INTRODUCTION À L’INFORMATIQUE CM4

Antonio E. Porreca
https://aeporreca.org/teaching

RECHERCHE DANS UN TABLEAU

```
fonction chercher(x,T)
    n ≔ longueur(T)
    i ≔ 0
    tant que i < n faire
        si T[i] = x alors
            retourner i
        i ≔ i + 1
    retourner –1
```

RECHERCHE LINÉAIRE

recherche de 33

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

RECHERCHE LINÉAIRE

recherche de 33

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

RECHERCHE LINÉAIRE

recherche de 33

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

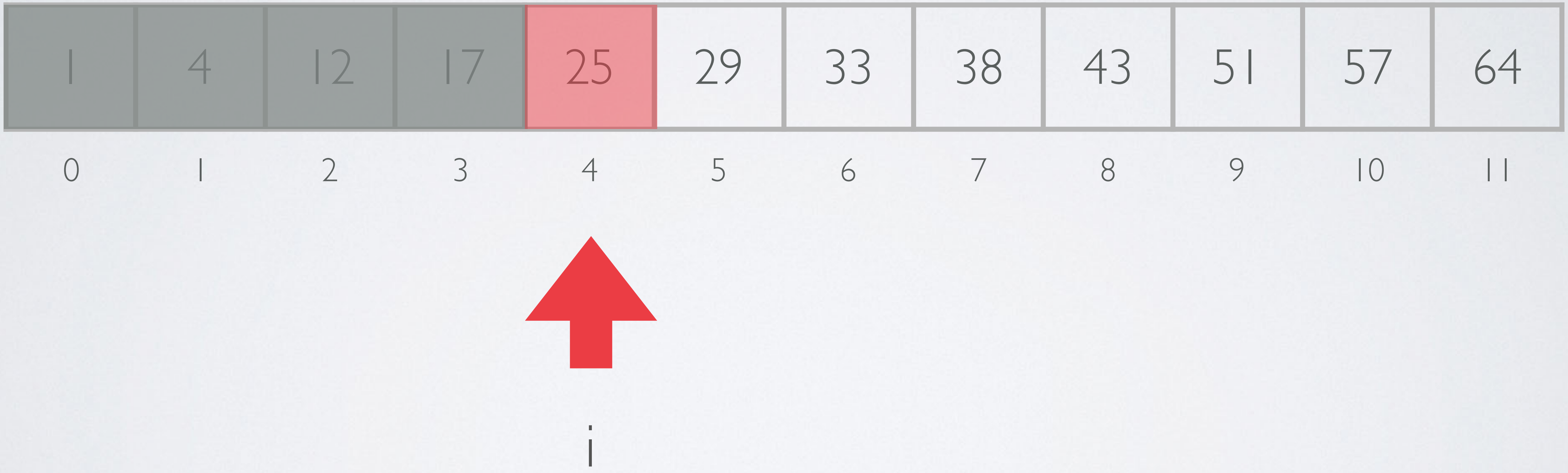
RECHERCHE LINÉAIRE

recherche de 33

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

RECHERCHE LINÉAIRE

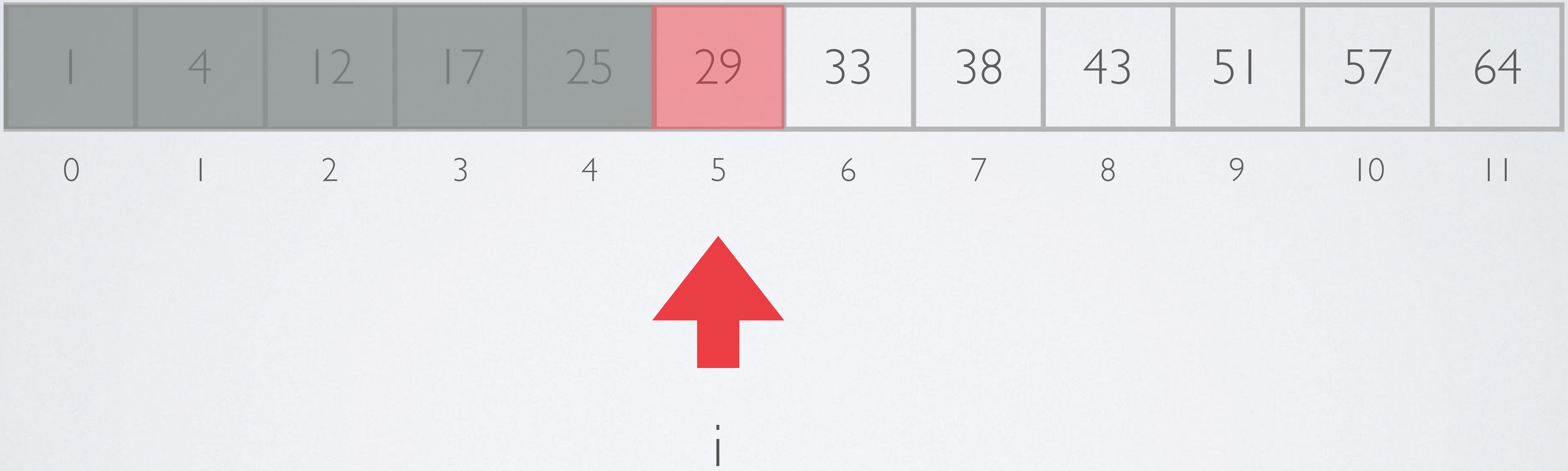
recherche de 33



1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

RECHERCHE LINÉAIRE

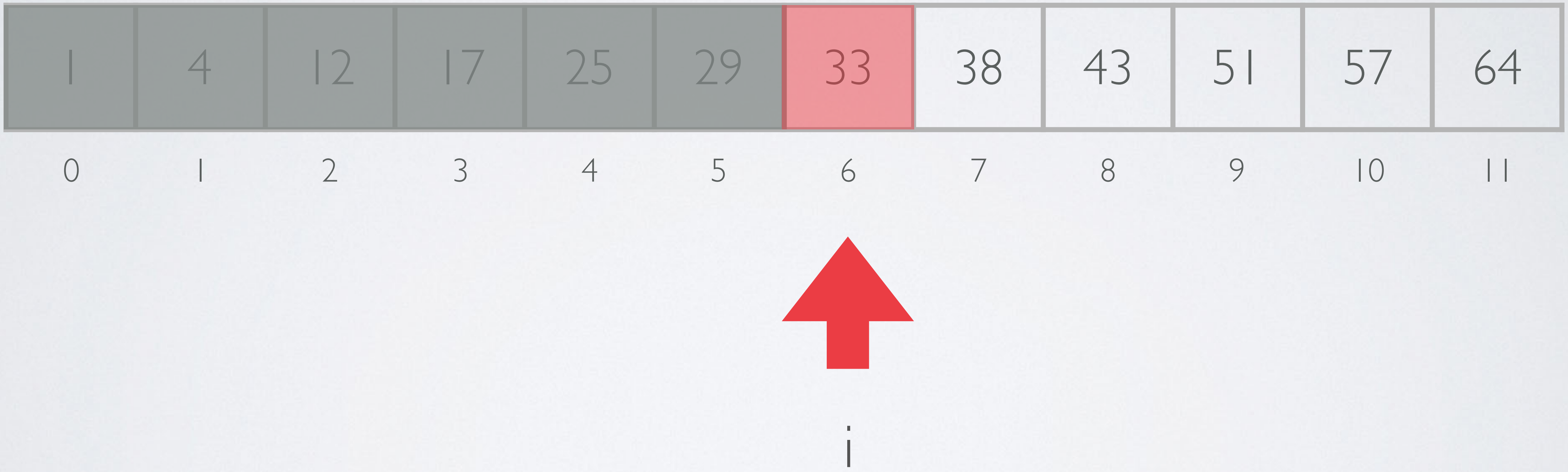
recherche de 33



1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

RECHERCHE LINÉAIRE

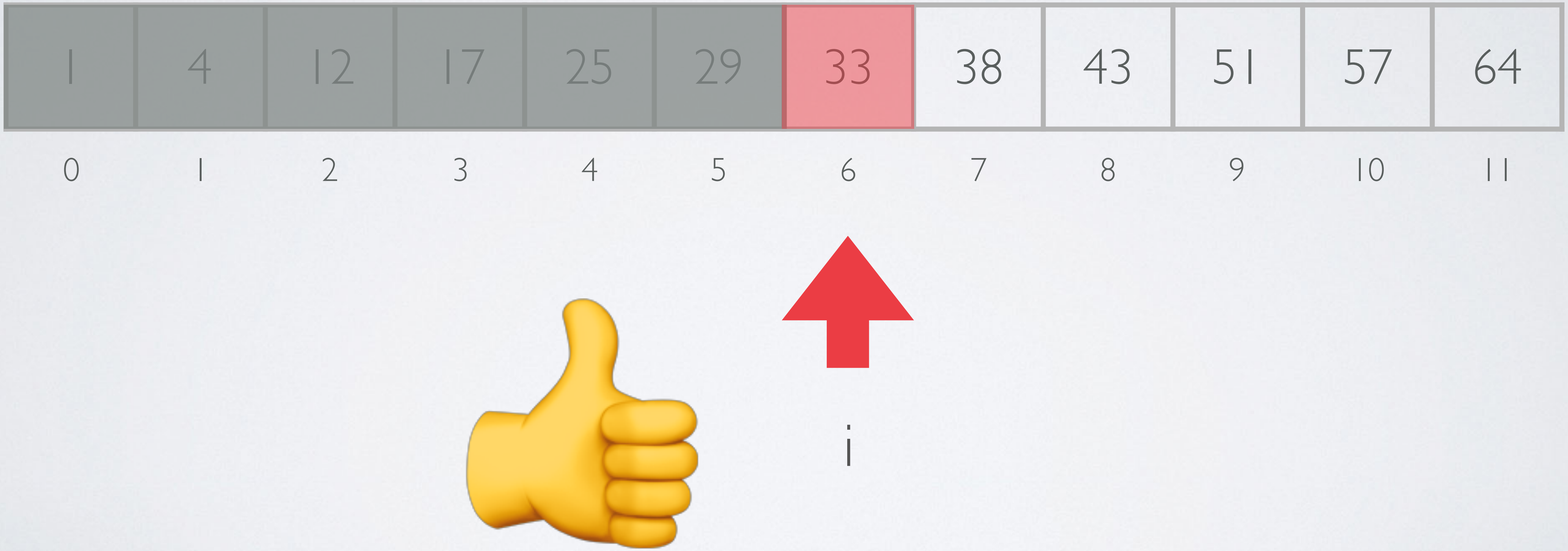
recherche de 33



1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

RECHERCHE LINÉAIRE

recherche de 33



1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

RECHERCHE DANS UN TABLEAU

```
fonction chercher(x, T)
    n ≔ longueur(T)
    i ≔ 0
    tant que i < n faire
        si T[i] = x alors
            retourner i
        i ≔ i + 1
    retourner –1
```

Terminaison ?

