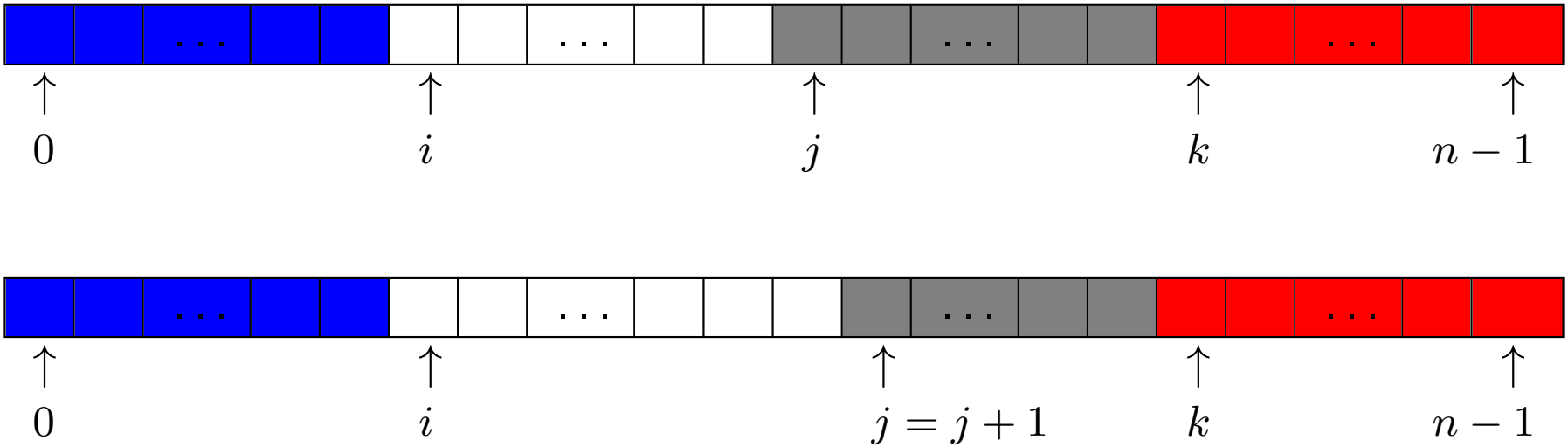
Supposons que toutes les conditions sont vérifiées en début de boucle.

Si $L[j] < p$ à la ligne 6, $L[i]$ se voit affecter la valeur de $L[j]$, et $L[j]$ se voit affecter la valeur précédente de $L[i]$, qui était égale à p d'après l'invariant de boucle. De plus, i et j sont augmentés de 1, donc les conditions 2 et 3 de l'invariant restent vérifiées. Comme $k - j > 0$ au début de la boucle, on a de plus toujours $0 \leq i \leq j \leq k \leq n - 1$.



Drapeau hollandais : terminaison et correction

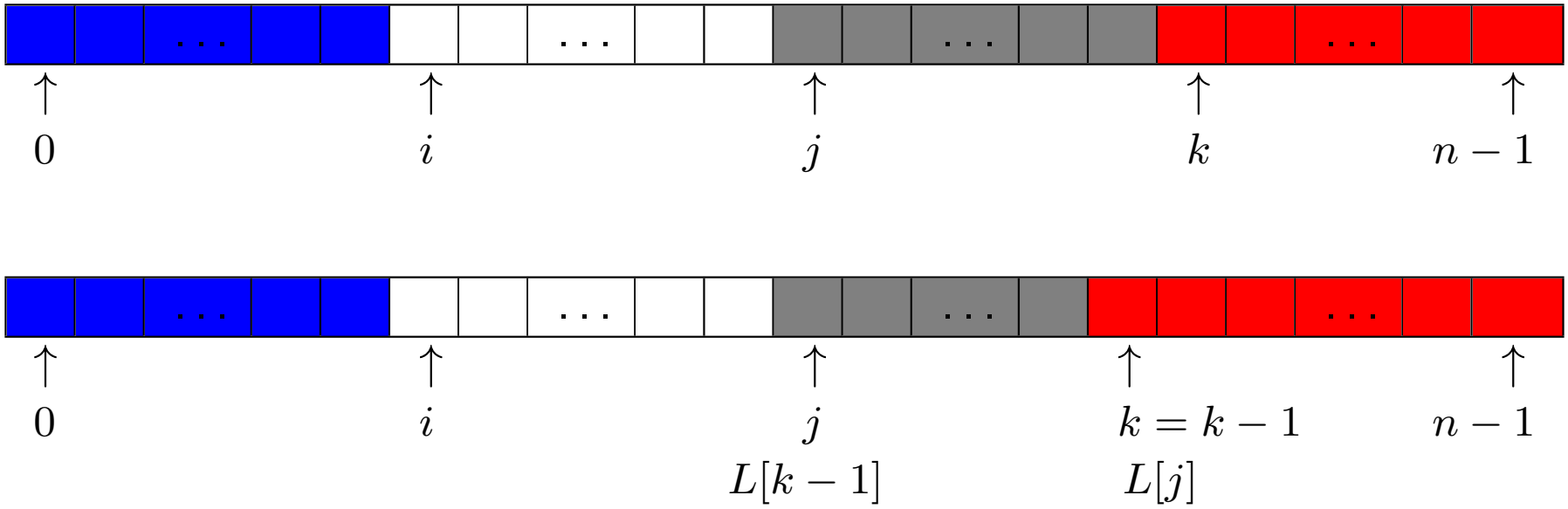
Sinon, si $L[j] = p$, aucun élément n'est modifié et j est augmenté de 1, donc la troisième condition de l'invariant reste vérifiée pour la nouvelle valeur de j. Comme $k - j > 0$ au début de la boucle, on a toujours $0 \leq i \leq j \leq k \leq n - 1$.



Drapeau hollandais : terminaison et correction

Enfin, si $L[j] > p$, $L[k-1]$ se voit affecter la valeur de $L[j]$ qui était strictement supérieure à p, $L[j]$ se voit affecter la valeur précédente de $L[k-1]$ et k diminue de 1, donc la quatrième condition de l'invariant reste vérifiée pour la nouvelle valeur de j. Comme $k - j > 0$ au début de la boucle, on a toujours $0 \leq i \leq j \leq k \leq n-1$,

L'invariant reste donc vérifié en fin de boucle.



Drapeau hollandais : terminaison et correction

Lorsque le programme s'arrête, on a $k - j = 0$ d'après la condition d'arrêt donc $j = k$. Les deux dernières conditions de l'invariant deviennent identiques, et la post-condition est vérifiée.

Drapeau hollandais : nombres de tests et d'échanges

Soit n la taille du tableau.

Tous les éléments du tableau sont testés à tour de rôle, on effectue donc n tests.

Le cas le plus favorable est celui où on ne fait aucun échange, par exemple si tous les éléments sont "blancs".

Le cas le plus défavorable est celui où on fait après chaque test deux échanges, par exemple si tous les éléments sont "rouge" : on effectue dans ce cas n échanges.

Remarque : la complexité en moyenne est difficile à évaluer car elle dépend de la répartition initiale des valeurs de la liste. Si on fait l'hypothèse que chaque élément, au moment où on l'évalue, à une chance sur trois d'être dans chacune des trois catégories, on a deux cas où on doit réaliser deux échanges, et un cas où on n'en réalise pas : le nombre moyen d'échanges dans ce cas est donc environ égal à $\frac{2n}{3}$.