Chapitre 9 : Tris

1) Tri par sélection
2) Tri par insertion
3) Tri rapide (récursif)
4) Principe du Diviser pour Regner (rappel)
5) Tri fusion (diviser pour régner)
6) "Master Theorem" pour résoudre des récurrences obtenues en diviser pour régner

Tris

- Un **algorithme de tri** est un algorithme qui prend en entrée **un tableau ou une liste de nombres, et qui trie les nombres du tableau ou de la liste (dans l'ordre croissant,** sauf mention explicite du contraire).
- Les algorithmes de tris sont parmi les algorithmes **les plus utiles**, et aussi parmi les exercices d'algorithmiques **les plus courants.**
- Il existe 4 (voire 6) façons différentes très connues d'effectuer le tri.

Tris

Nous allons commencer par les deux tris considérés comme **les plus simples**, car ils correspondent vraiment aux façons les plus naturelles de **trier "à la main".**

- **Tri par sélection**
- **Tri par insertion**

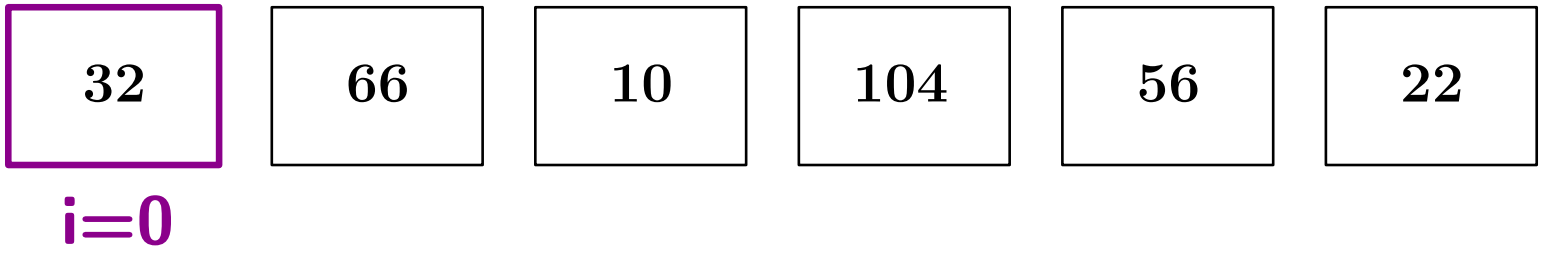
Pour illustrer nos algorithmes, nous allons imaginer que notre but est de trier une ensemble de cartes étudiant dans l'ordre croissant de numéro d'étudiant.

1) Tri par sélection

Début: imaginer que toutes les cartes étudiant sont étalées sur la table, alignées.

Principe: Je veux remplir correctement la case i=0

- Je parcours du regard l'ensemble des cartes une fois, de gauche à droite, pour **trouver le plus petit numéro**.
- Je prend dans ma main la carte de plus petit numéro, et je **la place tout devant** : cette carte ne **bougera plus**.

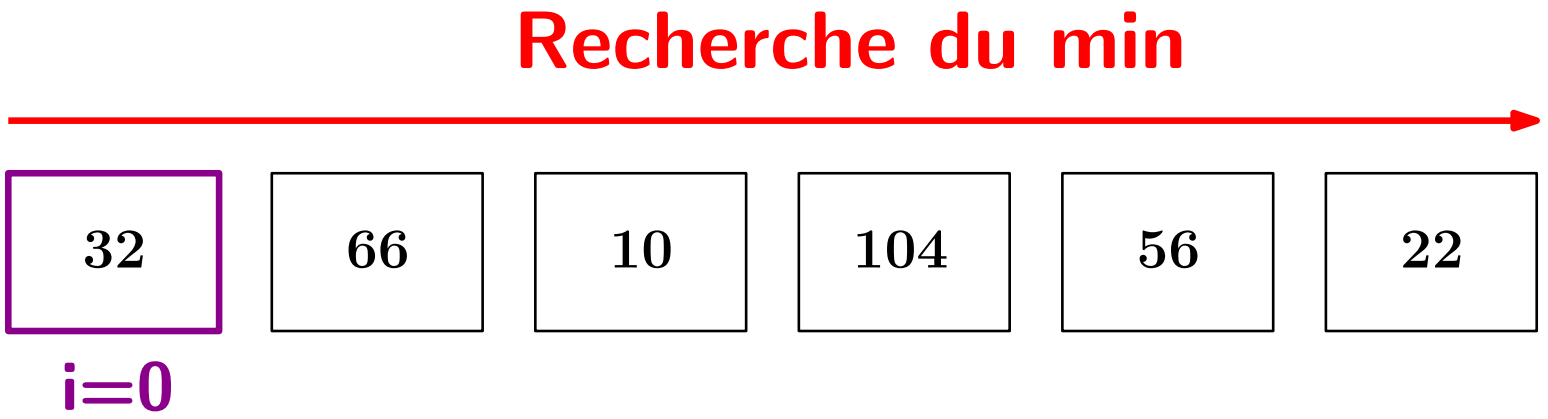


1) Tri par sélection

Début: imaginer que toutes les cartes étudiant sont étalées sur la table, alignées.

Principe: Je veux remplir correctement la case i=0

- Je parcourt du regard l'ensemble des cartes une fois, de gauche à droite, pour **trouver le plus petit numéro**.
- Je prend dans ma main la carte de plus petit numéro, et je **la place tout devant** : cette carte ne **bougera plus**.

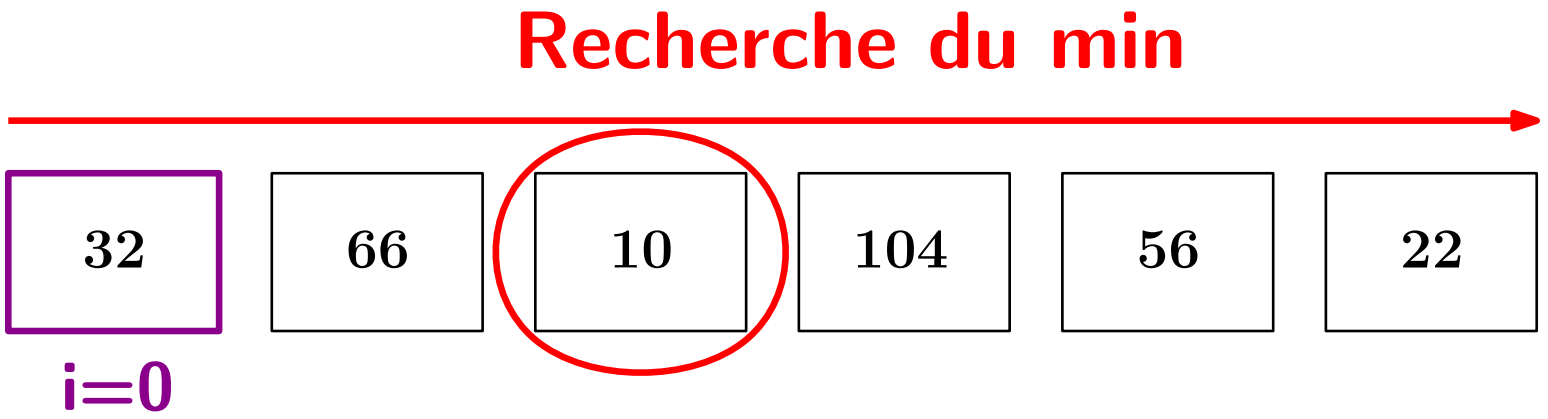


1) Tri par sélection

Début: imaginer que toutes les cartes étudiant sont étalées sur la table, alignées.

Principe: Je veux remplir correctement la case i=0

- Je parcourt du regard l'ensemble des cartes une fois, de gauche à droite, pour **trouver le plus petit numéro**.
- Je prend dans ma main la carte de plus petit numéro, et **je l'échange** avec la 1e : cette carte ne **bougera plus**.

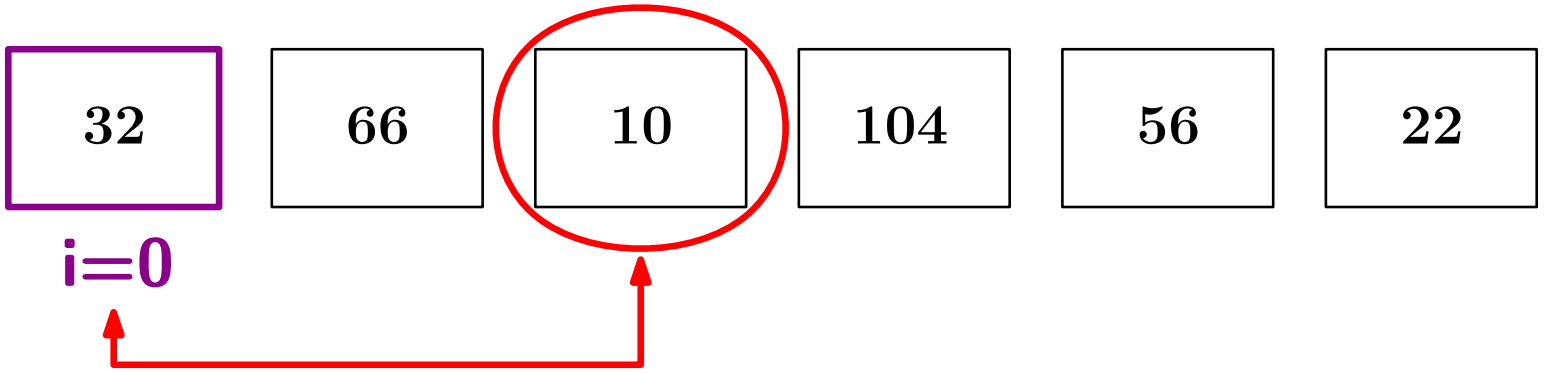


1) Tri par sélection

Début: imaginer que toutes les cartes étudiant sont étalées sur la table, alignées.

Principe: Je veux remplir correctement la case i=0

- Je parcourt du regard l'ensemble des cartes une fois, de gauche à droite, pour **trouver le plus petit numéro**.
- Je prend dans ma main la carte de plus petit numéro, et **je l'échange** avec la 1^{e} : cette carte ne **bougera plus**.

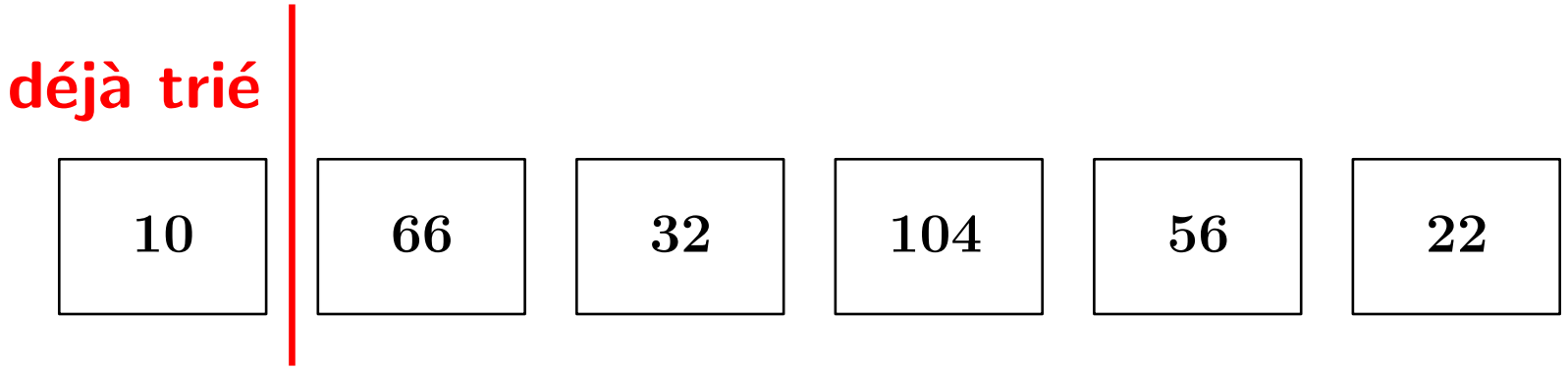


1) Tri par sélection

Début: imaginer que toutes les cartes étudiant sont étalées sur la table, alignées.

Principe: Je veux remplir correctement la case i=0

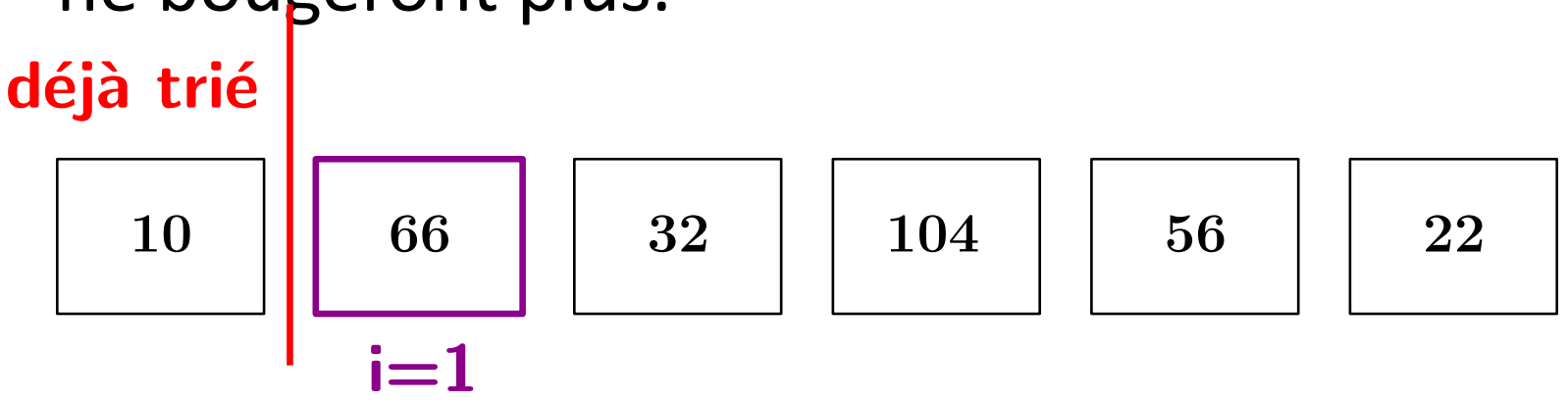
- Je parcourt du regard l'ensemble des cartes une fois, de gauche à droite, pour **trouver le plus petit numéro**.
- Je prend dans ma main la carte de plus petit numéro, et **je l'échange** avec la 1^{e} : cette carte ne **bougera plus**.



1) Tri par sélection

Ensuite je veux **remplir correctement la case i=1**.