Correction ?

Efficacité ?

TERMINAISON

```
fonction chercher(x,T)
    n ≔ longueur(T)
    i ≔ 0
    tant que i < n faire
        si T[i] = x alors
            retourner i
        i ≔ i + 1
    retourner –1
```

- Au début, on a i = 0
- Il reste toujours n – i positions à examiner
- i est incrémenté à chaque itération
- Tôt ou tard on trouve x ou on arrive à i = n, et l'algorithme termine

CORRECTION

```
fonction chercher(x, T)
    n ≔ longueur(T)
    i ≔ 0
    tant que i < n faire
        si T[i] = x alors
            retourner i
        i ≔ i + 1
    retourner –1
```

- Si x est dans le tableau, il se trouve dans le sous-tableau T[i, …, n – 1]
  - C'est vrai au début de l'algorithme
  - Ça reste vrai à chaque itération de la boucle, parce qu'on vérifie toujours si T[i] = x
- Si on sort de la boucle parce que i = n, alors si x est dans le tableau, il est dans le sous-tableau vide T[n, n – 1], c'est à dire qu'il n'est pas là

EFFICACITÉ

```
fonction chercher(x, T)
    n := longueur(T)
    i := 0
    tant que i < n faire
        si T[i] = x alors
            retourner i
        i := i + 1
    retourner –1
```

- Si on a de la chance, on a T[0] = x et on termine tout de suite en 5 opérations

- Si T[k] = x on fait
  2 + 3(k + 1) opérations

- Si x n'est pas là on fait 2 + 3n + 2 = 3n + 4 opérations

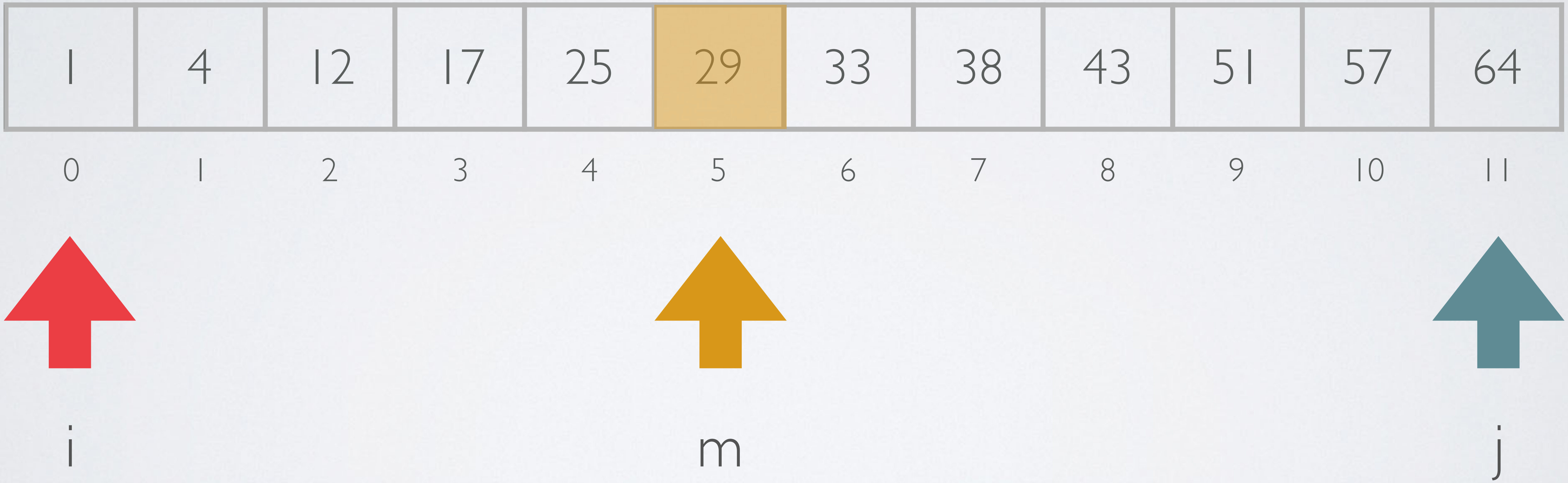
- Dans le pire des cas, on fait donc O(n) opérations : temps linéaire

RECHERCHE DICHOTOMIQUE DANS UN TABLEAU D'ENTIERS TRIÉ

```
fonction chercher(x, T)
    n ≔ longueur(T)
    i ≔ 0
    j ≔ n – 1
    tant que i ≤ j faire
        m ≔(i + j) ÷ 2
        si T[m] = x alors
            retourner m
        sinon si x < T[m] alors
            j ≔ m – 1
        sinon
            i ≔ m + 1
    retourner –1
```

RECHERCHE DICHOTOMIQUE

recherche de 33



1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

RECHERCHE DICHOTOMIQUE

recherche de 33

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

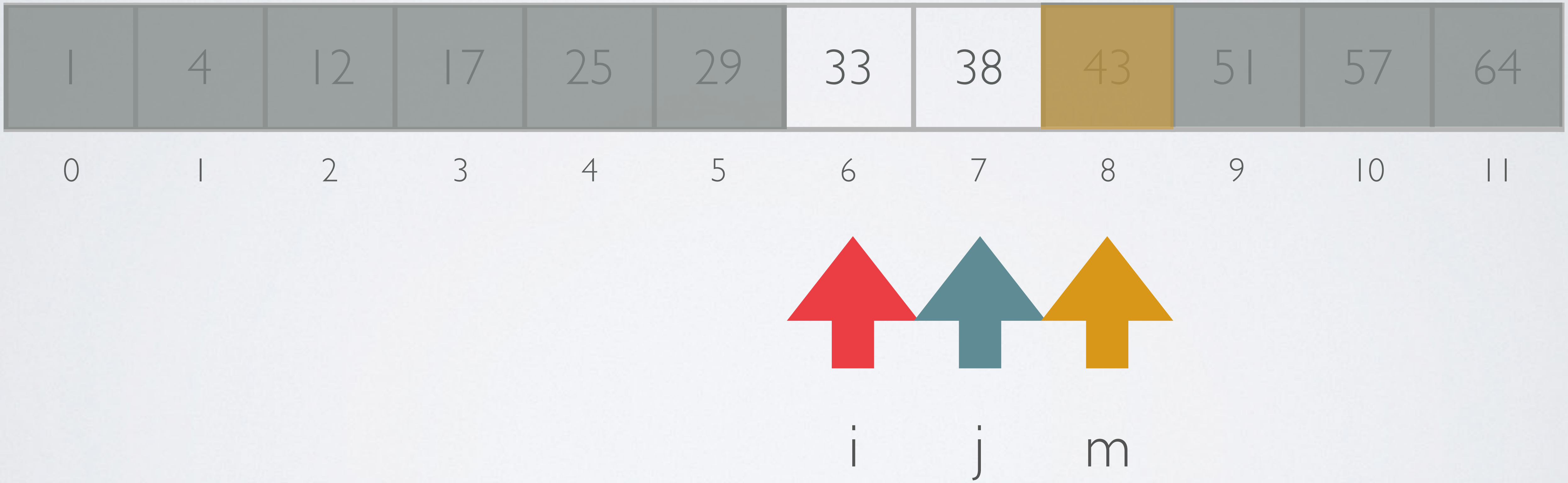
RECHERCHE DICHOTOMIQUE

recherche de 33

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

RECHERCHE DICHOTOMIQUE

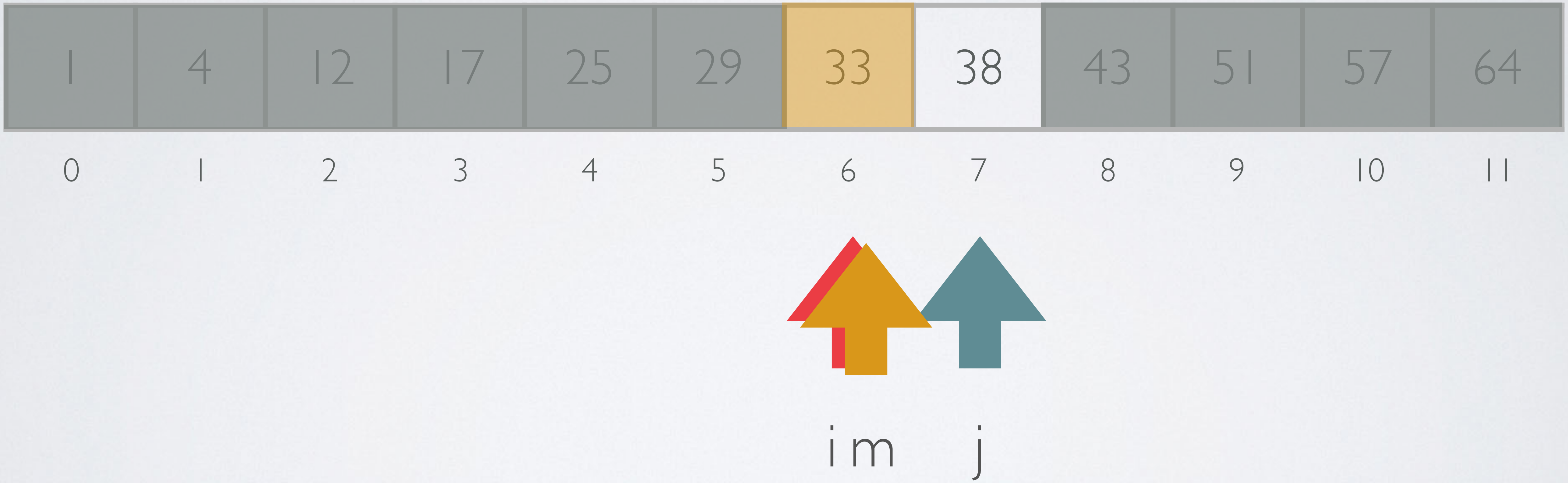
recherche de 33



1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

RECHERCHE DICHOTOMIQUE

recherche de 33



1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

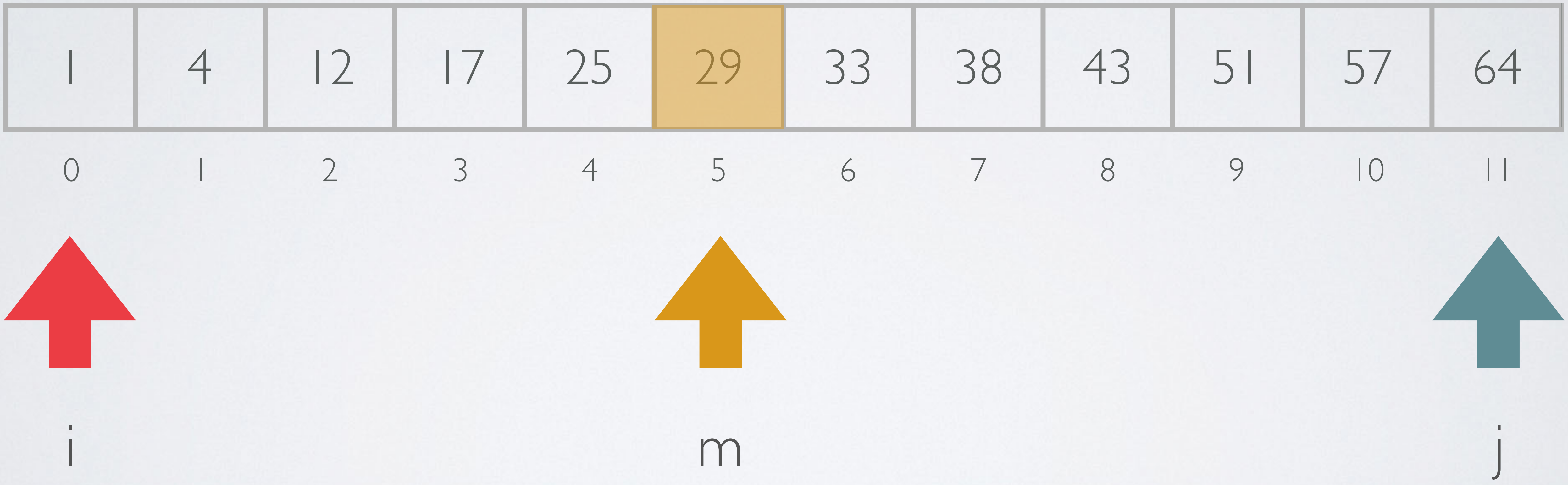
RECHERCHE DICHOTOMIQUE

recherche de 33

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

RECHERCHE DICHOTOMIQUE

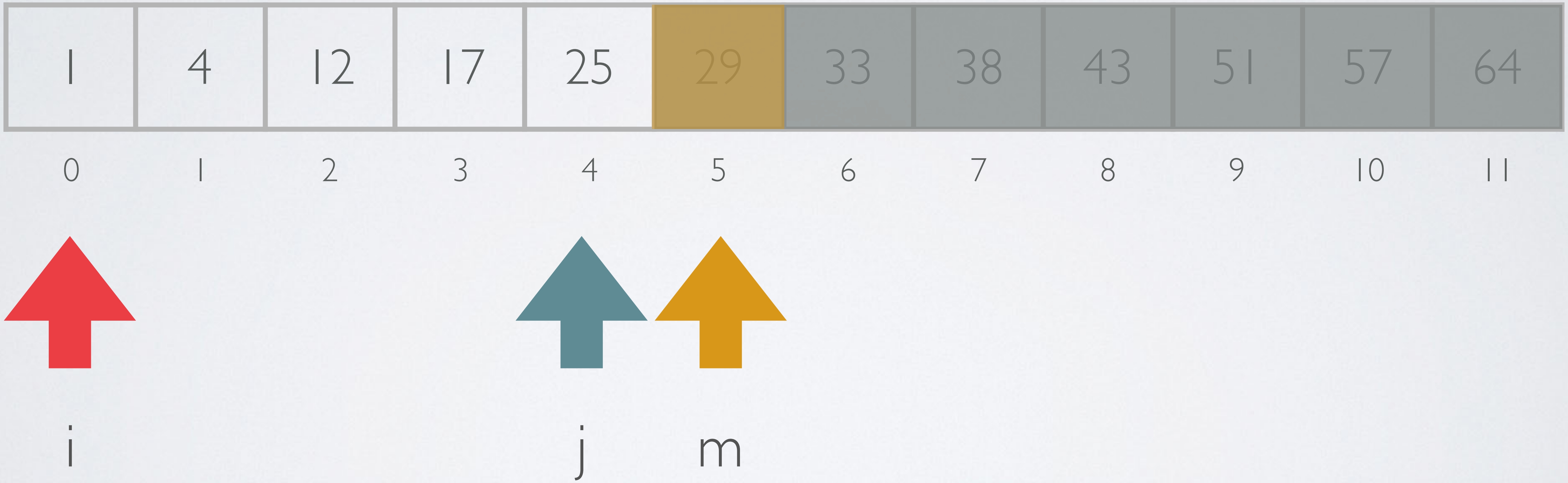
recherche de 16



0	1	2	3	4	5	6	7	8	9	10	11
1	4	12	17	25	29	33	38	43	51	57	64

RECHERCHE DICHOTOMIQUE

recherche de 16



1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

RECHERCHE DICHOTOMIQUE

recherche de 16

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

RECHERCHE DICHOTOMIQUE

recherche de 16

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

m i j

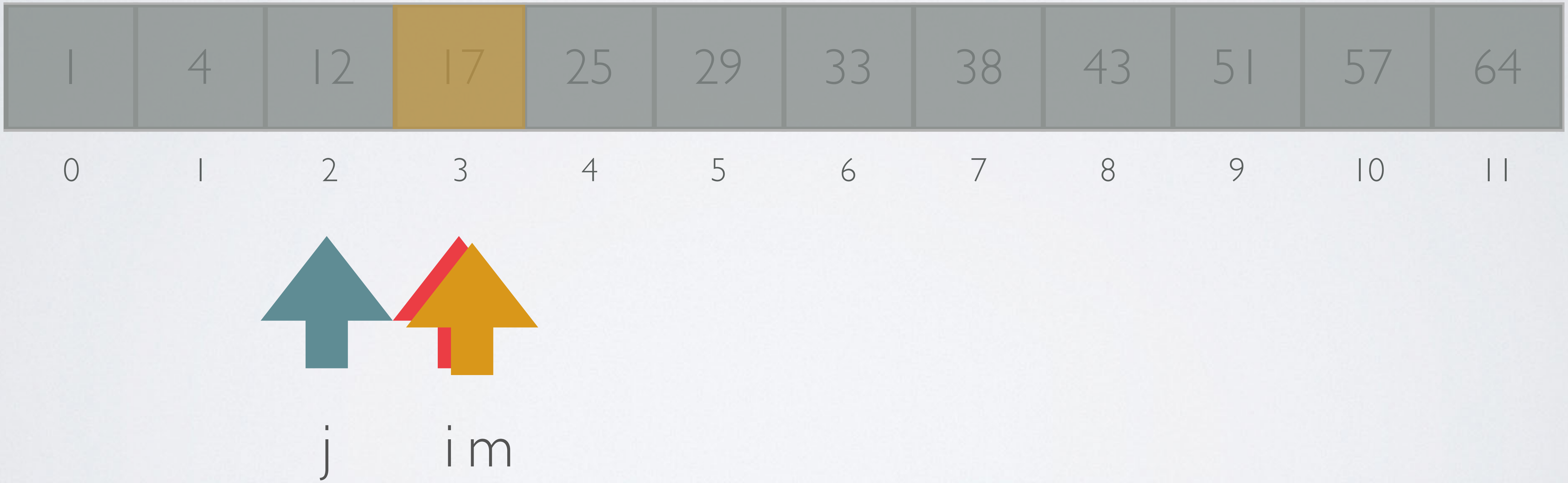
RECHERCHE DICHOTOMIQUE

recherche de 16

1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

RECHERCHE DICHOTOMIQUE

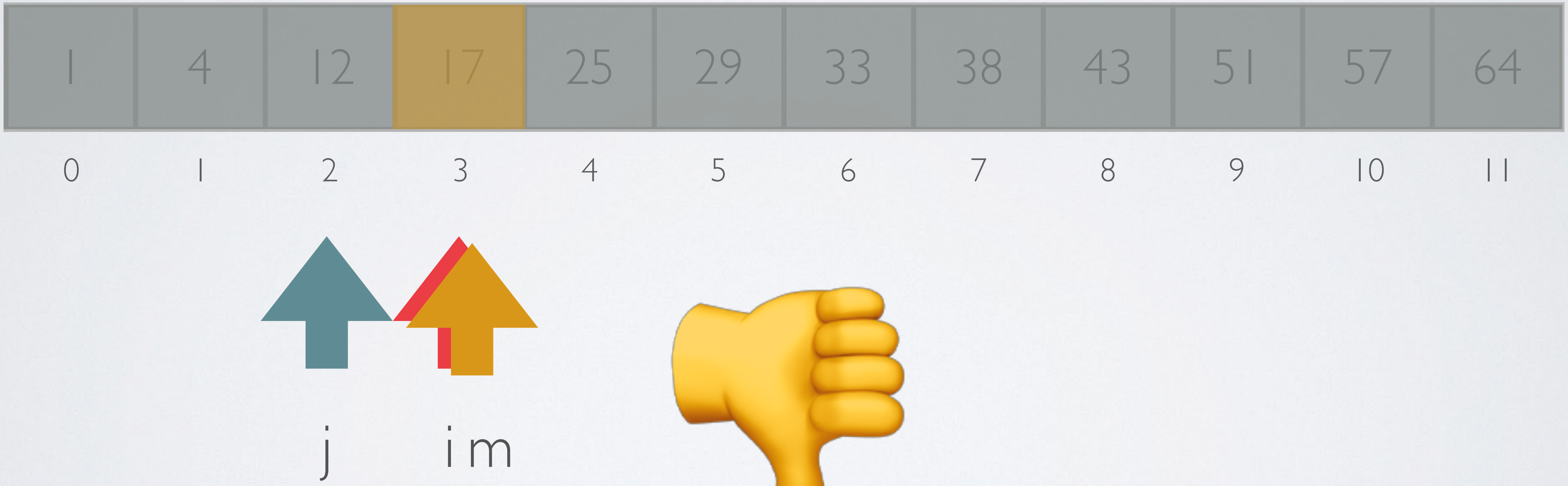
recherche de 16



1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

RECHERCHE DICHOTOMIQUE

recherche de 16



1	4	12	17	25	29	33	38	43	51	57	64
0	1	2	3	4	5	6	7	8	9	10	11

👎

RECHERCHE DICHOTOMIQUE DANS UN TABLEAU D'ENTIERS TRIÉ

```
fonction chercher(x,T)
    n ≔ longueur(T)
    i ≔ 0
    j ≔ n – 1
    tant que i ≤ j faire
        m ≔(i + j) ÷ 2
        si T[m] = x alors
            retourner m
        sinon si x < T[m] alors
            j ≔ m – 1
        sinon
            i ≔ m + 1
    retourner –1
```

Terminaison ?