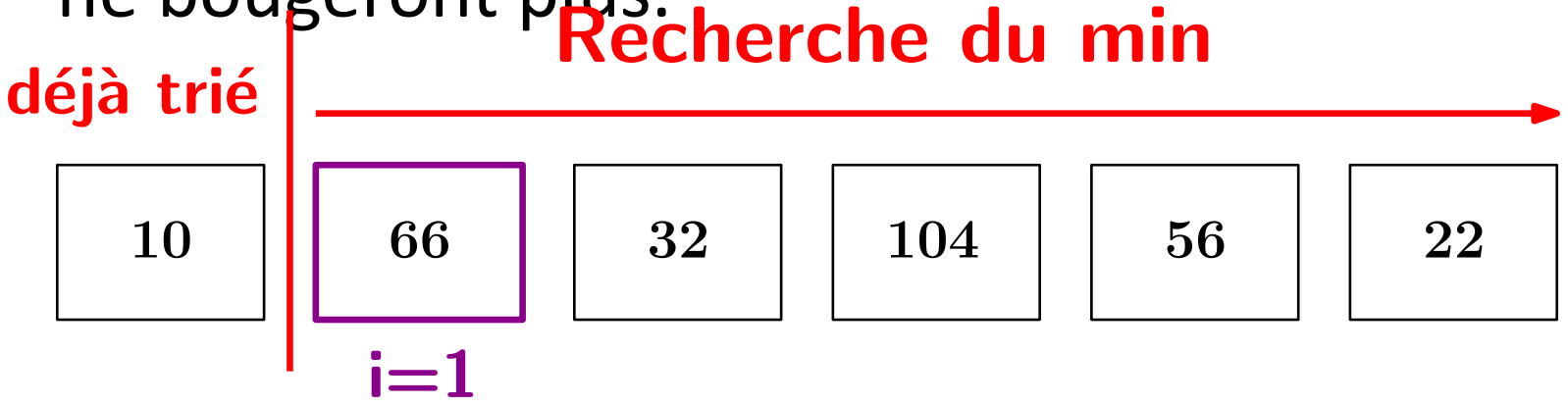
- Je parcourt du regard l'ensemble des cartes, à part la première (qui est déjà traitée), pour **trouver la plus petite parmi les cartes restantes.**
- Je prend cette plus petit carte dans ma main, je **la place juste après la première, en échangeant avec la carte i=1**. J'ai donc désormais deux cartes triées qui ne bougeront plus.



1) Tri par sélection

Ensuite je veux **remplir correctement la case i=1**.

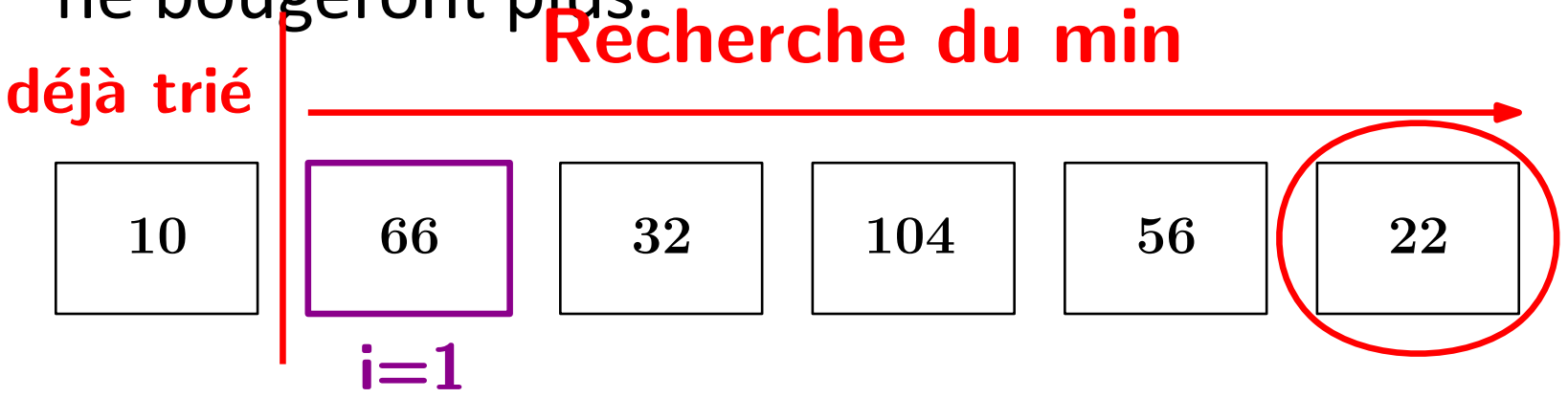
- Je parcourt du regard l'ensemble des cartes, à part la première (qui est déjà traitée), pour **trouver la plus petite parmi les cartes restantes.**
- Je prend cette plus petit carte dans ma main, je **la place juste après la première, en échangeant avec la carte i=1**. J'ai donc désormais deux cartes triées qui ne bougeront plus.



1) Tri par sélection

Ensuite je veux **remplir correctement la case i=1**.

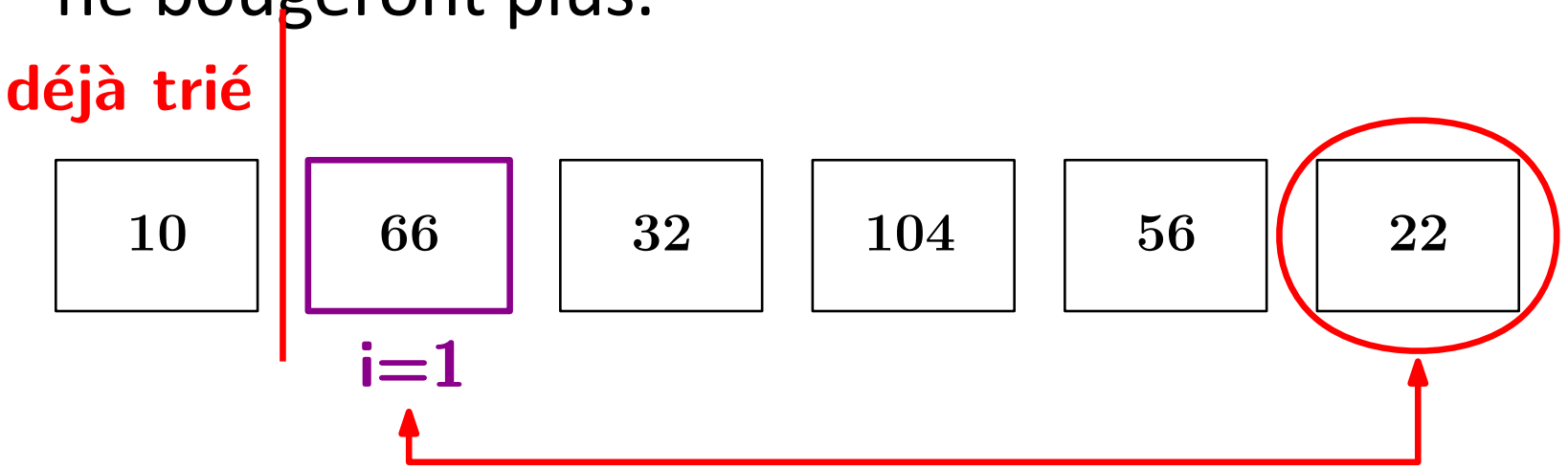
- Je parcourt du regard l'ensemble des cartes, à part la première (qui est déjà traitée), pour **trouver la plus petite parmi les cartes restantes.**
- Je prend cette plus petit carte dans ma main, je **la place juste après la première, en échangeant avec la carte i=1**. J'ai donc désormais deux cartes triées qui ne bougeront plus.



1) Tri par sélection

Ensuite je veux **remplir correctement la case i=1**.

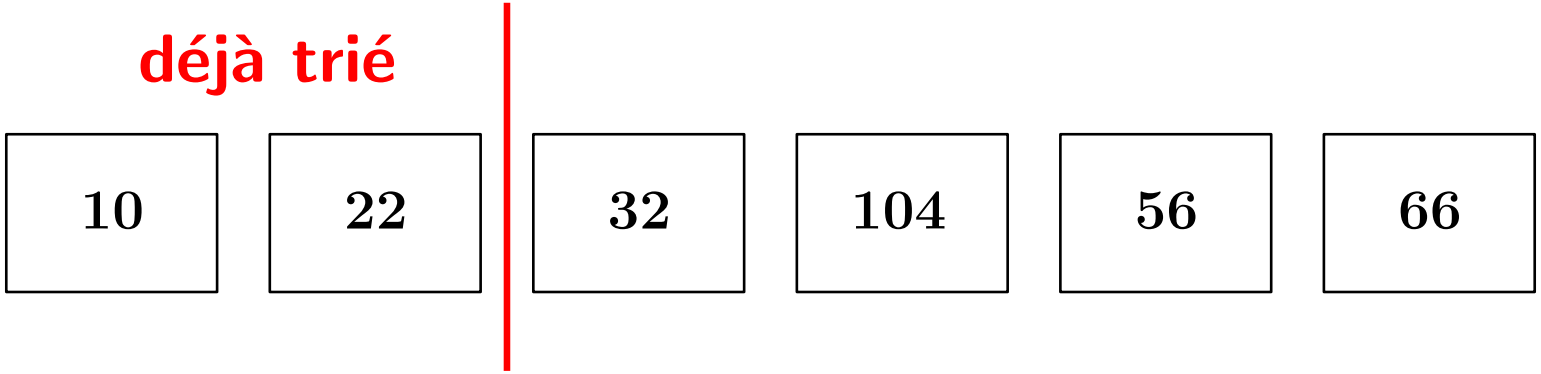
- Je parcourt du regard l'ensemble des cartes, à part la première (qui est déjà traitée), pour **trouver la plus petite parmi les cartes restantes.**
- Je prend cette plus petit carte dans ma main, je **la place juste après la première, en échangeant avec la carte i=1**. J'ai donc désormais deux cartes triées qui ne bougeront plus.



1) Tri par sélection

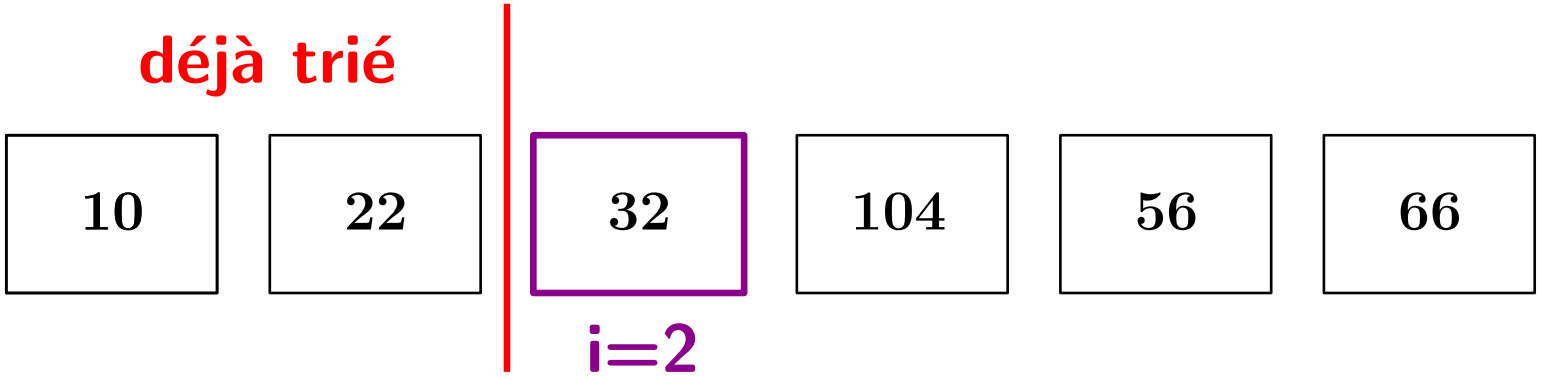
Ensuite je veux **remplir correctement la case i=1**.

- Je parcourt du regard l'ensemble des cartes, à part la première (qui est déjà traitée), pour **trouver la plus petite parmi les cartes restantes.**
- Je prend cette plus petit carte dans ma main, je **la place juste après la première, en échangeant avec la carte i=1**. J'ai donc désormais deux cartes triées qui ne bougeront plus.



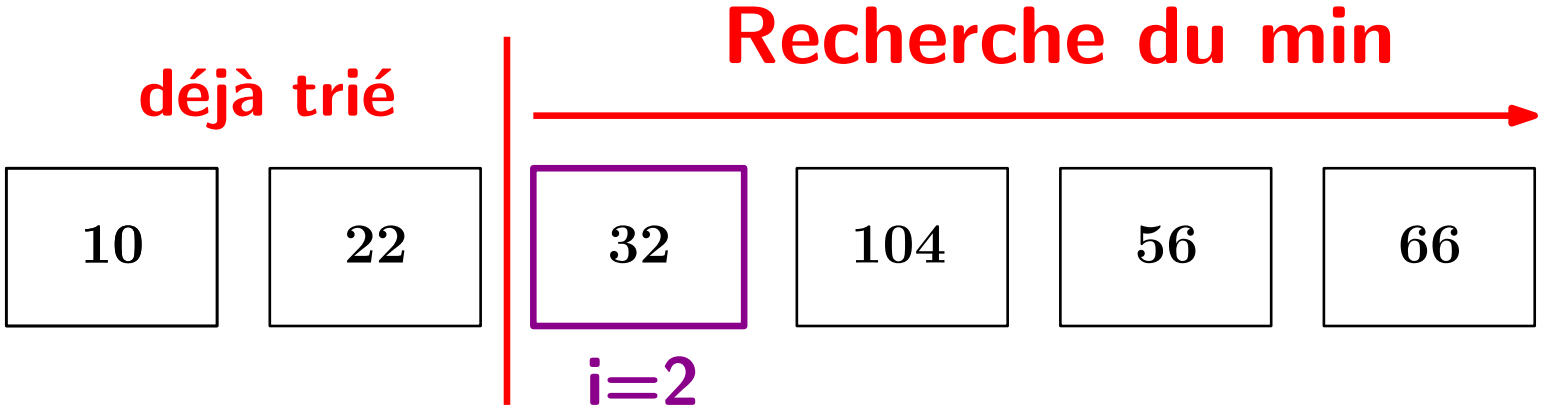
1) Tri par sélection

- **Je recommence pour i=2**: je laisse de côté les deux premières cartes, je **cherche la plus petite carte parmi ce qui reste**, et **la place à la fin de ma partie triée**.
- **Etc…** jusqu'à ce que la partie déjà triée devienne l'ensemble tout entier des cartes.



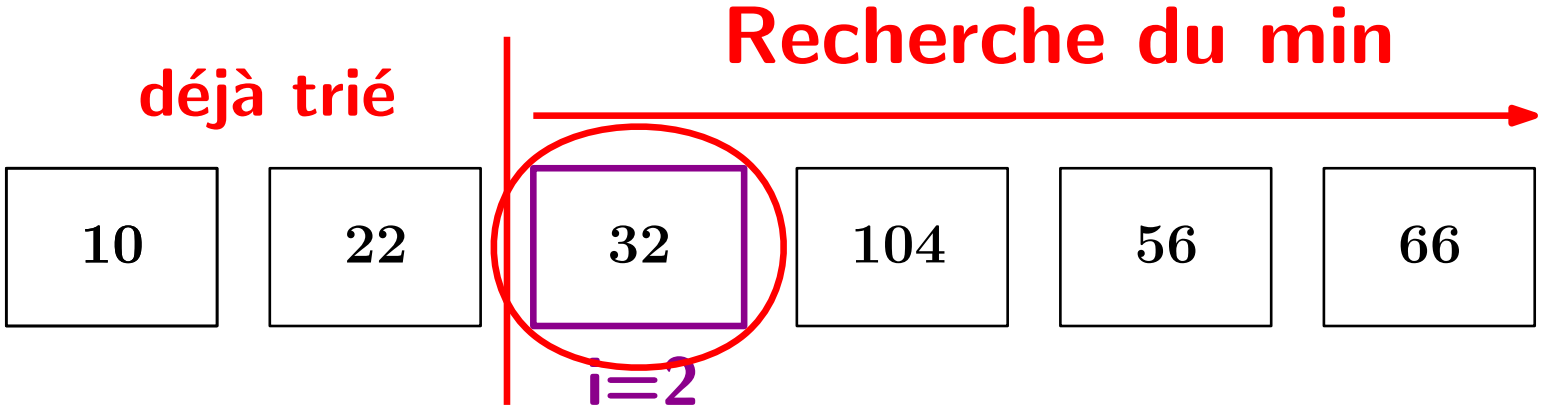
1) Tri par sélection

- **Je recommence pour i=2**: je laisse de côté les deux premières cartes, je **cherche la plus petite carte parmi ce qui reste**, et **la place à la fin de ma partie triée**.
- **Etc…** jusqu'à ce que la partie déjà triée devienne l'ensemble tout entier des cartes.