Correction ?

Efficacité ?

TERMINAISON

```
fonction chercher(x,T)
    n ≔ longueur(T)
    i ≔ 0
    j ≔ n – 1
    tant que i ≤ j faire
        m ≔(i + j) ÷ 2
        si T[m] = x alors
            retourner m
        sinon si x < T[m] alors
            j ≔ m – 1
        sinon
            i ≔ m + 1
    retourner –1
```

- Si x est dans le tableau, il se trouve dans le sous-tableau T[i, …, j]
  - C'est vrai au début de l'algorithme
  - Ça reste vrai à chaque itération de la boucle, parce qu'on vérifie toujours si T[m] = x ou T[m] > x ou T[m] < x
- Si on sort de la boucle parce que i ≥ j, alors si x est dans le tableau, il est dans le sous-tableau vide T[i, j], c'est à dire qu'il n'est pas là

CORRECTION

```
fonction chercher(x,T)
    n ≔ longueur(T)
    i ≔ 0
    j ≔ n – 1
    tant que i ≤ j faire
        m ≔(i + j) ÷ 2
        si T[m] = x alors
            retourner m
        sinon si x < T[m] alors
            j ≔ m – 1
        sinon
            i ≔ m + 1
    retourner –1
```

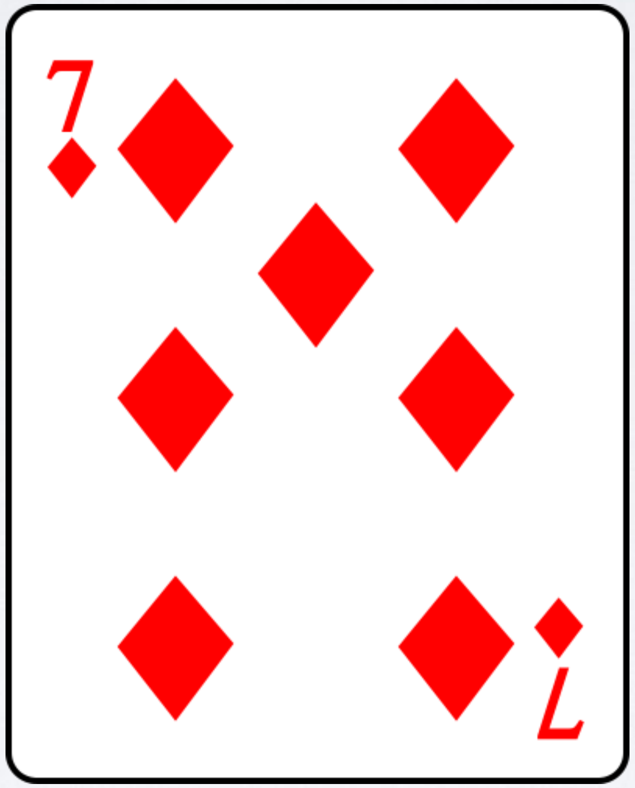
- Il reste toujours j – i + 1 éléments à examiner
- À chaque itération, on élimine approx. la moitié des éléments qui restent
- Tôt ou tard on trouve x, ou on reste sans éléments, et l'algorithme termine

EFFICACITÉ

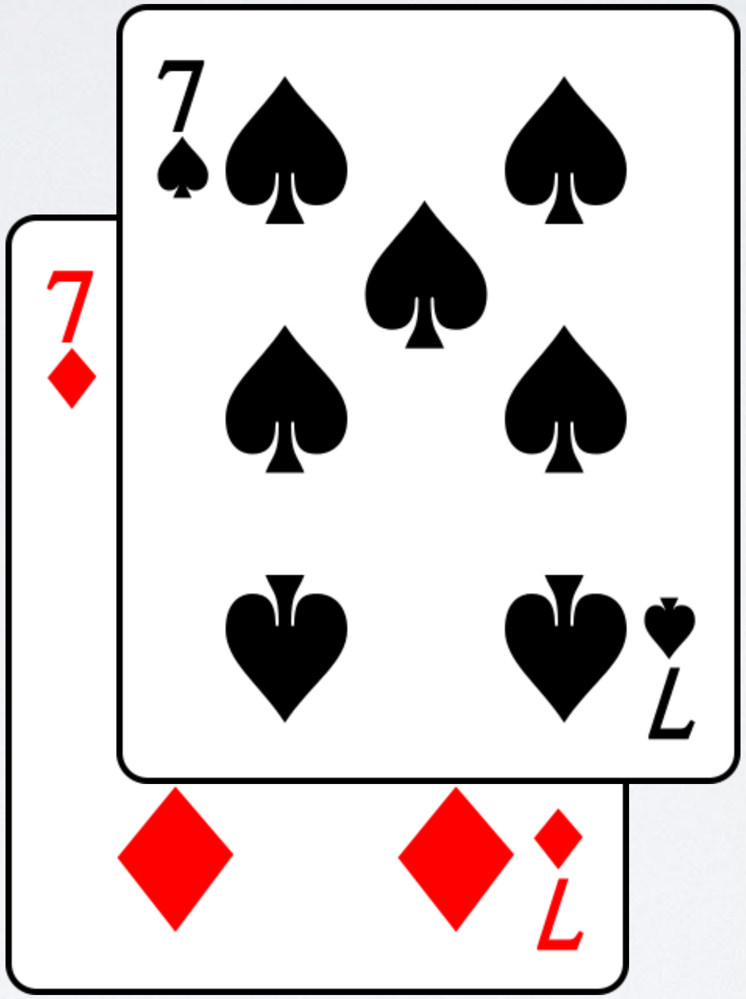
```
fonction chercher(x,T)
    n ≔ longueur(T)
    i ≔ 0
    j ≔ n – 1
    tant que i ≤ j faire
        m ≔(i + j) ÷ 2
        si T[m] = x alors
            retourner m
        sinon si x < T[m] alors
            j ≔ m – 1
        sinon
            i ≔ m + 1
    retourner –1
```

- Dans le pire des cas, x n’est pas là
- Comme on élimine à chaque itération la moitié du tableau, on exécute la boucle $\log_2 n$ fois au maximum
- Ça fait $O(\log_2 n)$ opérations

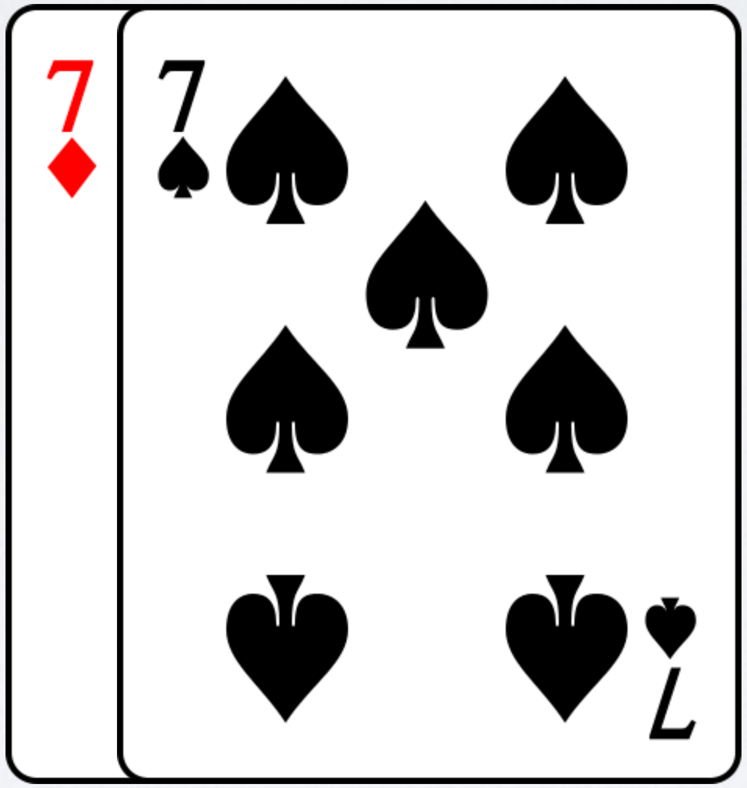
TRI PAR INSERTION



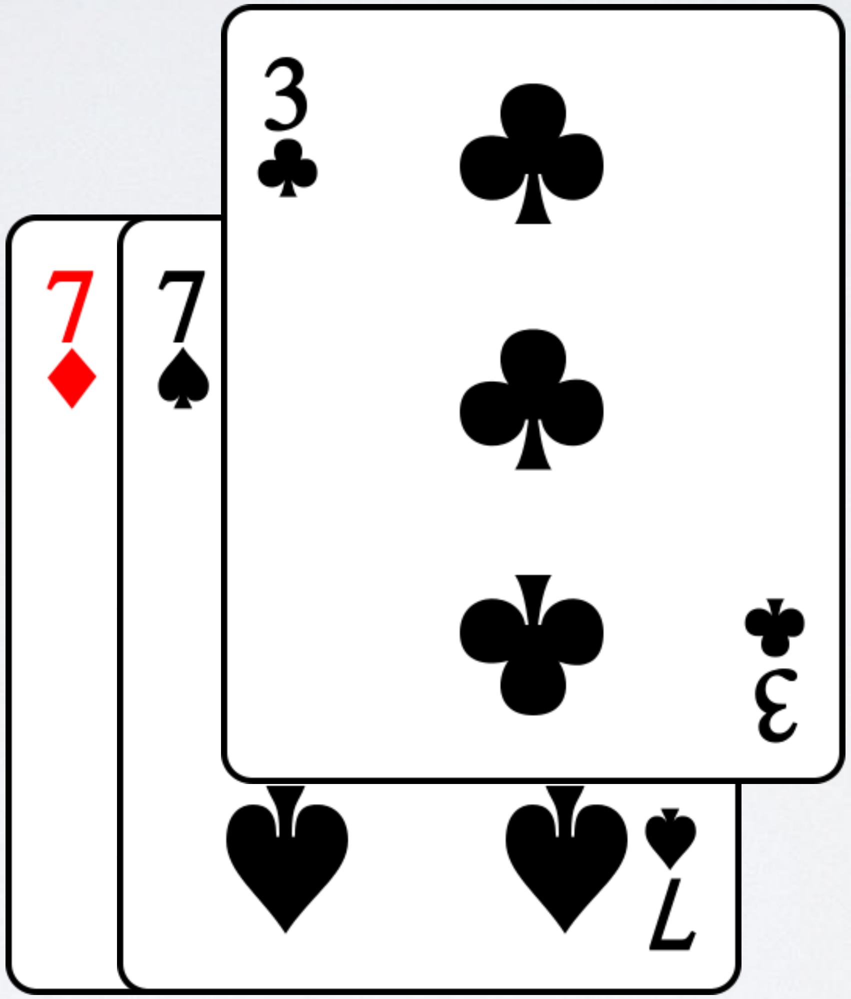
TRI PAR INSERTION



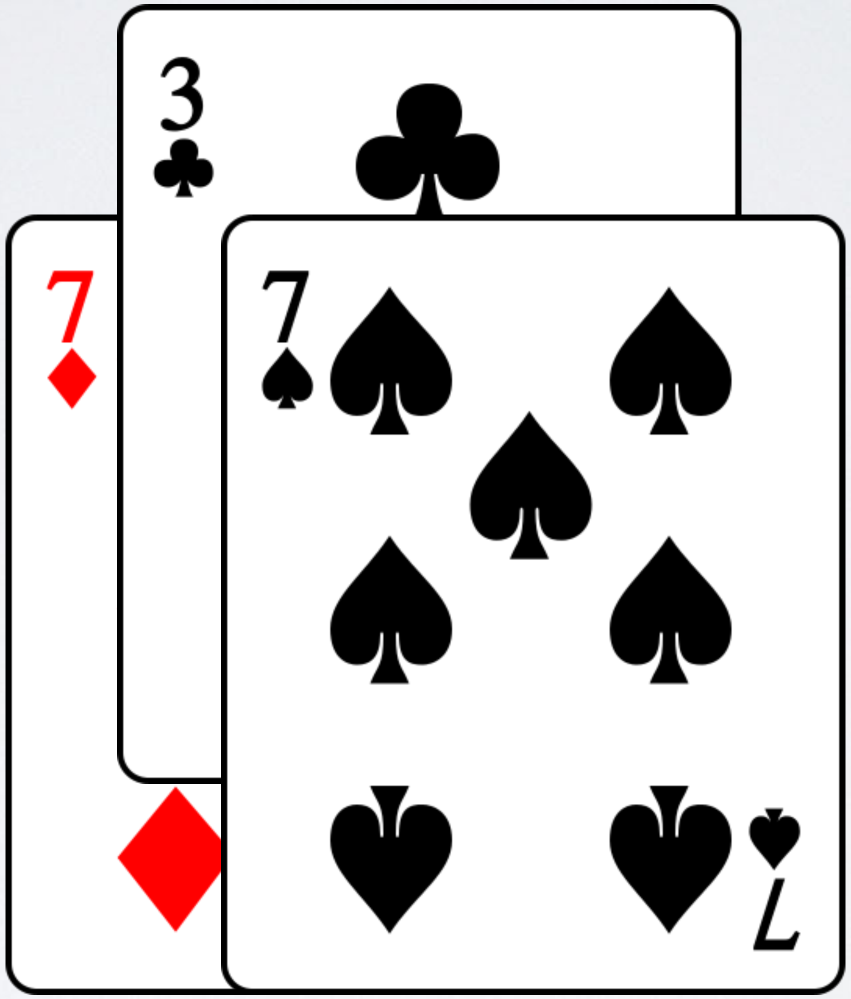
TRI PAR INSERTION



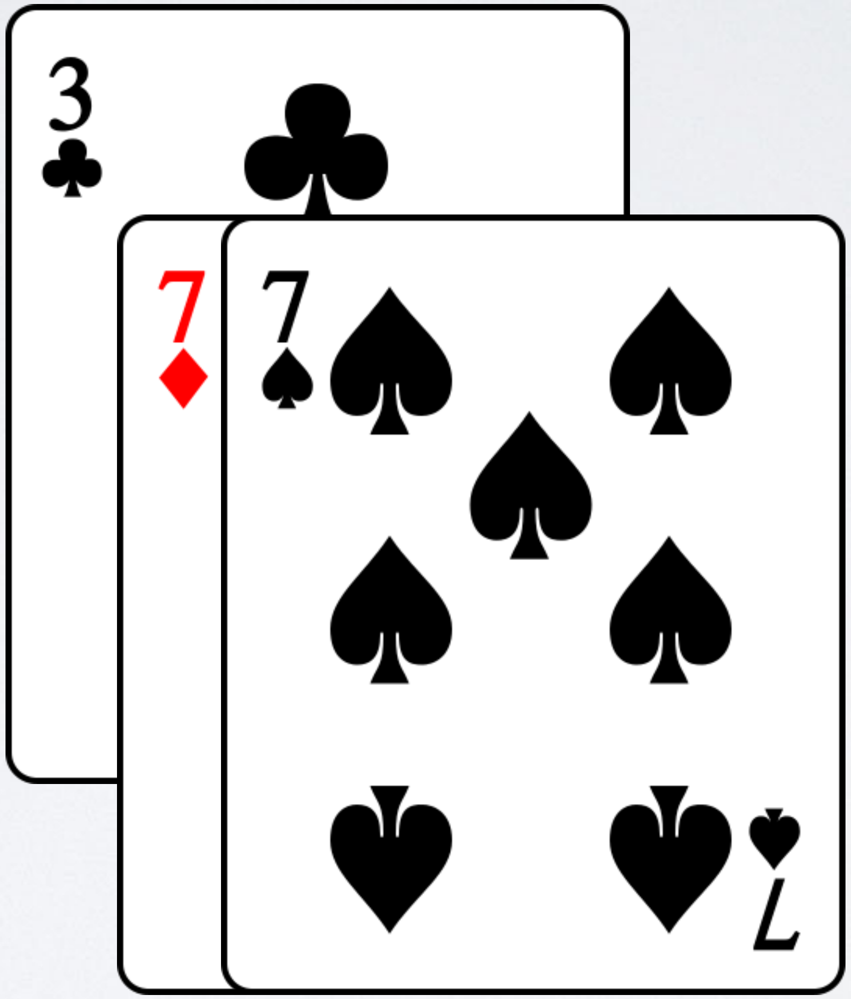
TRI PAR INSERTION



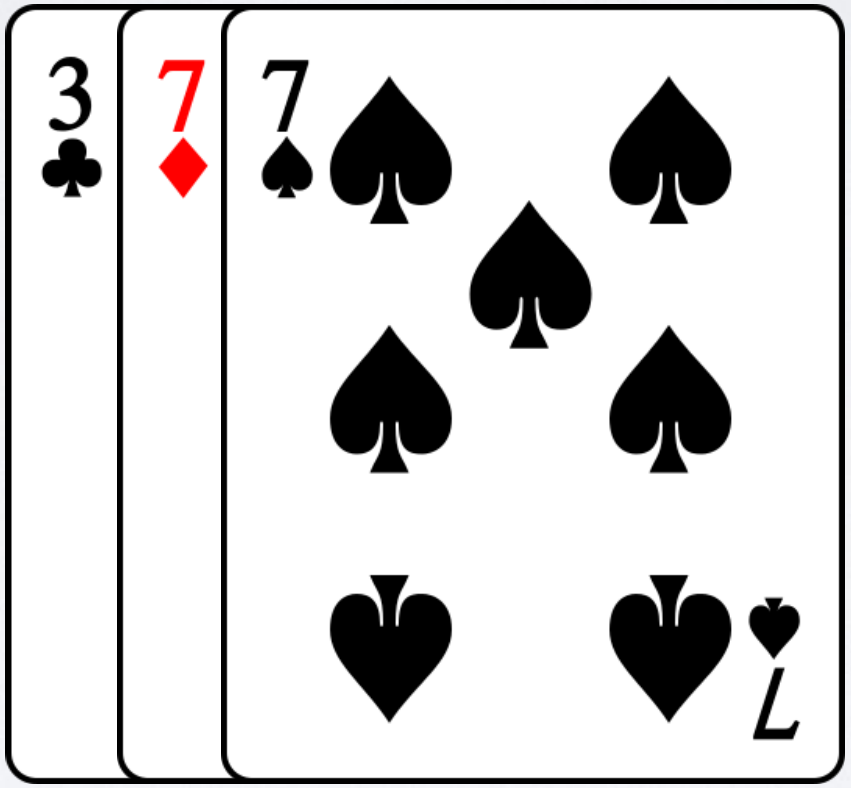