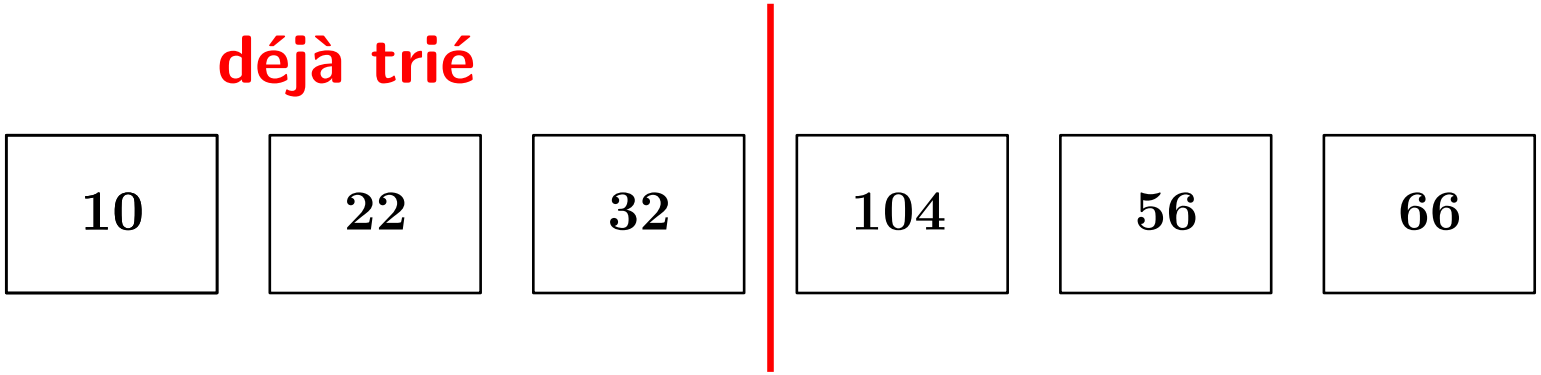
1) Tri par sélection

- **Je recommence pour i=2**: je laisse de côté les deux premières cartes, je **cherche la plus petite carte parmi ce qui reste**, et **la place à la fin de ma partie triée**.
- **Etc…** jusqu'à ce que la partie déjà triée devienne l'ensemble tout entier des cartes.



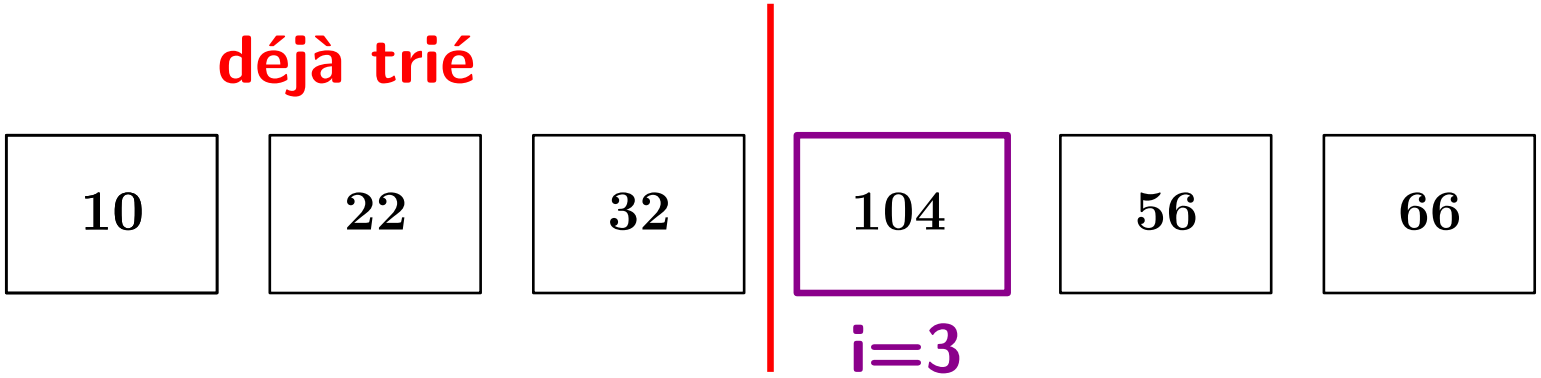
1) Tri par sélection

- **Je recommence pour i=2**: je laisse de côté les deux premières cartes, je **cherche la plus petite carte parmi ce qui reste**, et **la place à la fin de ma partie triée**.
- **Etc…** jusqu'à ce que la partie déjà triée devienne l'ensemble tout entier des cartes.



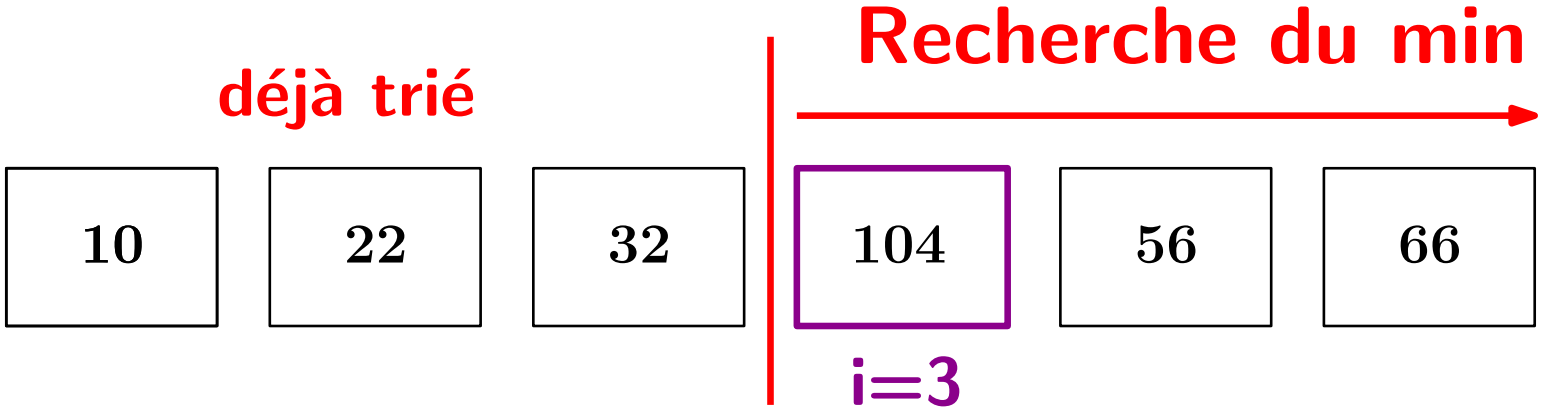
1) Tri par sélection

- **Je recommence pour i=2**: je laisse de côté les deux premières cartes, je **cherche la plus petite carte parmi ce qui reste**, et **la place à la fin de ma partie triée**.
- **Etc…** jusqu'à ce que la partie déjà triée devienne l'ensemble tout entier des cartes.



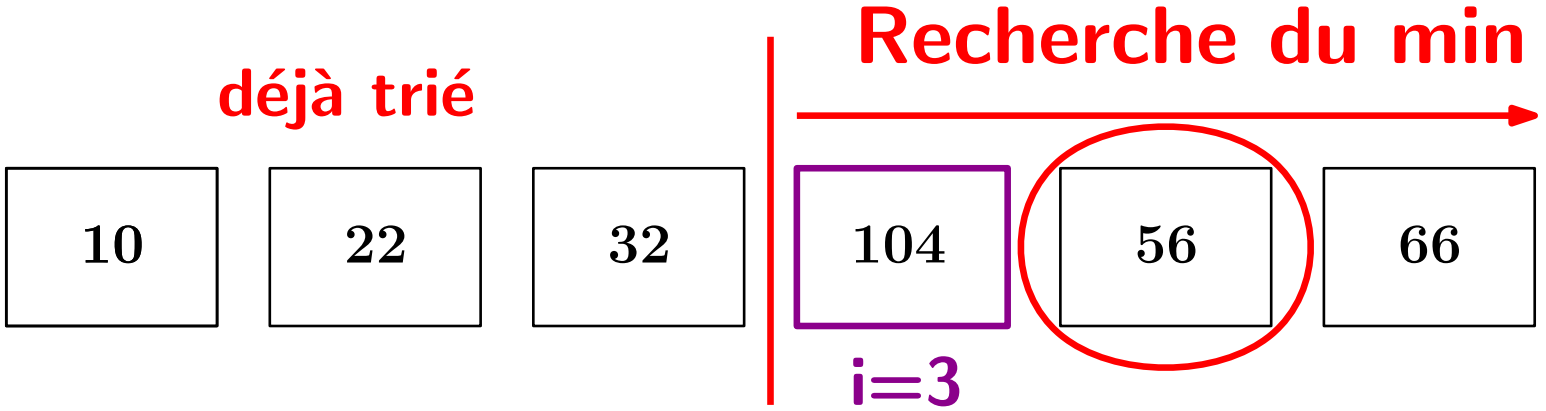
1) Tri par sélection

- **Je recommence pour i=2**: je laisse de côté les deux premières cartes, je **cherche la plus petite carte parmi ce qui reste**, et **la place à la fin de ma partie triée**.
- **Etc…** jusqu'à ce que la partie déjà triée devienne l'ensemble tout entier des cartes.



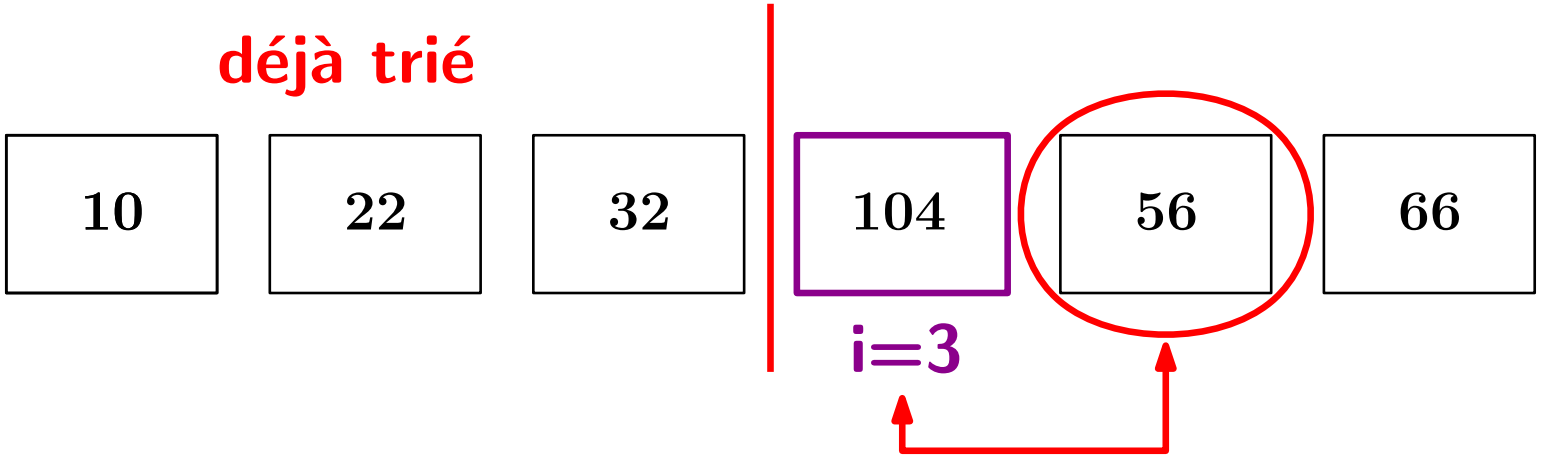
1) Tri par sélection

- **Je recommence pour i=2**: je laisse de côté les deux premières cartes, je **cherche la plus petite carte parmi ce qui reste**, et **la place à la fin de ma partie triée**.
- **Etc…** jusqu'à ce que la partie déjà triée devienne l'ensemble tout entier des cartes.



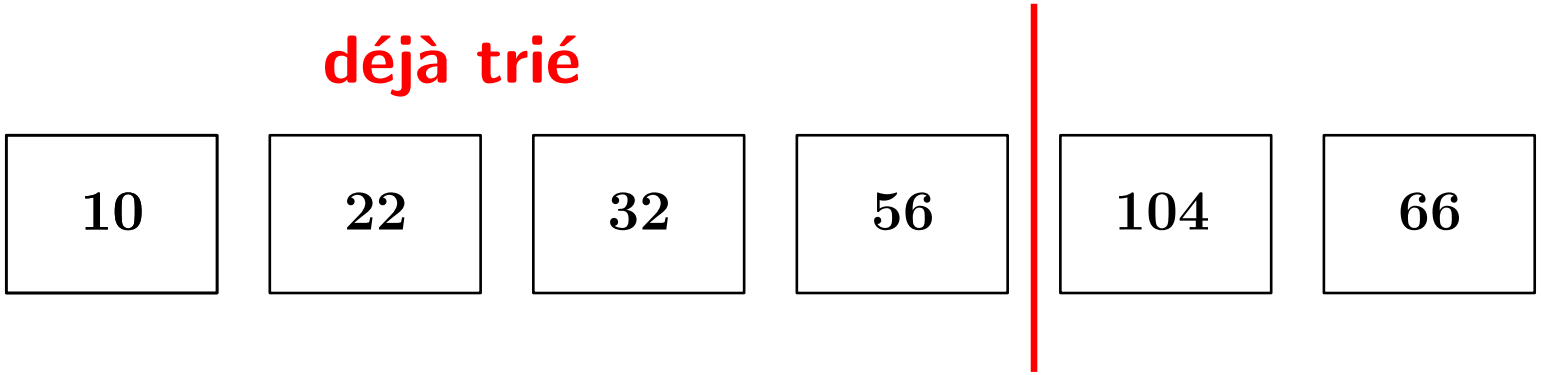
1) Tri par sélection

- **Je recommence pour i=2**: je laisse de côté les deux premières cartes, je **cherche la plus petite carte parmi ce qui reste**, et **la place à la fin de ma partie triée**.
- **Etc…** jusqu'à ce que la partie déjà triée devienne l'ensemble tout entier des cartes.