TRI PAR INSERTION



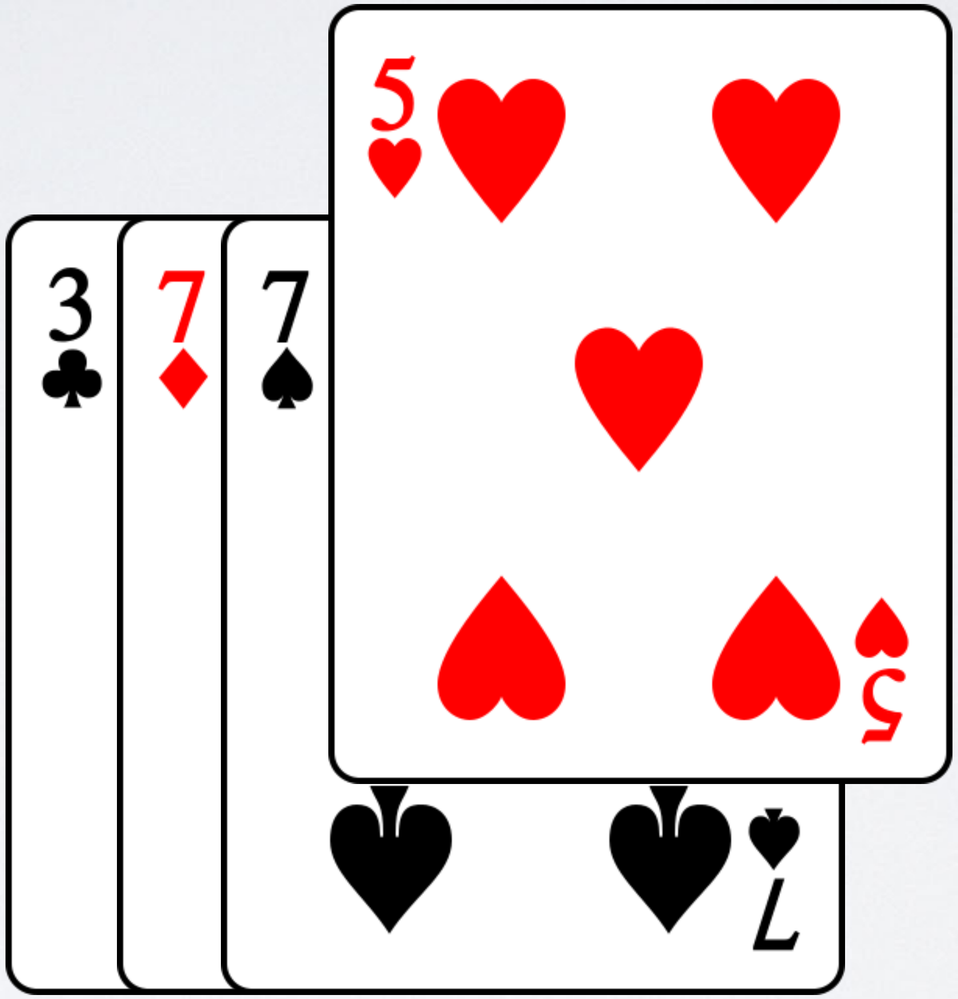
TRI PAR INSERTION



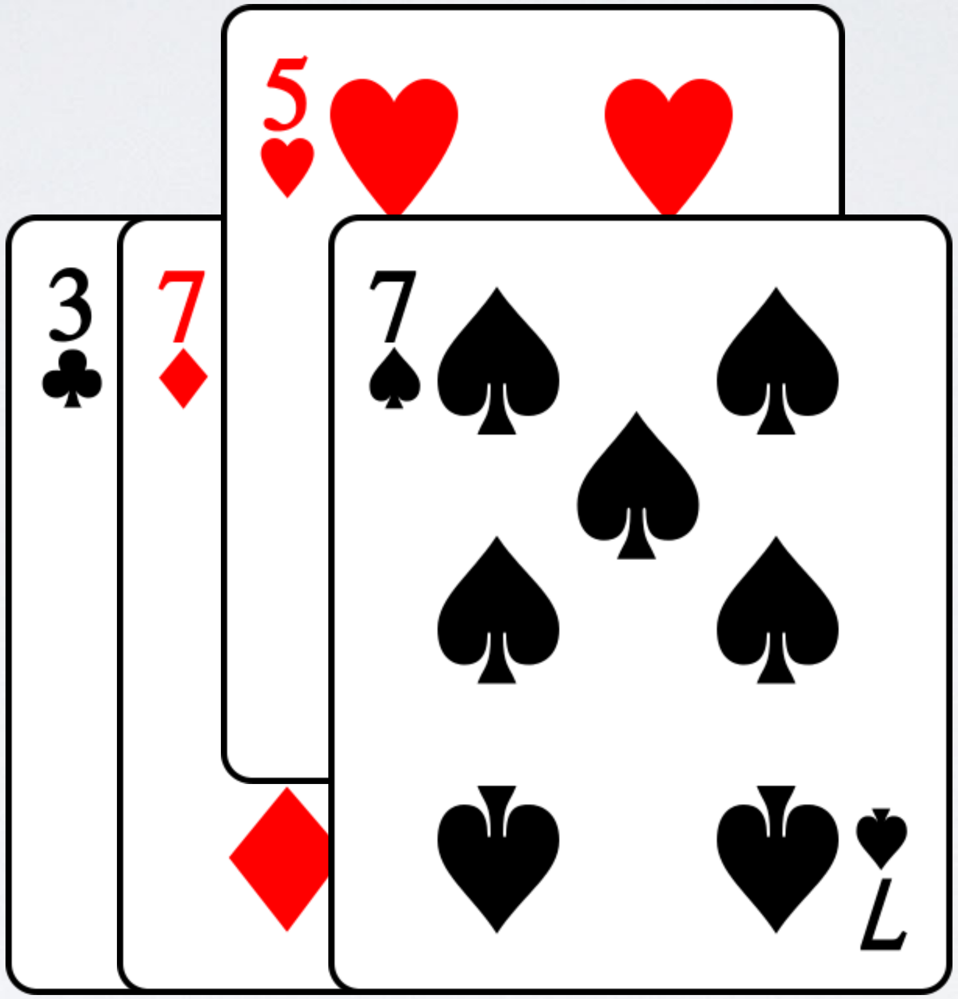
TRI PAR INSERTION



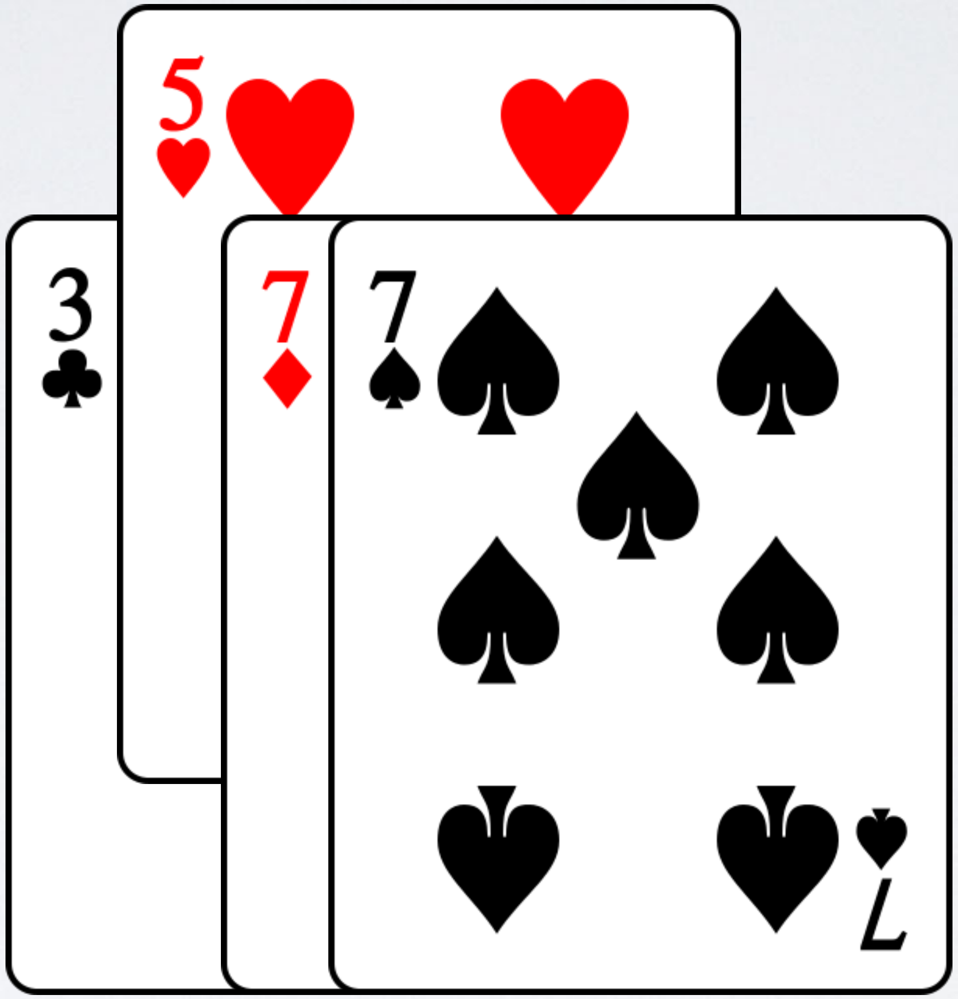
TRI PAR INSERTION



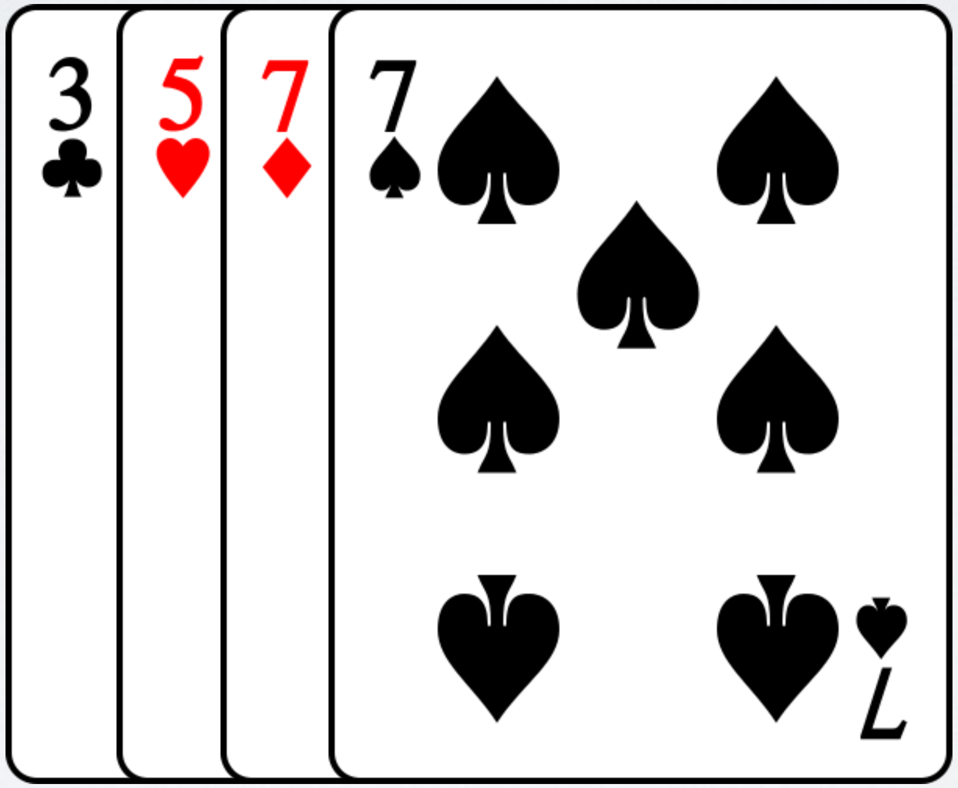
TRI PAR INSERTION