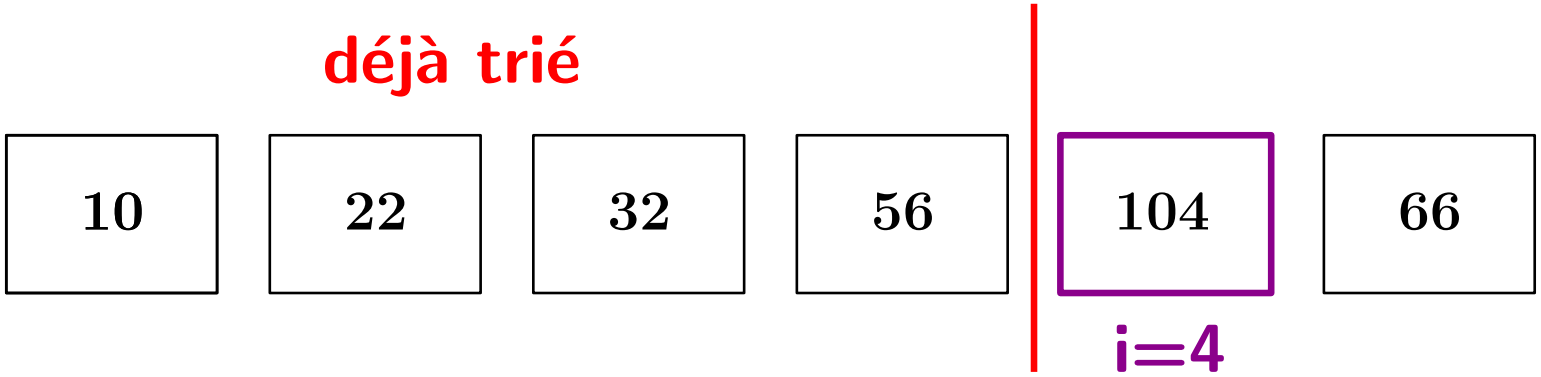
1) Tri par sélection

- **Je recommence pour i=2**: je laisse de côté les deux premières cartes, je **cherche la plus petite carte parmi ce qui reste**, et **la place à la fin de ma partie triée**.
- **Etc…** jusqu'à ce que la partie déjà triée devienne l'ensemble tout entier des cartes.



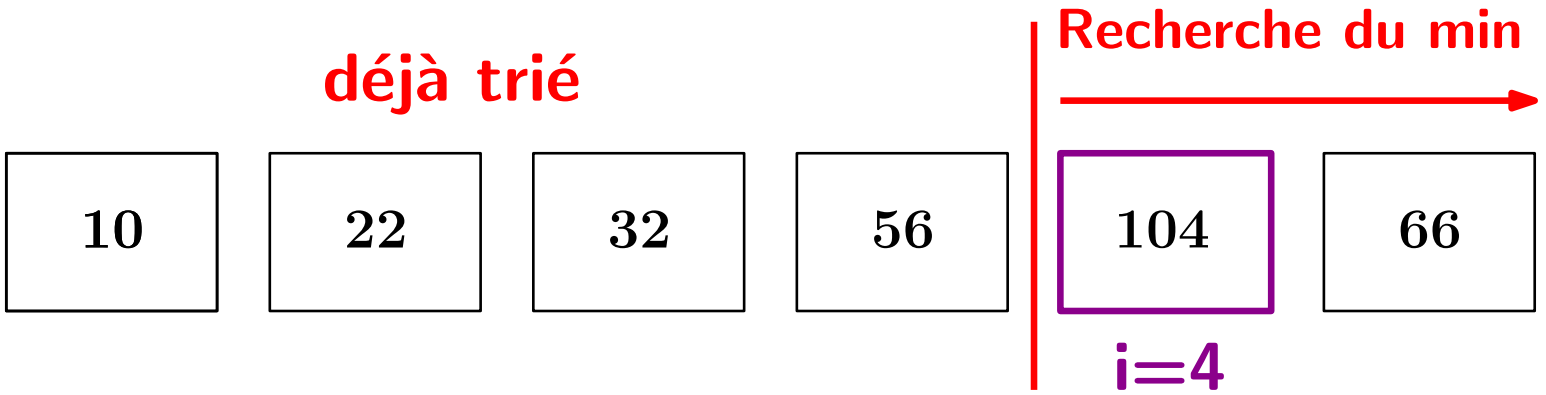
1) Tri par sélection

- **Je recommence pour i=2**: je laisse de côté les deux premières cartes, je **cherche la plus petite carte parmi ce qui reste**, et **la place à la fin de ma partie triée**.
- **Etc…** jusqu'à ce que la partie déjà triée devienne l'ensemble tout entier des cartes.



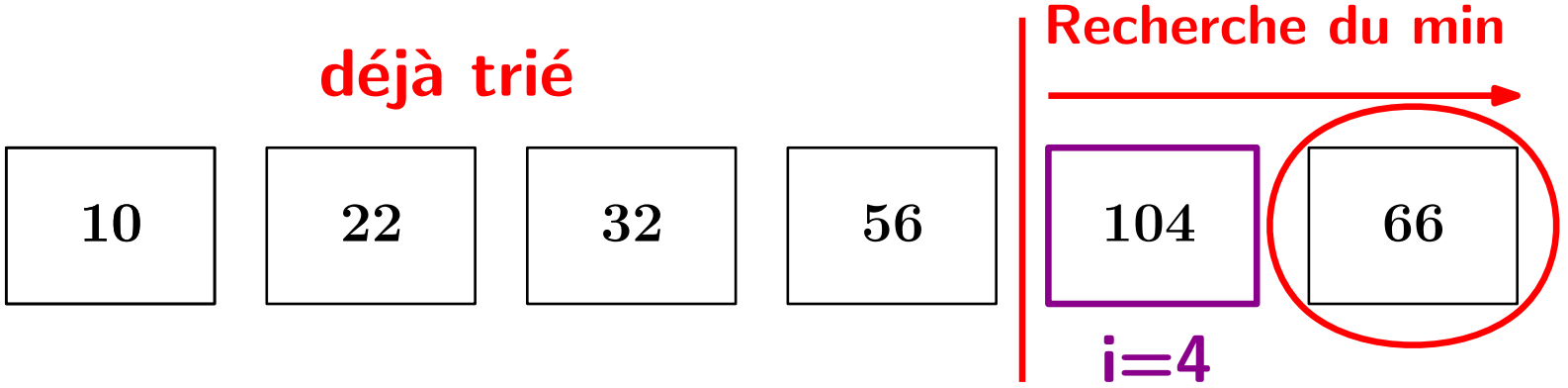
1) Tri par sélection

- **Je recommence pour i=2**: je laisse de côté les deux premières cartes, je **cherche la plus petite carte parmi ce qui reste**, et **la place à la fin de ma partie triée**.
- **Etc…** jusqu'à ce que la partie déjà triée devienne l'ensemble tout entier des cartes.



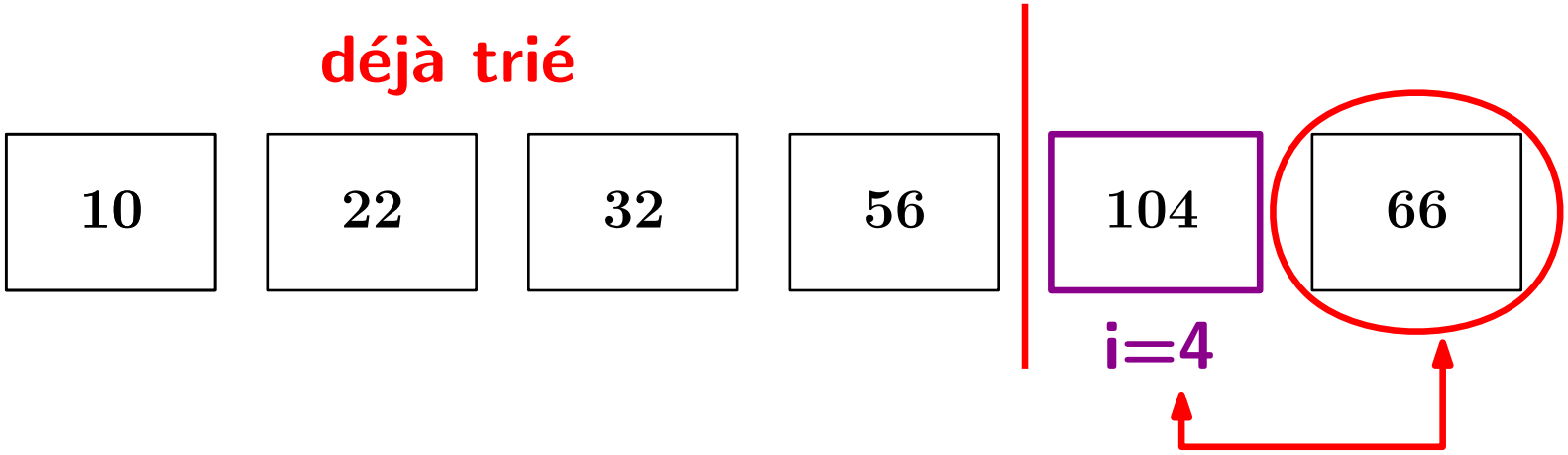
1) Tri par sélection

- **Je recommence pour i=2**: je laisse de côté les deux premières cartes, je **cherche la plus petite carte parmi ce qui reste**, et **la place à la fin de ma partie triée**.
- **Etc…** jusqu'à ce que la partie déjà triée devienne l'ensemble tout entier des cartes.



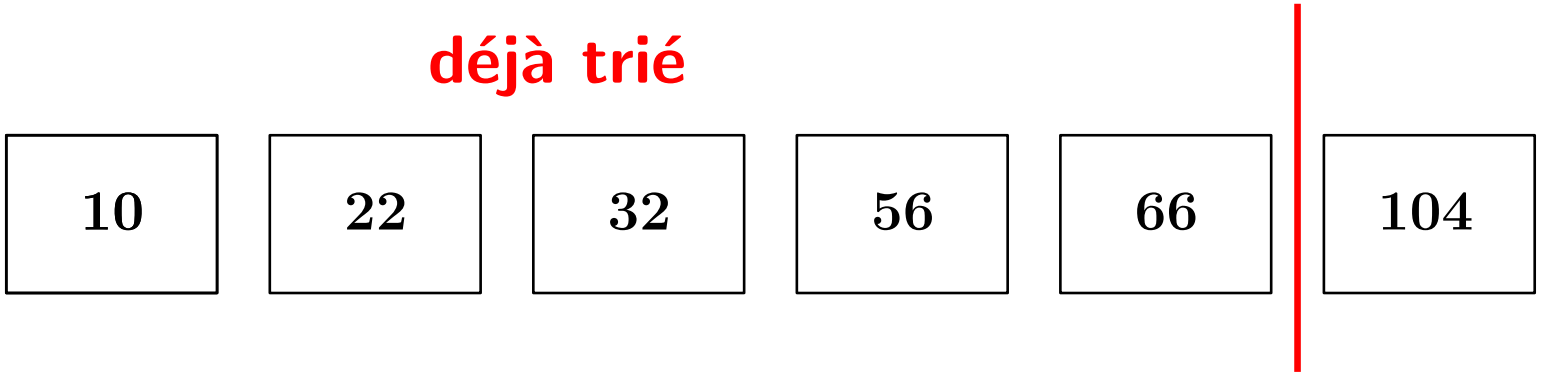
1) Tri par sélection

- **Je recommence pour i=2**: je laisse de côté les deux premières cartes, je **cherche la plus petite carte parmi ce qui reste**, et **la place à la fin de ma partie triée**.
- **Etc…** jusqu'à ce que la partie déjà triée devienne l'ensemble tout entier des cartes.



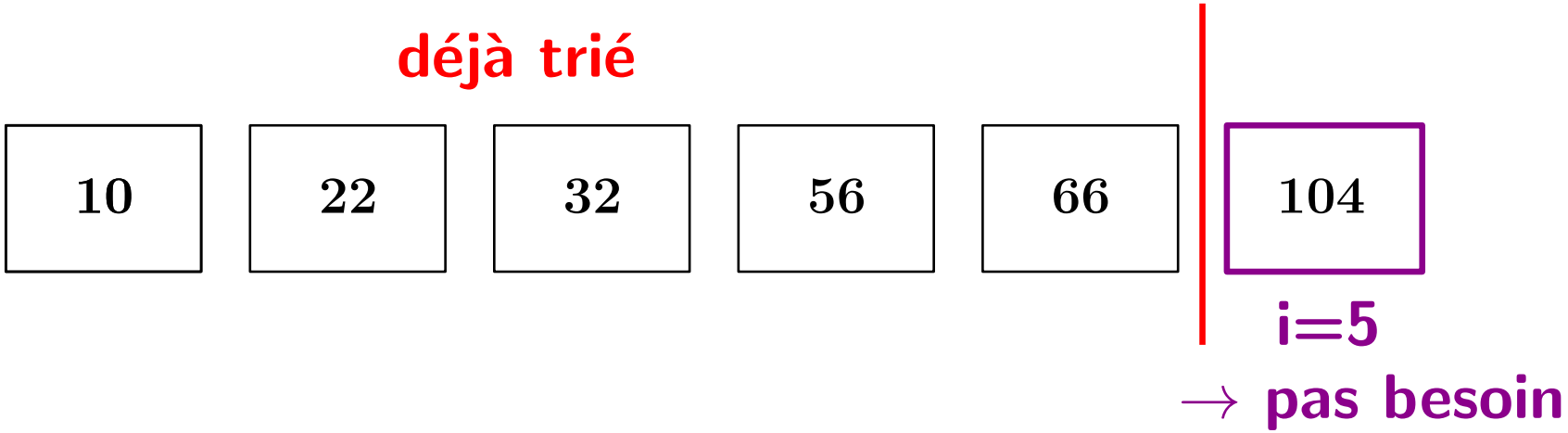
1) Tri par sélection

- **Je recommence pour i=2**: je laisse de côté les deux premières cartes, je **cherche la plus petite carte parmi ce qui reste**, et **la place à la fin de ma partie triée**.
- **Etc…** jusqu'à ce que la partie déjà triée devienne l'ensemble tout entier des cartes.



1) Tri par sélection

- **Je recommence pour i=2**: je laisse de côté les deux premières cartes, je **cherche la plus petite carte parmi ce qui reste**, et **la place à la fin de ma partie triée**.
- **Etc…** jusqu'à ce que la partie déjà triée devienne l'ensemble tout entier des cartes.



1) Tri par sélection

- **Je recommence pour i=2**: je laisse de côté les deux premières cartes, je **cherche la plus petite carte parmi ce qui reste**, et **la place à la fin de ma partie triée**.
- **Etc…** jusqu'à ce que la partie déjà triée devienne l'ensemble tout entier des cartes.

1) Tri par sélection

```
Fun int selectMinParmi(nombre tab[ ], int indiceDebut, int indiceFin):
    /* Renvoie l'indice de la case contenant la plus petite
       valeur, parmi les cases dont le numéro  est entre
       indiceDebut et indiceFin */
    int i
    int indiceDuMin:=indiceDebut
    Pour i de indiceDebut+1 à indiceFin:
        Si tab[i]<tab[indiceDuMin]:
            indiceDuMin:=i
    Renvoyer indiceDuMin
```

1) Tri par sélection

```
Proc echangeContenu(nombre tab[ ], int i, int j):
    /* Echange le contenu de la case i avec celui de la case j*/
    nombre sauv:=tab[i]
    tab[i]:=tab[j]
    tab[j]:=sauv
```

1) Tri par sélection

```
Proc triSelection(nombre tab[ ]):
    int i, indiceDuMin
    Pour i de 0 à longu(tab)-2:
        indiceDuMin:=selectMinParmi(tab, i, longu(tab)-1)
        echangeContenu(tab, indiceDuMin, i)
        /* Invariant de boucle: à cet endroit de l'algo., les cases
         numérotées 0 à i contiennent exactement les i+1 plus
        petites valeurs du tableau, dans l'ordre croissant.
        Chaque nombre initialement présent dans le tableau apparaît
        toujours dans l'état actuel du tableau (le même nombre de fois
         qu'initialement). */
```

1) Tri par sélection

Rappel sur l'invariant de boucle: un **invariant de boucle** sert à vérifier/**prouver** ce que notre algorithme fait réellement.

- il faut vérifier qu'il marche lors du premier passage
- il faut vérifier que, s'il est vrai à un certain passage, alors il est également vrai au passage suivant.
- il faut vérifier que, lors du dernier passage, l'invariant soit en accord avec ce que nous cherchons pour notre algorithme.

Choisir à l'avance l'invariant de boucle que l'on souhaite obtenir peut même **nous aider à écrire notre algorithme**!

1) Tri par sélection

Sur notre exemple:

- il faut vérifier qu'il marche lors du **premier passage** (ici **i=0**)

```
Proc triSelection(nombre tab[ ]):
    int i, indiceDuMin
    Pour i de 0 à longu(tab)-2:
        indiceDuMin:=selectMinParmi(tab, i, longu(tab)-1)
        echangeContenu(tab, indiceDuMin, i)
```

Inv.: les cases numérotées 0 à i contiennent exactement les i+1 plus petites valeurs du tableau, dans l'ordre croissant
+ conservation des valeurs

/* Au premier passage, la case numérotée 0 contient exactement la (1e) plus petite valeur du tableau (dans l'ordre croissant) */

1) Tri par sélection

Inv.: les cases numérotées 0 à i contiennent exactement les i+1 plus petites valeurs du tableau, dans l'ordre croissant + conservation des valeurs

- il faut vérifier que, s'il est **vrai à un certain passage**, alors il est également vrai au **passage suivant** (≈ récurrence).

```
Proc triSelection(nombre tab[ ]):
    int i, indiceDuMin
    Pour i de 0 à longu(tab)-2:
        /* On sait que: les cases 0 à i-1 contiennent les i
           plus petites valeurs, dans l'ordre croissant*/
        indiceDuMin:=selectMinParmi(tab, i, longu(tab)-1)
        echangeContenu(tab, indiceDuMin, i)
        /* La case i a reçu la plus petite valeur parmi celles
           restantes dans les cases i à longu(tab)-1
           → c'est forcément la (i+1)e plus petite valeur du
           tableau initial → ok pour l'invariant*/
```

1) Tri par sélection

Inv.: les cases numérotées 0 à i contiennent exactement les i+1 plus petites valeurs du tableau, dans l'ordre croissant + conservation des valeurs

- il faut vérifier que, **lors du dernier passage, l'invariant soit en accord avec ce que nous cherchons** pour notre algorithme.

```
Proc triSelection(nombre tab[ ]):
    int i, indiceDuMin
    Pour i de 0 à longu(tab)-2:
        indiceDuMin:=selectMinParmi(tab, i, longu(tab)-1)
        echangeContenu(tab, indiceDuMin, i)
        /* Dernier passage (i=longu(tab)-2): les cases num. 0 à
        longu(tab)-2 contiennent les longu(tab)-1 plus petites
        valeurs, dans l'ordre croissant. → la case longu(tab)-1
        contient donc forcément la plus grande valeur
        → cases 0 à longu(tab)-1 sont dans l'ordre croissant
        → tout le tableau est dans l'ordre croissant !*/
```

1) Tri par sélection

Complexité: on appelle *n* la taille du tableau, et on compte les op. élémentaires de tableau.

```
Fun int selectMinParmi(nombre tab[ ], int indiceDebut, int indiceFin):
    /* Renvoie l'indice de la case contenant la plus petite  valeur, parmi les
            cases dont le numéro  est entre  indiceDebut et indiceFin */
    int i
    int indiceDuMin:=indiceDebut
    Pour i de indiceDebut+1 à indiceFin:
        Si tab[i]<tab[indiceDuMin]:
            indiceDuMin:=i
    Renvoyer indiceDuMin
```

Complexité: O(*n*)

1) Tri par sélection

```
Proc echangeContenu(nombre tab[ ], int i, int j):
    /* Echange le contenu de la case i avec celui de la case j*/
    nombre sauv:=tab[i]
    tab[i]:=tab[j]
    tab[j]:=sauv
```

Complexité:
4 donc O(1)

1) Tri par sélection

```
Proc triSelection(nombre tab[ ]):
    int i, indiceDuMin
    Pour i de 0 à longu(tab)-2:
        indiceDuMin:=selectMinParmi(tab, i, longu(tab)-1)
        echangeContenu(tab, indiceDuMin, i)
```

n-1 fois

O(*n*)

O(1)

Total: $(n-1)\times\big(O(n)+O(1)\big)$

$\approx (n-1)\times(n+1)$

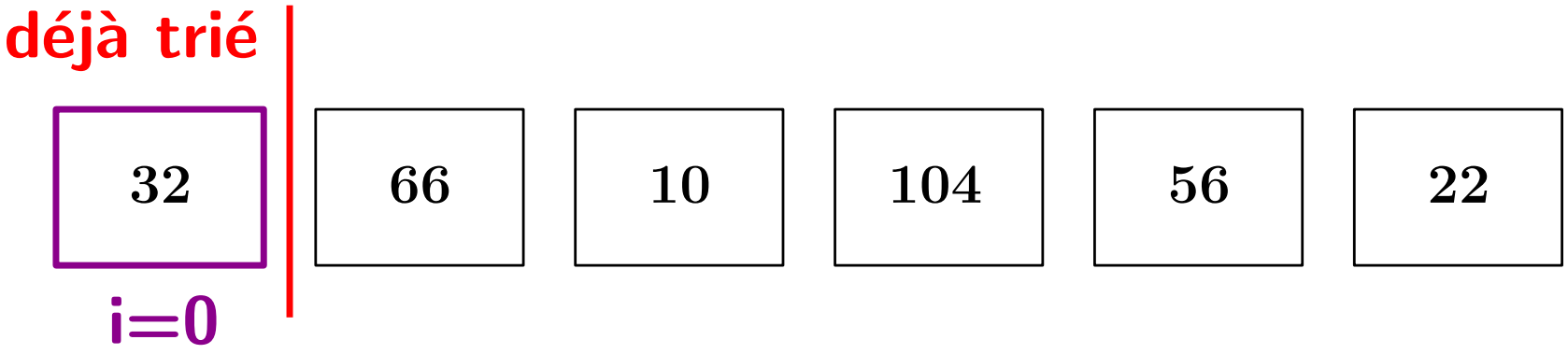
→ $O(n^2)$ quadratique

2) Tri par insertion

Début: imaginer que toutes les cartes étudiant sont étalées sur la table, alignées.

Principe: La case **i=0 forme déjà un ensemble trié.**

- Je considère la case i=1, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0 ou 1).
- **Maintenant, les cases 0 à 1 forment un ensemble trié.**

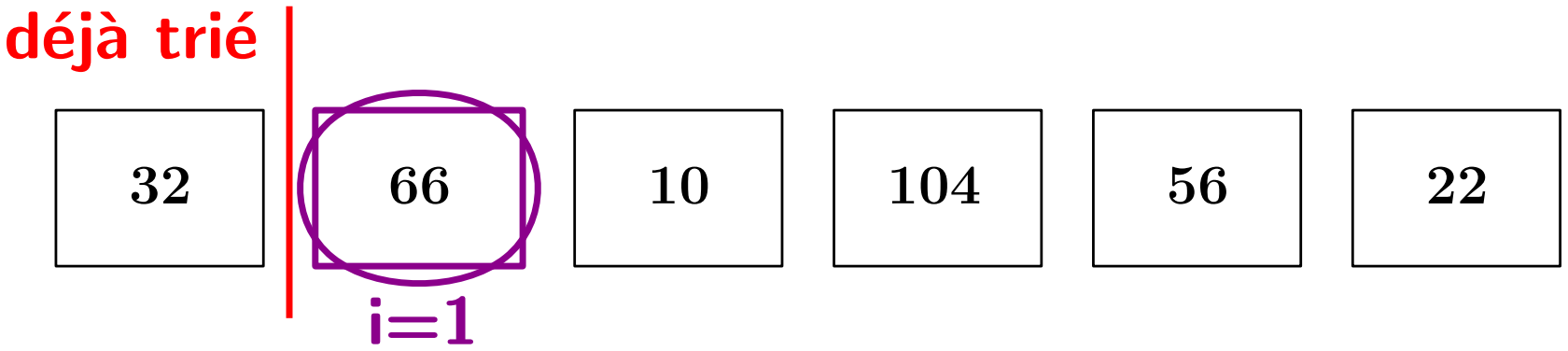


2) Tri par insertion

Début: imaginer que toutes les cartes étudiant sont étalées sur la table, alignées.

Principe: La case **i=0 forme déjà un ensemble trié.**

- Je considère la case i=1, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0 ou 1).
- **Maintenant, les cases 0 à 1 forment un ensemble trié.**

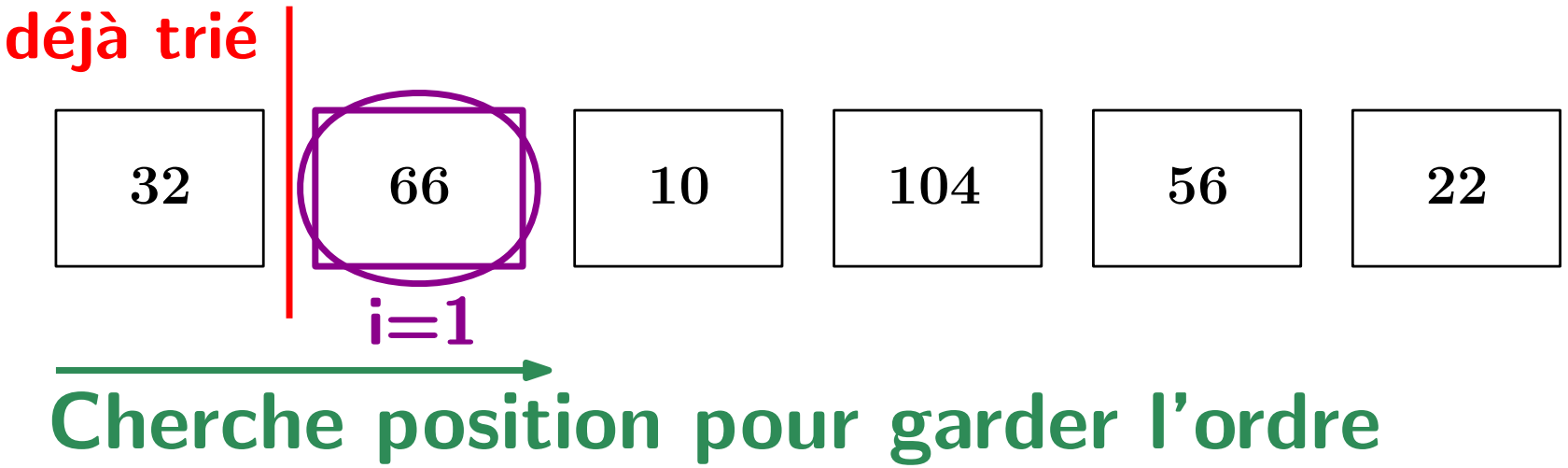


2) Tri par insertion

Début: imaginer que toutes les cartes étudiant sont étalées sur la table, alignées.

Principe: La case **i=0 forme déjà un ensemble trié.**

- Je considère la case i=1, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0 ou 1).
- **Maintenant, les cases 0 à 1 forment un ensemble trié.**

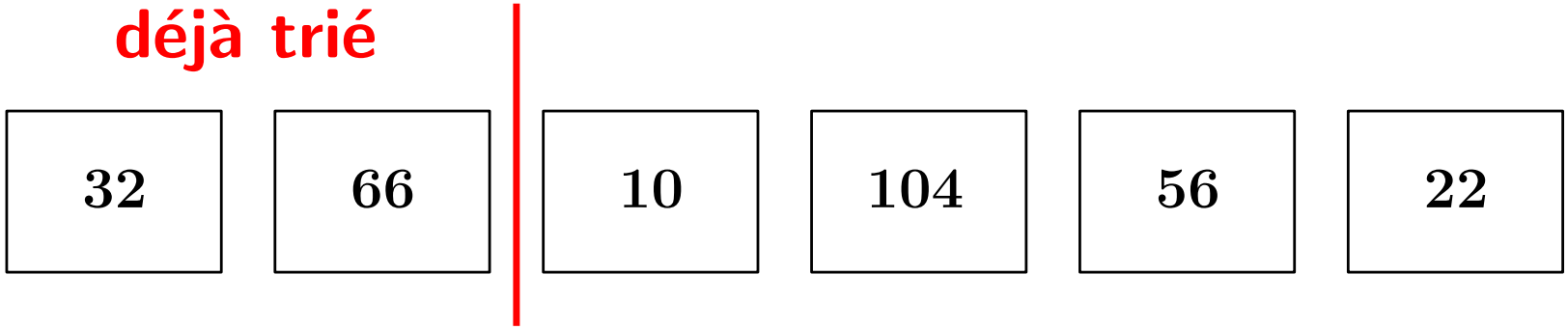


2) Tri par insertion

Début: imaginer que toutes les cartes étudiant sont étalées sur la table, alignées.