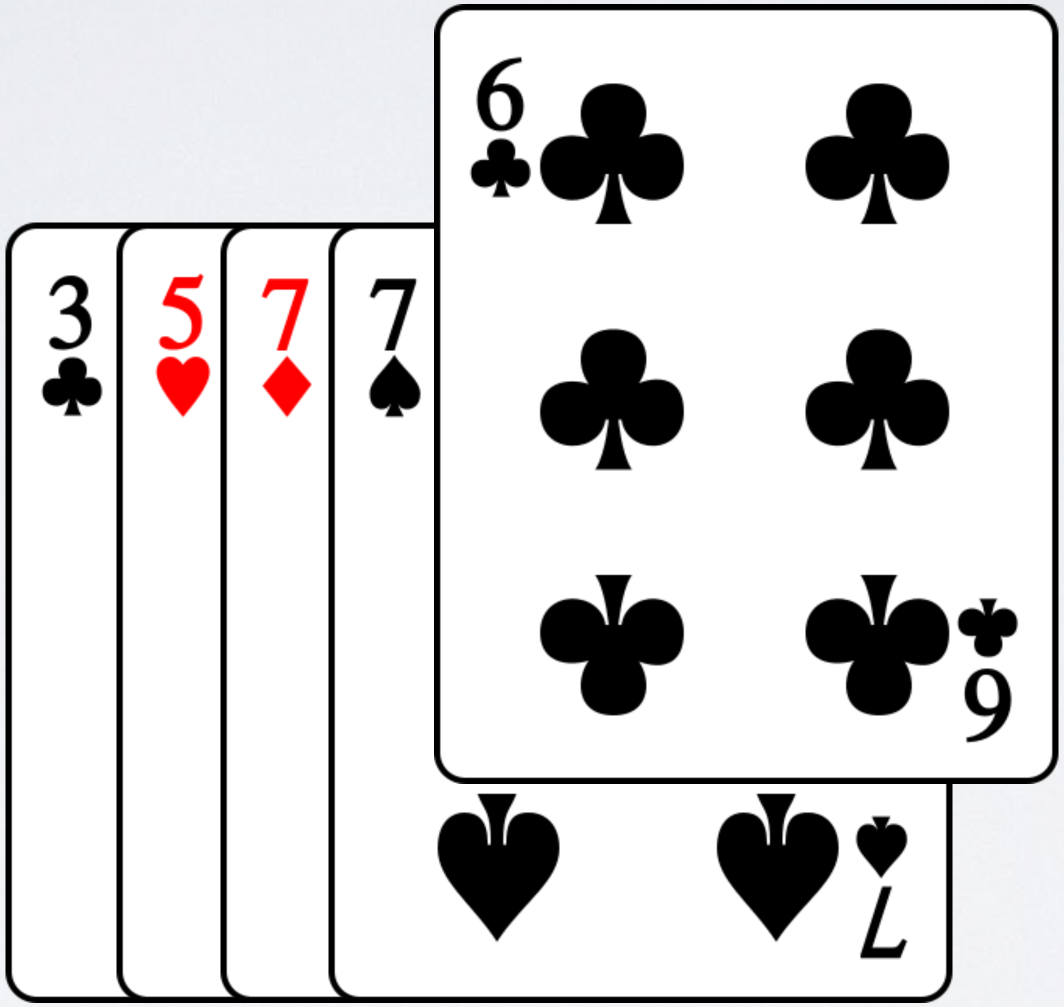
TRI PAR INSERTION



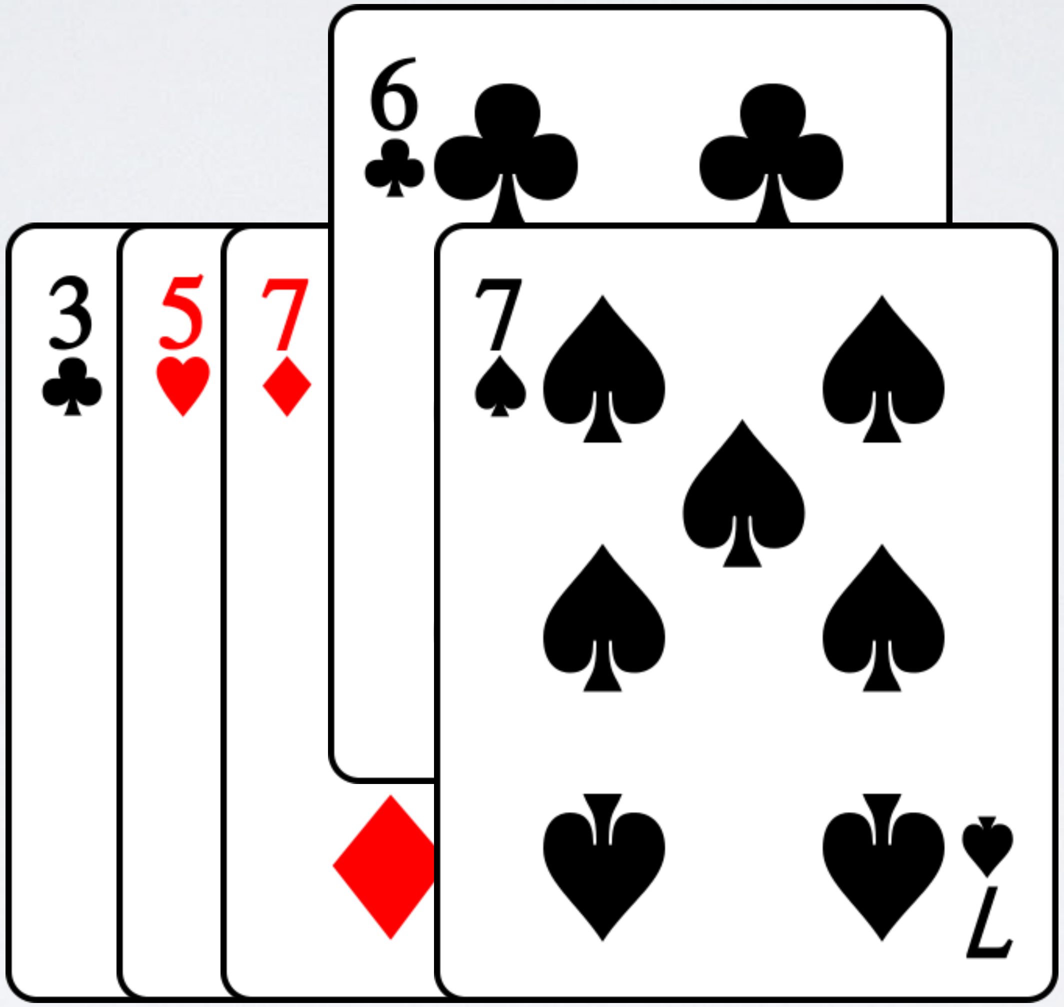
TRI PAR INSERTION



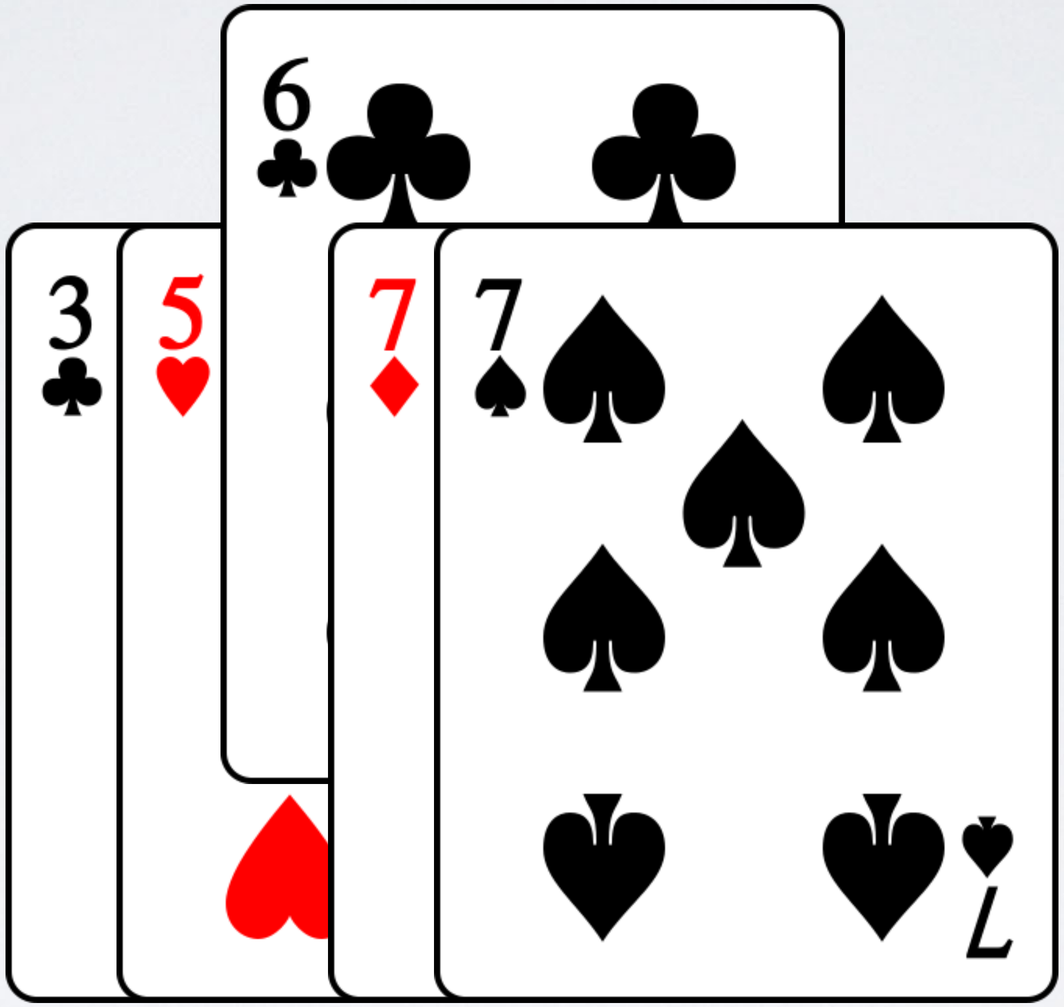
TRI PAR INSERTION



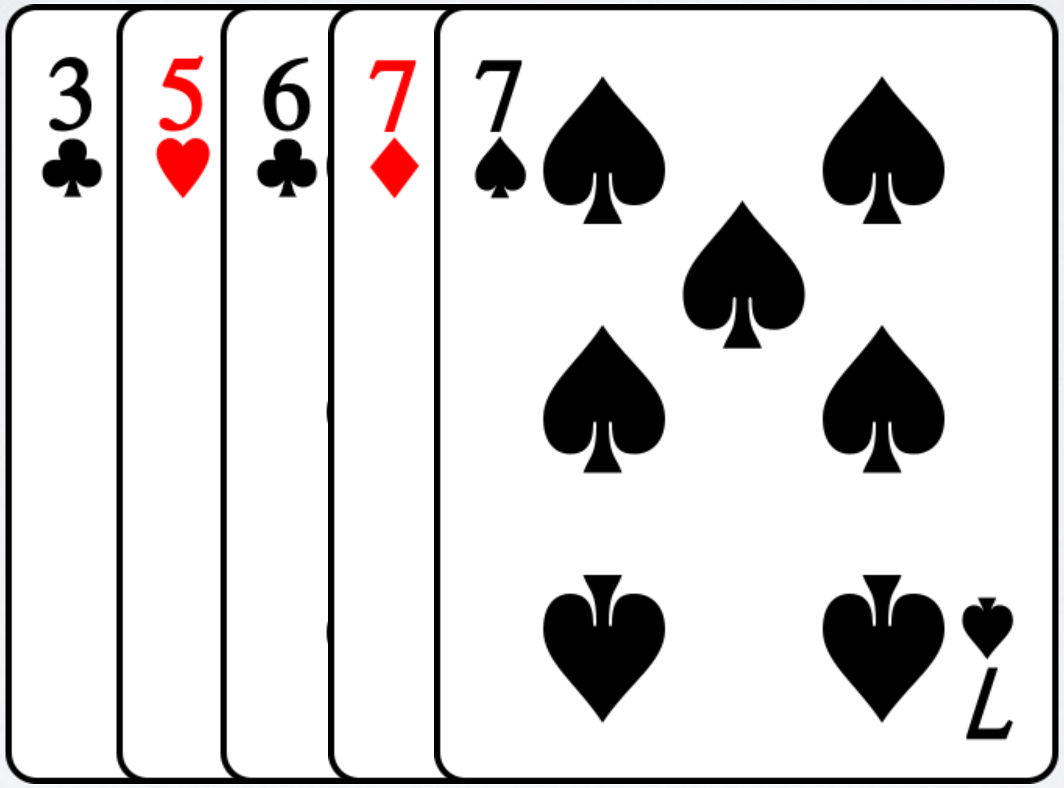
TRI PAR INSERTION



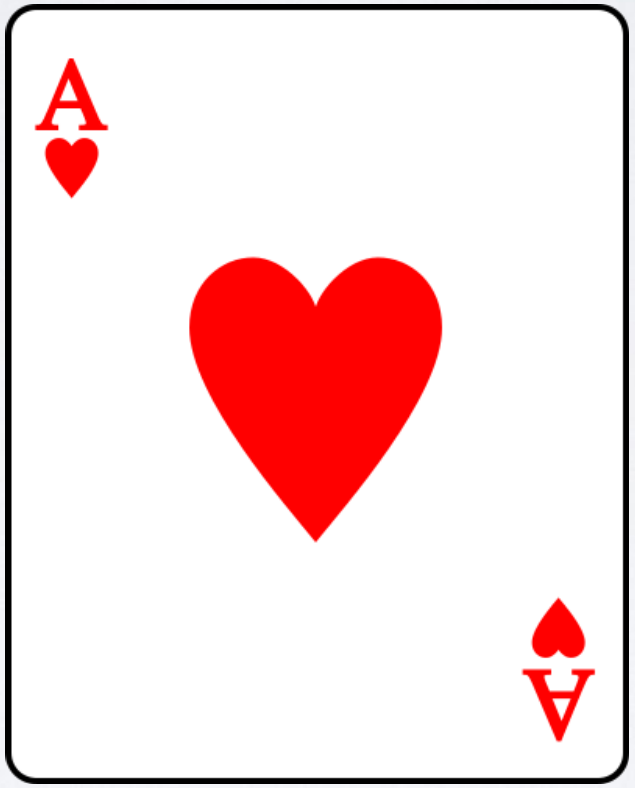