Principe: La case **i=0 forme déjà un ensemble trié.**

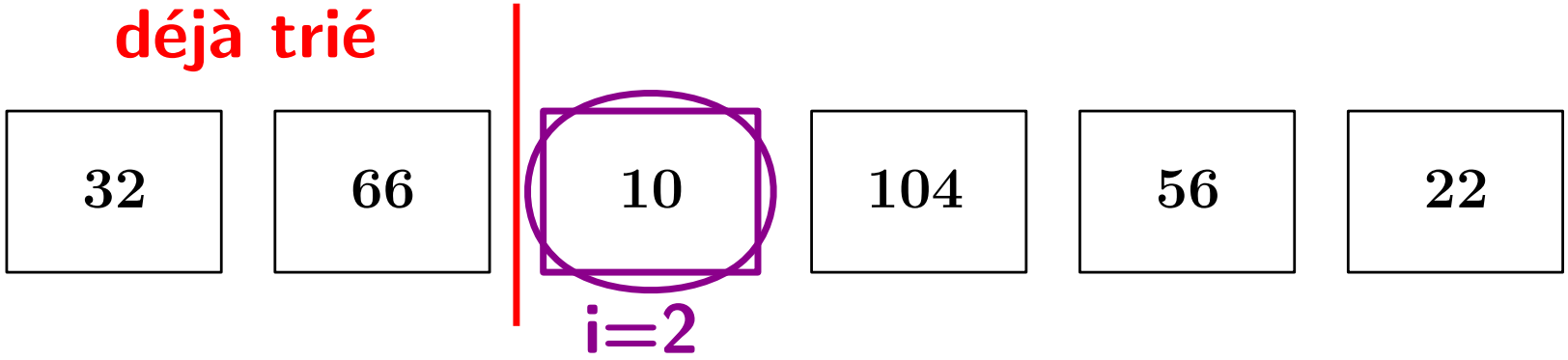
- Je considère la case i=1, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0 ou 1).
- **Maintenant, les cases 0 à 1 forment un ensemble trié.**



2) Tri par insertion

Principe: Les cases **0 et 1 forment déjà un ensemble trié.**

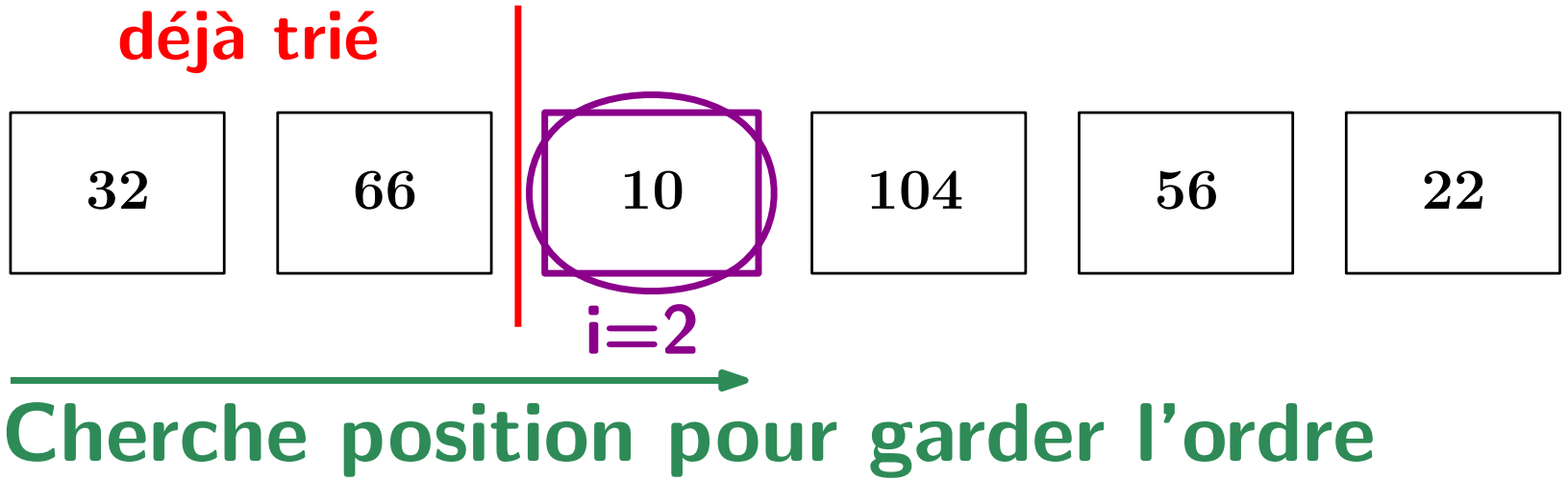
- Je considère la case i=2, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0, 1 ou 2).
- Lorsque j'insère une carte dans la partie triée, je **décale toute la partie droite de l'ensemble déjà** trié d'un cran**.**



2) Tri par insertion

Principe: Les cases **0 et 1 forment déjà un ensemble trié.**

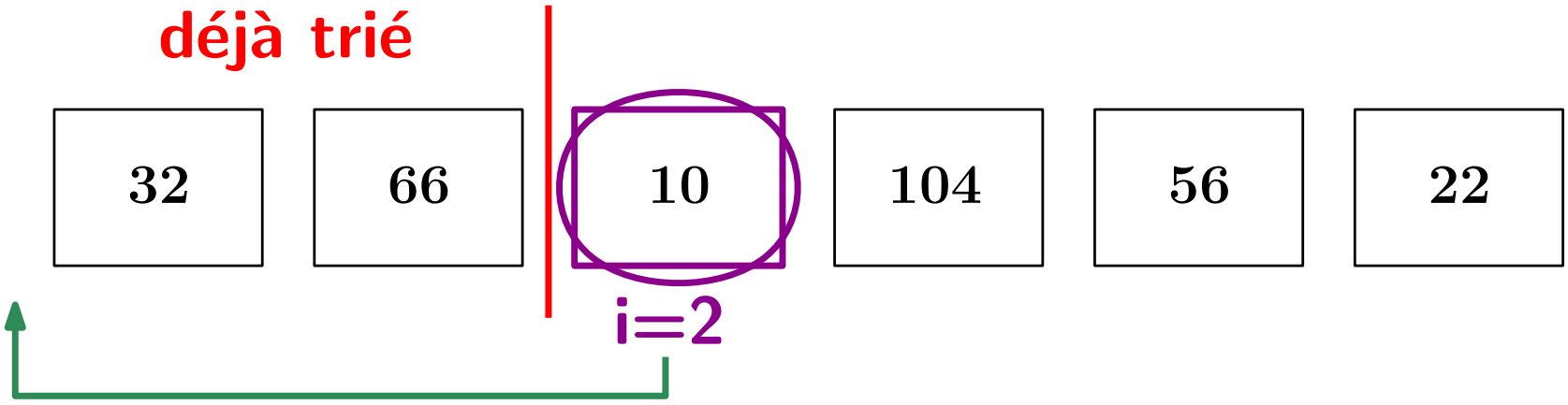
- Je considère la case i=2, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0, 1 ou 2).
- Lorsque j'insère une carte dans la partie triée, je **décale toute la partie droite de l'ensemble déjà** trié d'un cran**.**



2) Tri par insertion

Principe: Les cases **0 et 1 forment déjà un ensemble trié.**

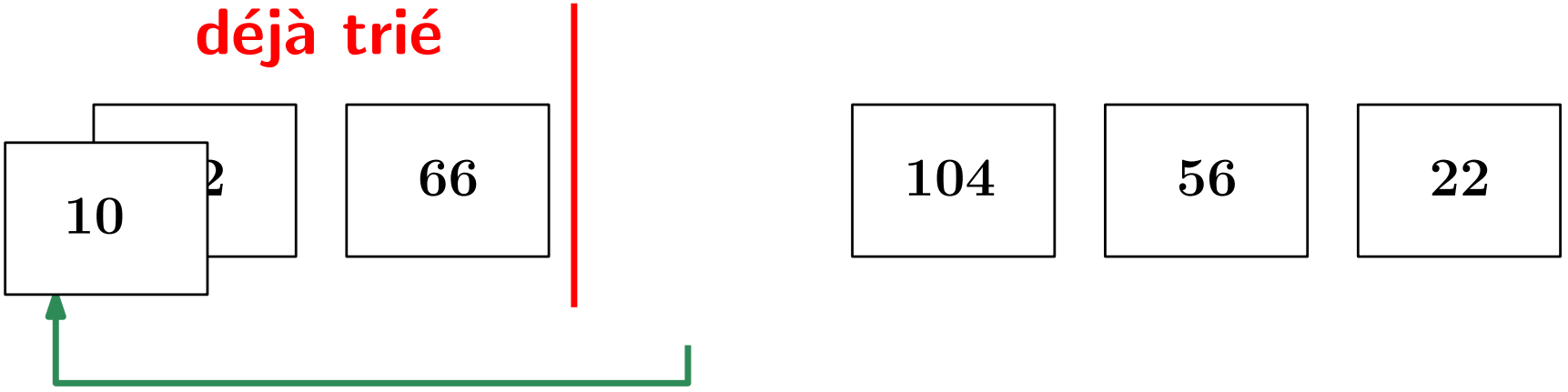
- Je considère la case i=2, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0, 1 ou 2).
- Lorsque j'insère une carte dans la partie triée, je **décale toute la partie droite de l'ensemble déjà** trié d'un cran**.**



2) Tri par insertion

Principe: Les cases **0 et 1 forment déjà un ensemble trié.**

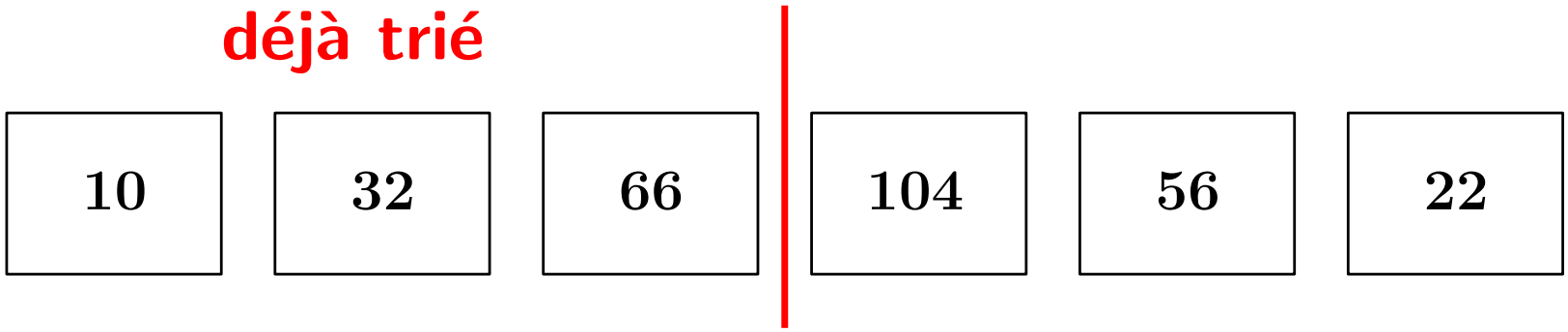
- Je considère la case i=2, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0, 1 ou 2).
- Lorsque j'insère une carte dans la partie triée, je **décale toute la partie droite de l'ensemble déjà** trié d'un cran**.**



2) Tri par insertion

Principe: Les cases **0 et 1 forment déjà un ensemble trié.**

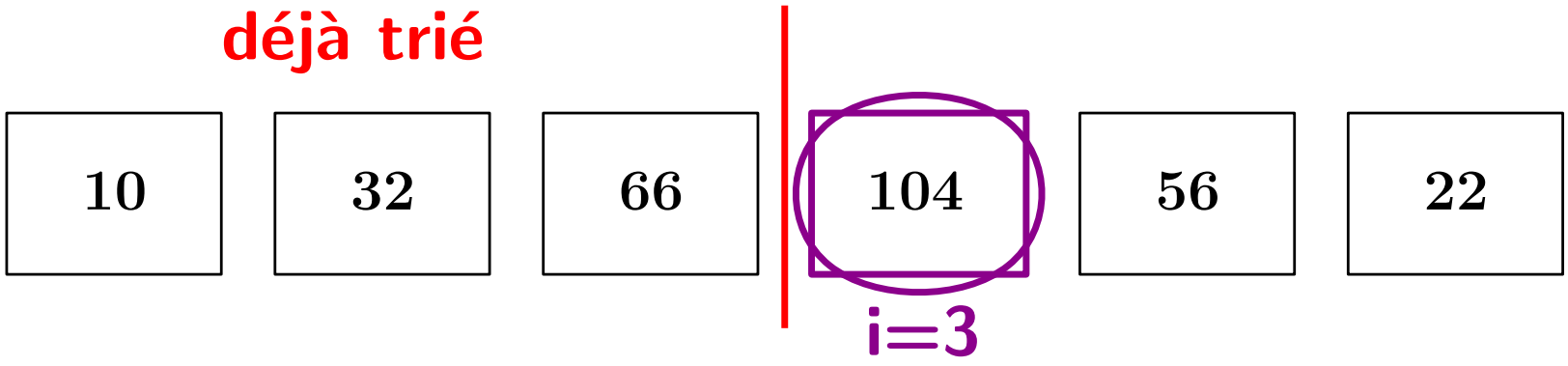
- Je considère la case i=2, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0, 1 ou 2).
- Lorsque j'insère une carte dans la partie triée, je **décale toute la partie droite de l'ensemble déjà** trié d'un cran**.**
- A la fin de cette étape, les cases 0 à 2 forment un ensemble trié.



2) Tri par insertion

Principe: Les cases **0 à 2 forment déjà un ensemble trié.**

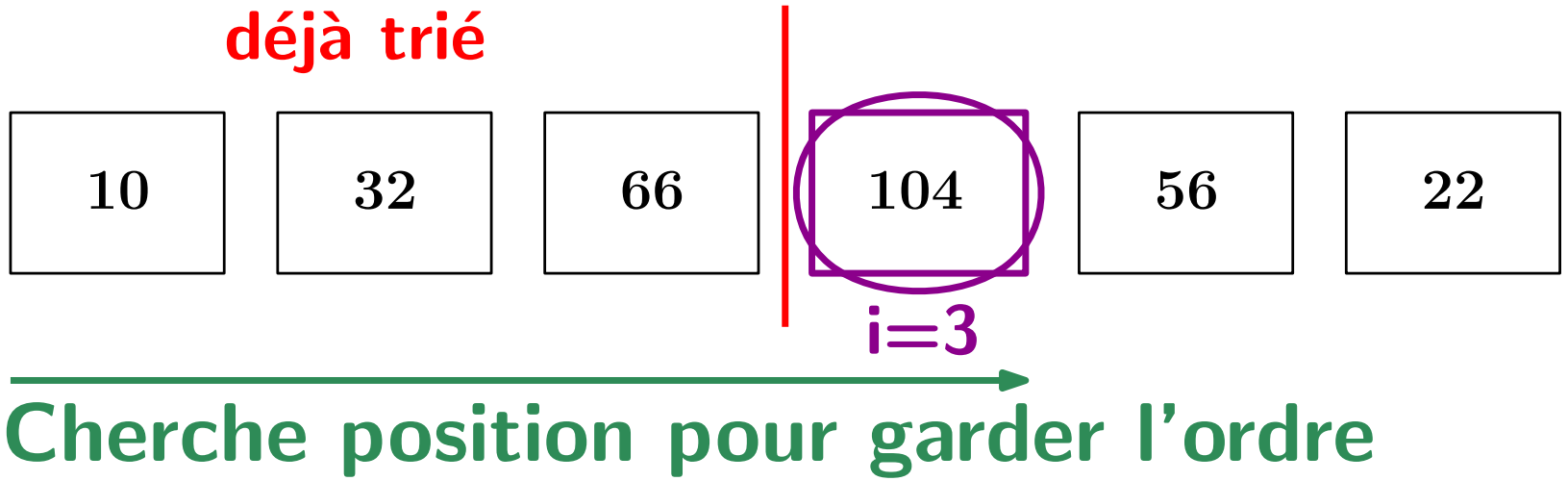
- Je considère la case i=3, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0, 1, 2 ou 3).
- Lorsque j'insère une carte dans la partie triée, je **décale toute la partie droite de l'ensemble déjà** trié d'un cran**.**
- Etc… jusqu'à ce que la partie déjà triée comprennent tout le tableau.



2) Tri par insertion

Principe: Les cases **0 à 2 forment déjà un ensemble trié.**

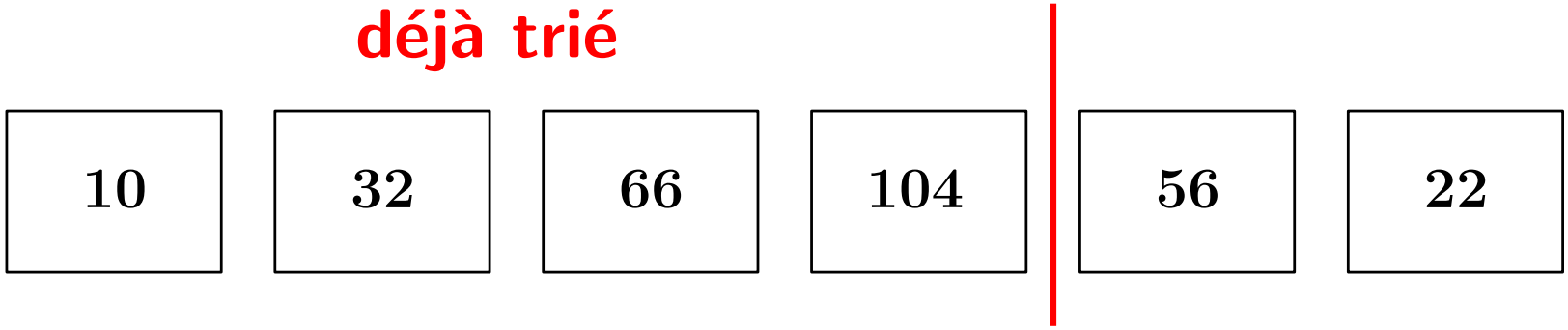
- Je considère la case i=3, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0, 1, 2 ou 3).
- Lorsque j'insère une carte dans la partie triée, je **décale toute la partie droite de l'ensemble déjà** trié d'un cran.
- Etc… jusqu'à ce que la partie déjà triée comprennent tout le tableau.



2) Tri par insertion

Principe: Les cases **0 à 2 forment déjà un ensemble trié.**

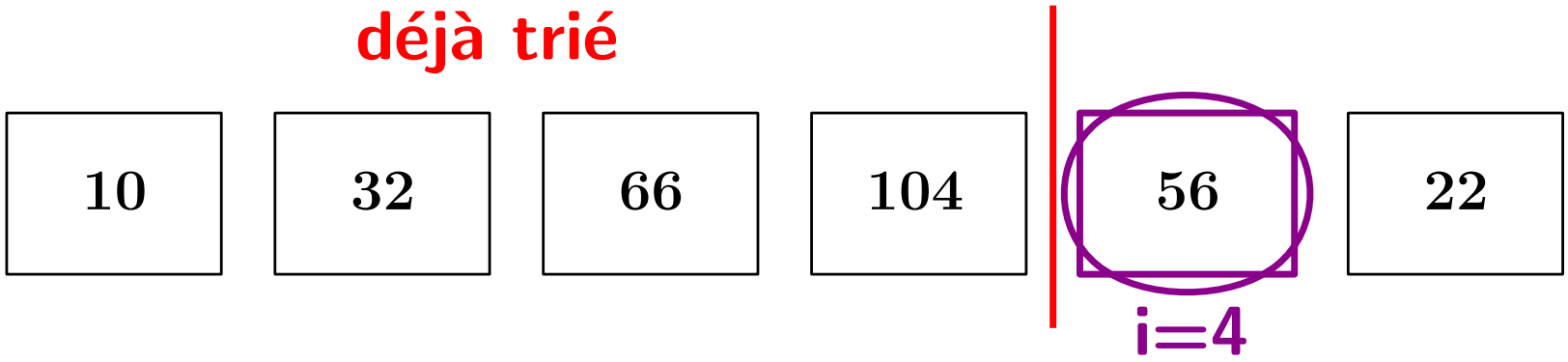
- Je considère la case i=3, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0, 1, 2 ou 3).
- Lorsque j'insère une carte dans la partie triée, je **décale toute la partie droite de l'ensemble déjà** trié d'un cran**.**
- Etc… jusqu'à ce que la partie déjà triée comprennent tout le tableau.



2) Tri par insertion

Principe: Les cases **0 à 2 forment déjà un ensemble trié.**

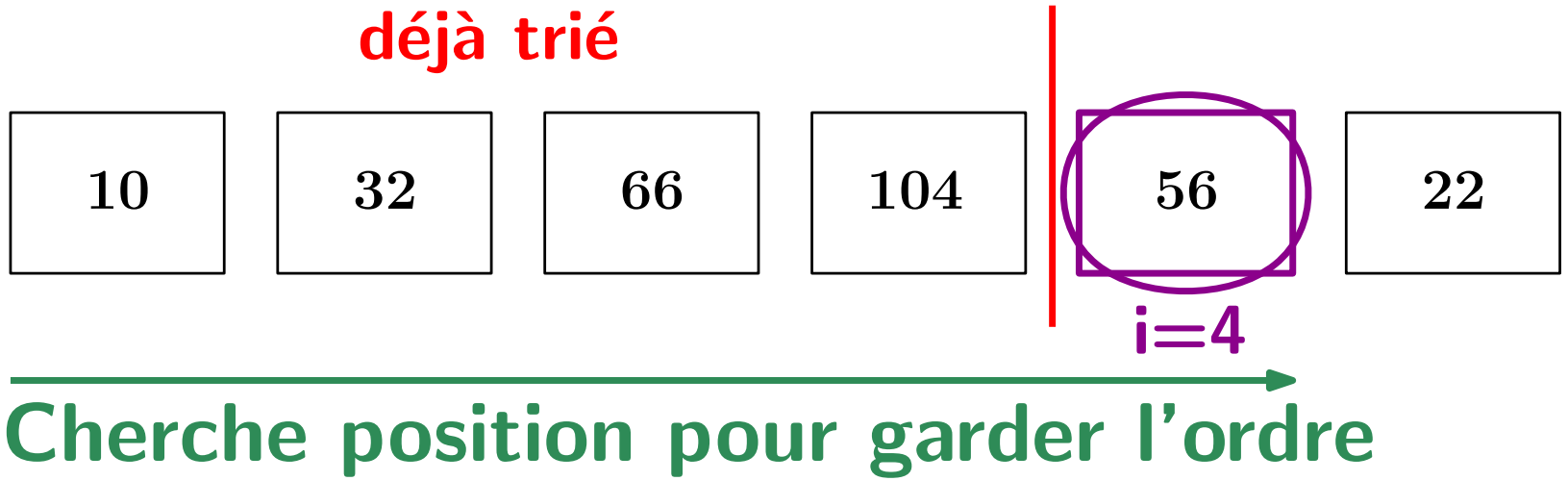
- Je considère la case i=3, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0, 1, 2 ou 3).
- Lorsque j'insère une carte dans la partie triée, je **décale toute la partie droite de l'ensemble déjà** trié d'un cran**.**
- Etc… jusqu'à ce que la partie déjà triée comprennent tout le tableau.



2) Tri par insertion

Principe: Les cases **0 à 2 forment déjà un ensemble trié.**

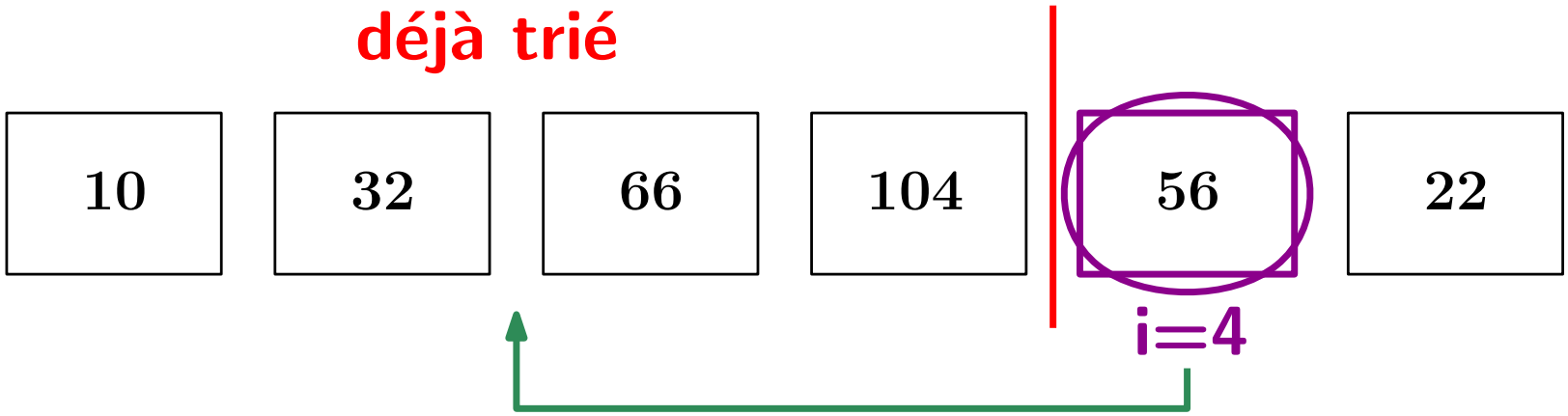
- Je considère la case i=3, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0, 1, 2 ou 3).
- Lorsque j'insère une carte dans la partie triée, je **décale toute la partie droite de l'ensemble déjà** trié d'un cran.
- Etc… jusqu'à ce que la partie déjà triée comprennent tout le tableau.



2) Tri par insertion

Principe: Les cases **0 à 2 forment déjà un ensemble trié.**

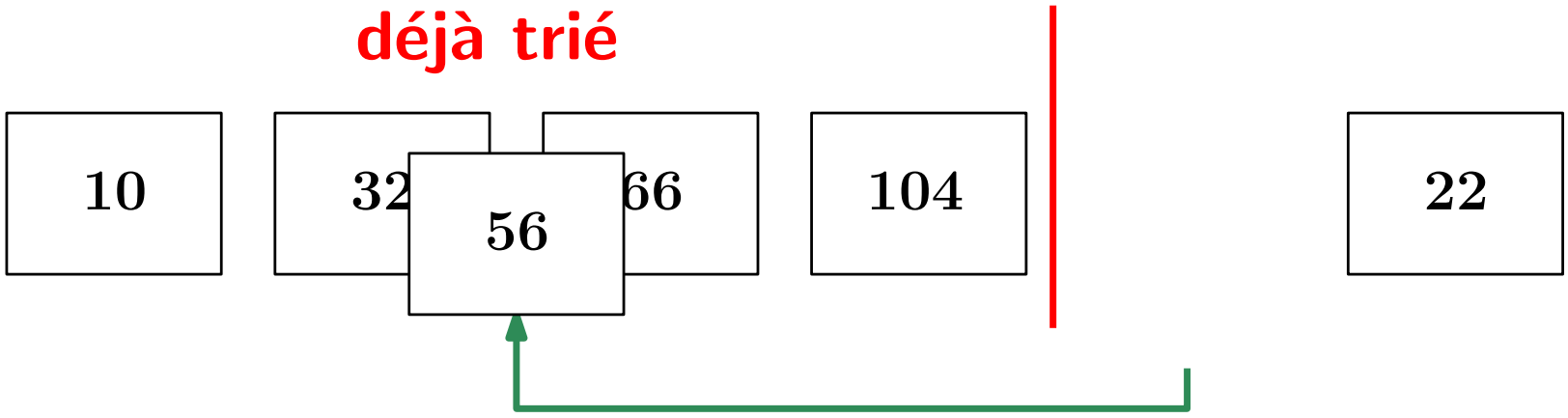
- Je considère la case i=3, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0, 1, 2 ou 3).
- Lorsque j'insère une carte dans la partie triée, je **décale toute la partie droite de l'ensemble déjà** trié d'un cran**.**
- Etc… jusqu'à ce que la partie déjà triée comprennent tout le tableau.



2) Tri par insertion

Principe: Les cases **0 à 2 forment déjà un ensemble trié.**

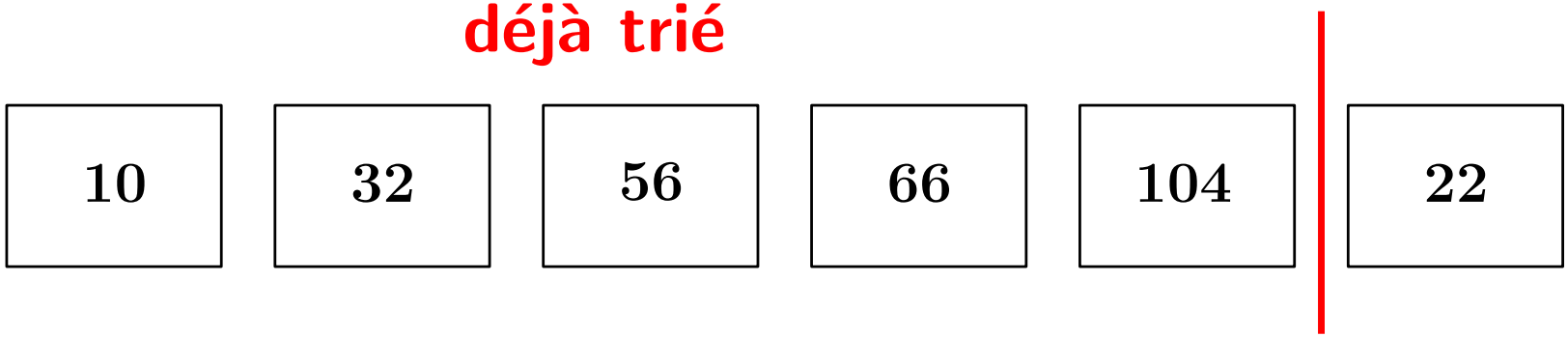
- Je considère la case i=3, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0, 1, 2 ou 3).
- Lorsque j'insère une carte dans la partie triée, je **décale toute la partie droite de l'ensemble déjà** trié d'un cran**.**
- Etc… jusqu'à ce que la partie déjà triée comprennent tout le tableau.



2) Tri par insertion

Principe: Les cases **0 à 2 forment déjà un ensemble trié.**

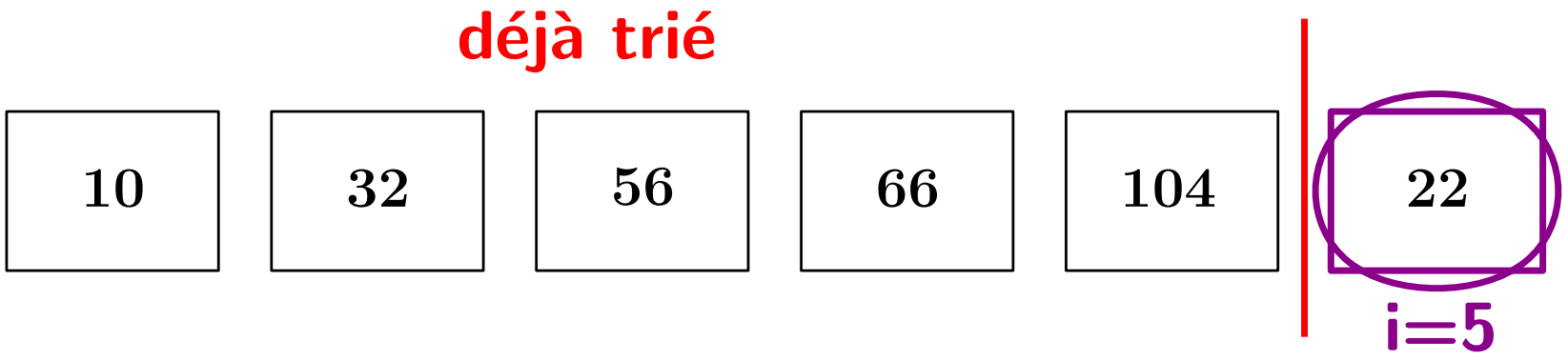
- Je considère la case i=3, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0, 1, 2 ou 3).
- Lorsque j'insère une carte dans la partie triée, je **décale toute la partie droite de l'ensemble déjà** trié d'un cran**.**
- Etc… jusqu'à ce que la partie déjà triée comprennent tout le tableau.



2) Tri par insertion

Principe: Les cases **0 à 2 forment déjà un ensemble trié.**

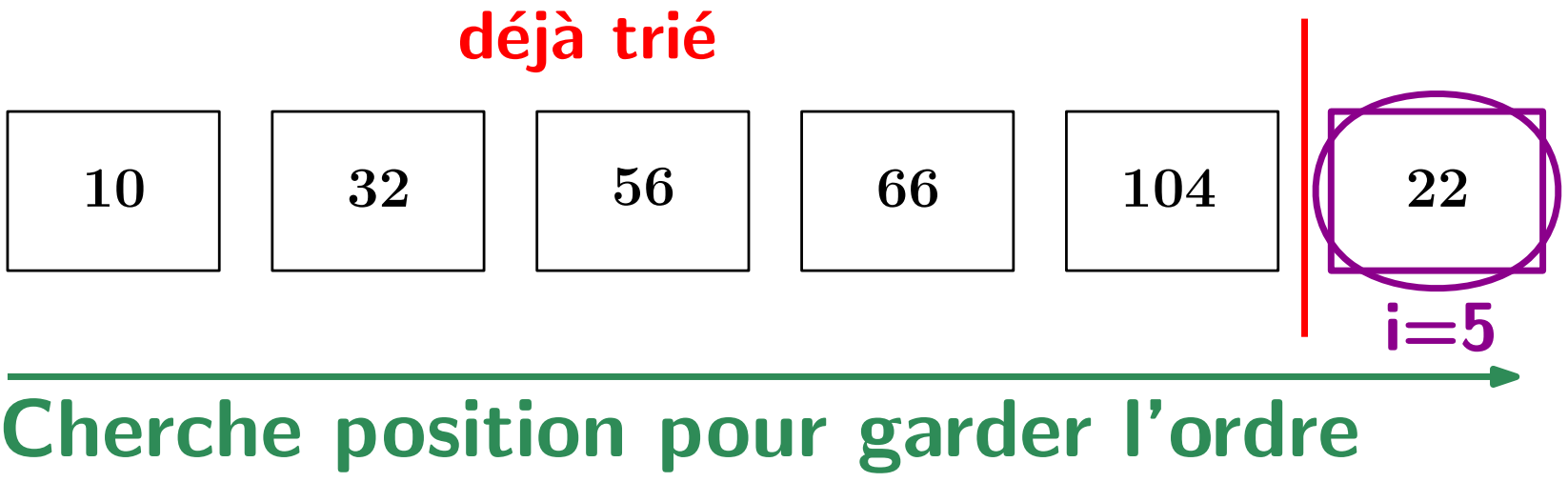
- Je considère la case i=3, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0, 1, 2 ou 3).
- Lorsque j'insère une carte dans la partie triée, je **décale toute la partie droite de l'ensemble déjà** trié d'un cran**.**
- Etc… jusqu'à ce que la partie déjà triée comprennent tout le tableau.



2) Tri par insertion

Principe: Les cases **0 à 2 forment déjà un ensemble trié.**

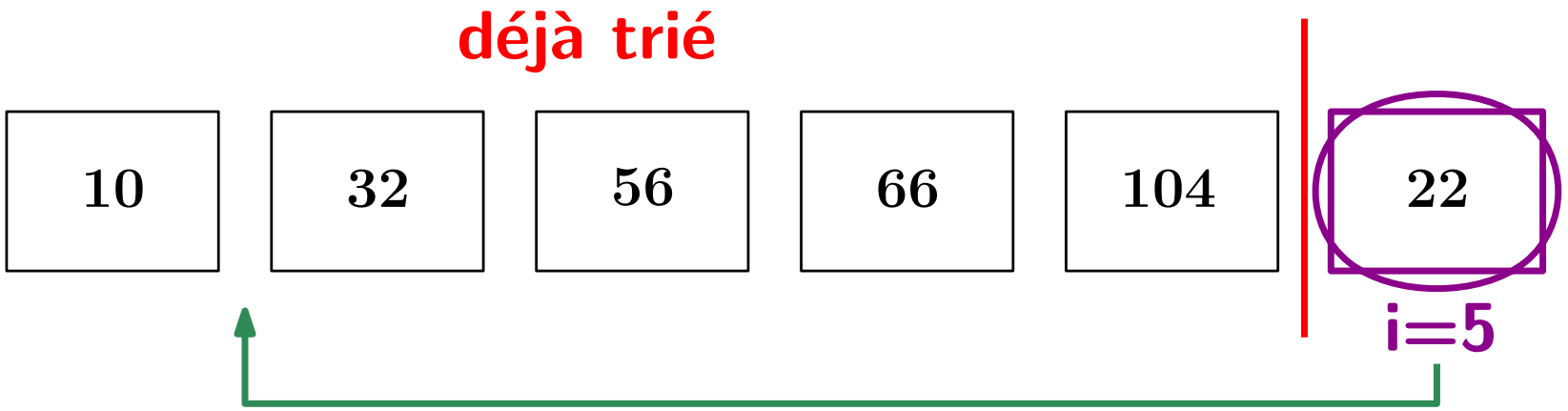
- Je considère la case i=3, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0, 1, 2 ou 3).
- Lorsque j'insère une carte dans la partie triée, je **décale toute la partie droite de l'ensemble déjà** trié d'un cran.
- Etc… jusqu'à ce que la partie déjà triée comprennent tout le tableau.



2) Tri par insertion

Principe: Les cases **0 à 2 forment déjà un ensemble trié.**

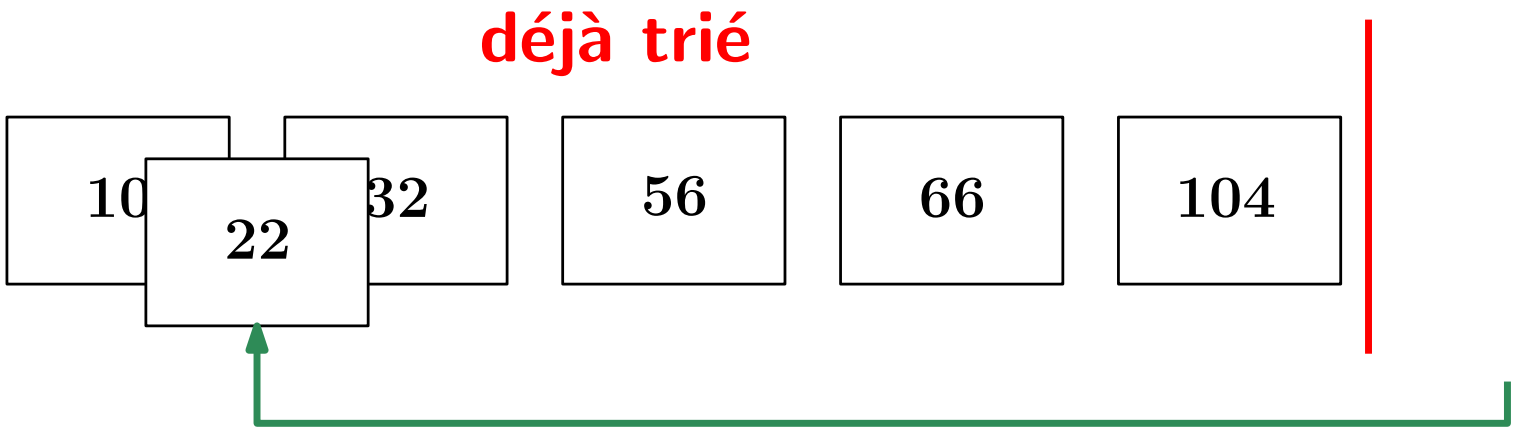
- Je considère la case i=3, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0, 1, 2 ou 3).
- Lorsque j'insère une carte dans la partie triée, je **décale toute la partie droite de l'ensemble déjà** trié d'un cran**.**
- Etc… jusqu'à ce que la partie déjà triée comprennent tout le tableau.



2) Tri par insertion

Principe: Les cases **0 à 2 forment déjà un ensemble trié.**

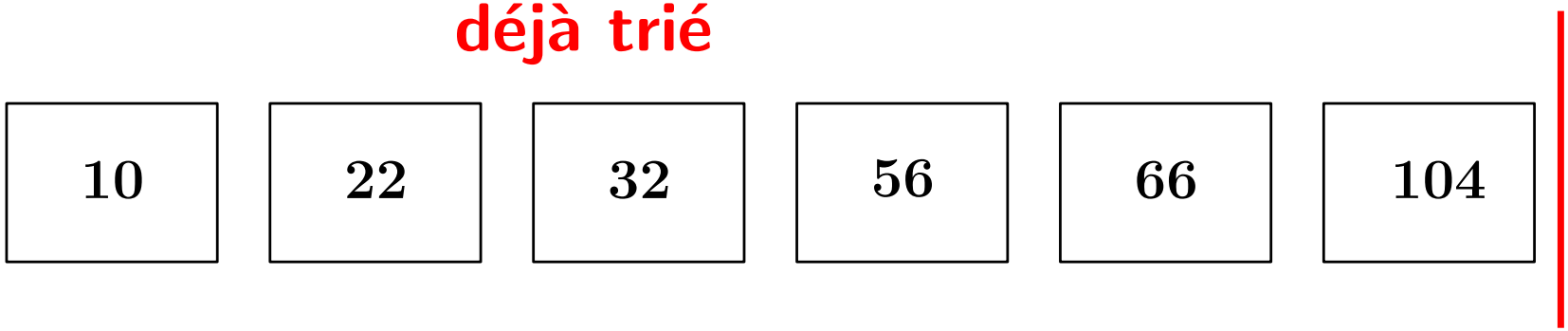
- Je considère la case i=3, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0, 1, 2 ou 3).
- Lorsque j'insère une carte dans la partie triée, je **décale toute la partie droite de l'ensemble déjà** trié d'un cran**.**
- Etc… jusqu'à ce que la partie déjà triée comprennent tout le tableau.



2) Tri par insertion

Principe: Les cases **0 à 2 forment déjà un ensemble trié.**

- Je considère la case i=3, et je veux **l'insérer dans la partie déjà triée** ou juste après (case 0, 1, 2 ou 3).
- Lorsque j'insère une carte dans la partie triée, je **décale toute la partie droite de l'ensemble déjà** trié d'un cran**.**
- Etc… jusqu'à ce que la partie déjà triée comprennent tout le tableau.



2) Tri par insertion

```
Proc insereDansCasesTriees(nombre tab[ ], nombre valeurAInserer,
                                                    int derniereCaseZone):
    /* En supposant que les cases numéro 0 à derniereCaseZone −1
     sont triées dans l'ordre croissant,  cette procédure insère la valeur
     au bon endroit entre les indices 0 et derniereCaseZone, en
     décalant toutes les valeurs suivantes d'un cran vers la droite */
    int i:=0
    Tant que tab[i]<valeurAInserer et i<derniereCaseZone :
        |    i++
    int positionOuInserer:=i
    Pour i de derniereCaseZone à positionOuInserer+1, par pas de -1:
        |    tab[i]:=tab[i-1]
    tab[positionOuInserer]:=valeurAInserer
```

2) Tri par insertion

```
Proc triInsertion(nombre tab[ ]) :
    int i
    nombre valeurAInserer
    Pour i de 1 à longu(tab)-1:
        valeurAInserer:=tab[i]
        insereDansCasesTriees(tab, valeurAInserer, i)
        /* Invariant de boucle: à cet endroit de l'algo., les cases
        numérotées 0 à i sont triées dans l'ordre croissant, et
        correspondent aux valeurs des cases 0 à i dans le tab. initial.
        Chaque nombre initialement présent dans le tableau apparaît
        toujours dans l'état actuel du tableau (le même nombre de fois
        qu'initialement). */
```

2) Tri par insertion

Invariant:

- il faut vérifier qu'il marche lors du **premier passage** (ici **i=1**)

```
Proc triInsertion(nombre tab[ ]) :
        int i
        nombre valeurAInserer
        Pour i de 1 à longu(tab)-1:
                valeurAInserer:=tab[i]
                insereDansCasesTriees(tab, valeurAInserer, i)
```

Invariant de boucle: les cases numérotées 0 à i sont triées dans l'ordre croissant, et correspondent aux valeurs des cases 0 à i dans le tab. initial + conservation des valeurs

```
                /* Les cases num. 0 et 1 sont triées dans l'ordre croissant
                et sont les valeurs des cases 0 et 1 initialement*/
```

2) Tri par insertion

Invariant:

- il faut vérifier que, s'il est **vrai à un certain passage**, alors il est également vrai au **passage suivant** (≈ récurrence).

```
Proc triInsertion(nombre tab[ ]) :
        int i
        nombre valeurAInserer
        Pour i de 1 à longu(tab)-1:
                /* On sait que les cases num 0 à i-1 sont triées dans
               l'ordre croissant, et sont les i 1e valeurs du tab. initial*/
                valeurAInserer:=tab[i]
                insereDansCasesTriees(tab, valeurAInserer, i)
                /* Maintenant on a ajouté le contenu de tab[i] en
                 gardant l'ordre croissant sur les cases num 0 à i */
```

2) Tri par insertion

Invariant:

- il faut vérifier que, **lors du dernier passage, l'invariant soit en accord avec ce que nous cherchons** pour notre algorithme.

```
Proc triInsertion(nombre tab[ ]) :
        int i
        nombre valeurAInserer
        Pour i de 1 à longu(tab)-1:
                valeurAInserer:=tab[i]
                insereDansCasesTriees(tab, valeurAInserer, i)
                /* Dernier passage: les cases 0 à longu(tab)-1 sont
                 triées dans l'ordre croissant et correspondent aux
                 valeurs des longu(tab) 1e cases du tableau initial
                → tout est là, dans l'ordre croissant!*/
```

2) Tri par insertion

Complexité: on appelle *n* la taille du tableau, et on compte les op. élémentaires de tableau.

```
Proc insereDansCasesTriees(nombre tab[ ], nombre valeurAInserer,
                                                    int derniereCaseZone):
        int i:=0
        Tant que tab[i]<valeurAInserer et i<derniereCaseZone :
        |       i++
        int positionOuInserer:=i
        Pour i de derniereCaseZone à positionOuInserer+1, par pas de -1:
        |       tab[i]:=tab[i-1]
        tab[positionOuInserer]:=valeurAInserer
```

Complexité: O(*n*)

2) Tri par insertion

```
Proc triInsertion(nombre tab[ ]) :
        int i
        nombre valeurAInserer
        Pour i de 1 à longu(tab)-1:
                valeurAInserer:=tab[i]
                insereDansCasesTriees(tab, valeurAInserer, i)
```

1

O(n)

n-1 fois

Total: $(n-1)\times(O(n)+1)$

$\approx (n-1)\times(n+1)$

$\rightarrow O(n^2)$ quadratique