TRI PAR INSERTION



TRI PAR INSERTION

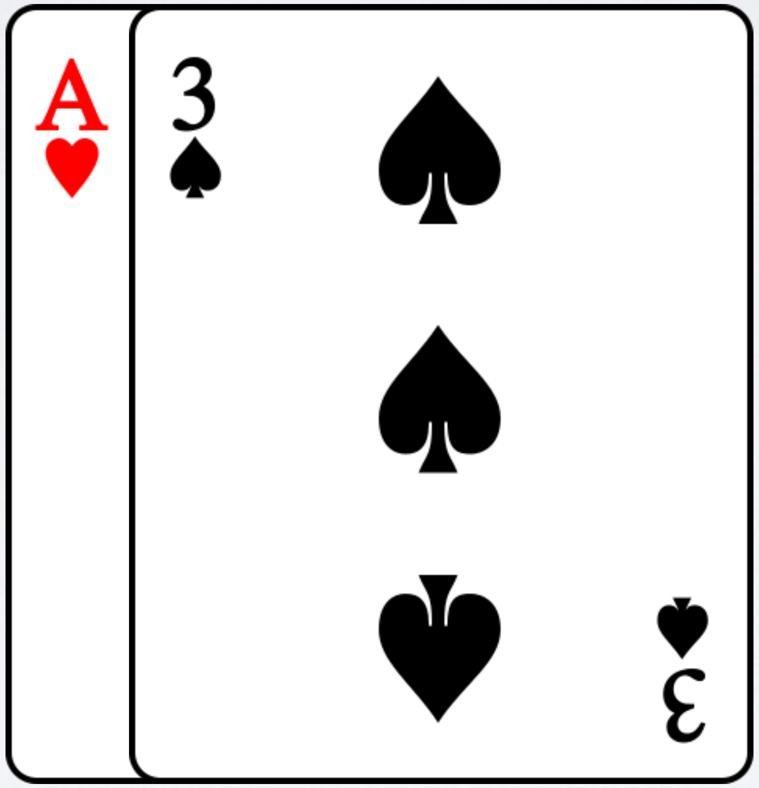


LE MEILLEUR DES CAS



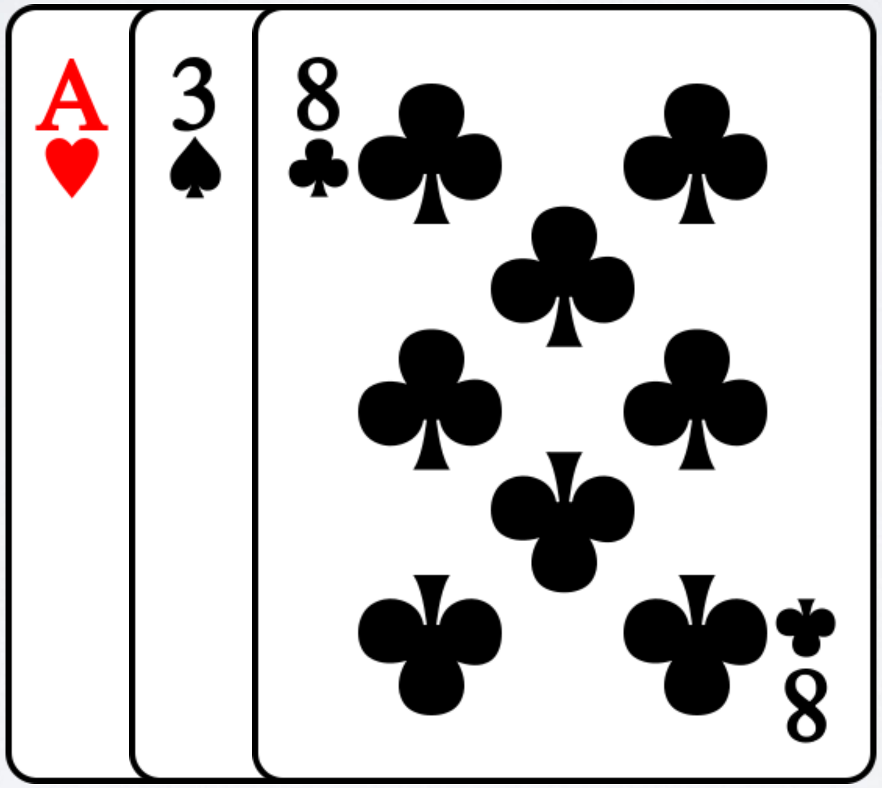
№ operations = 1

LE MEILLEUR DES CAS



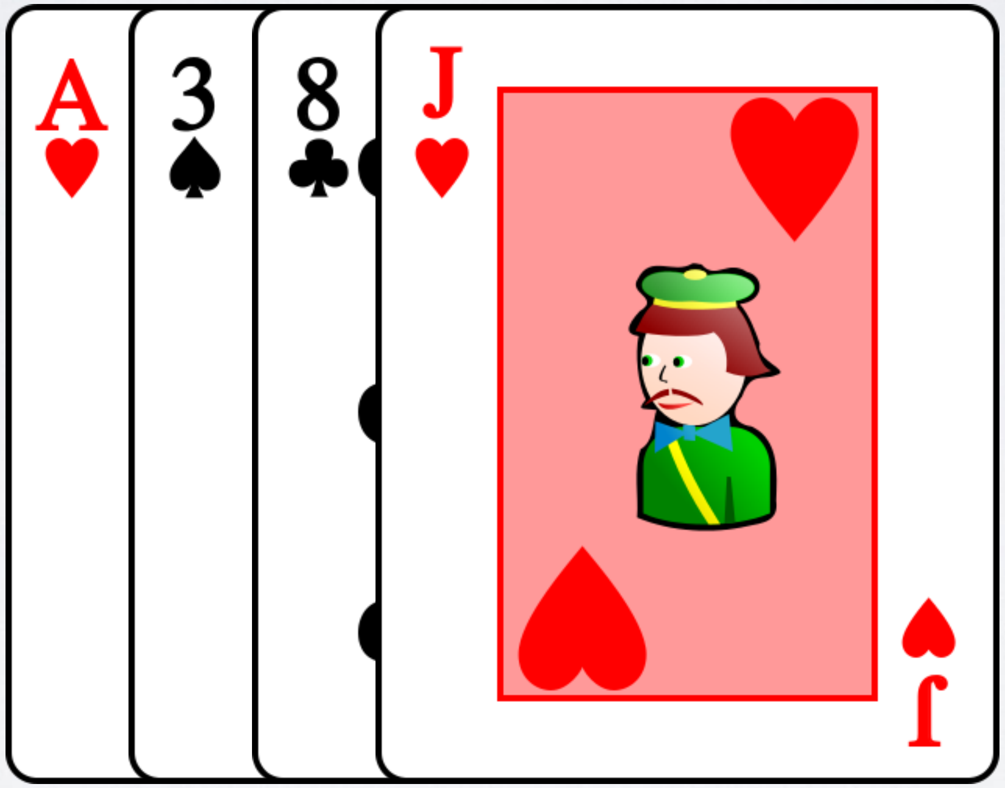
№ operations = 2

LE MEILLEUR DES CAS



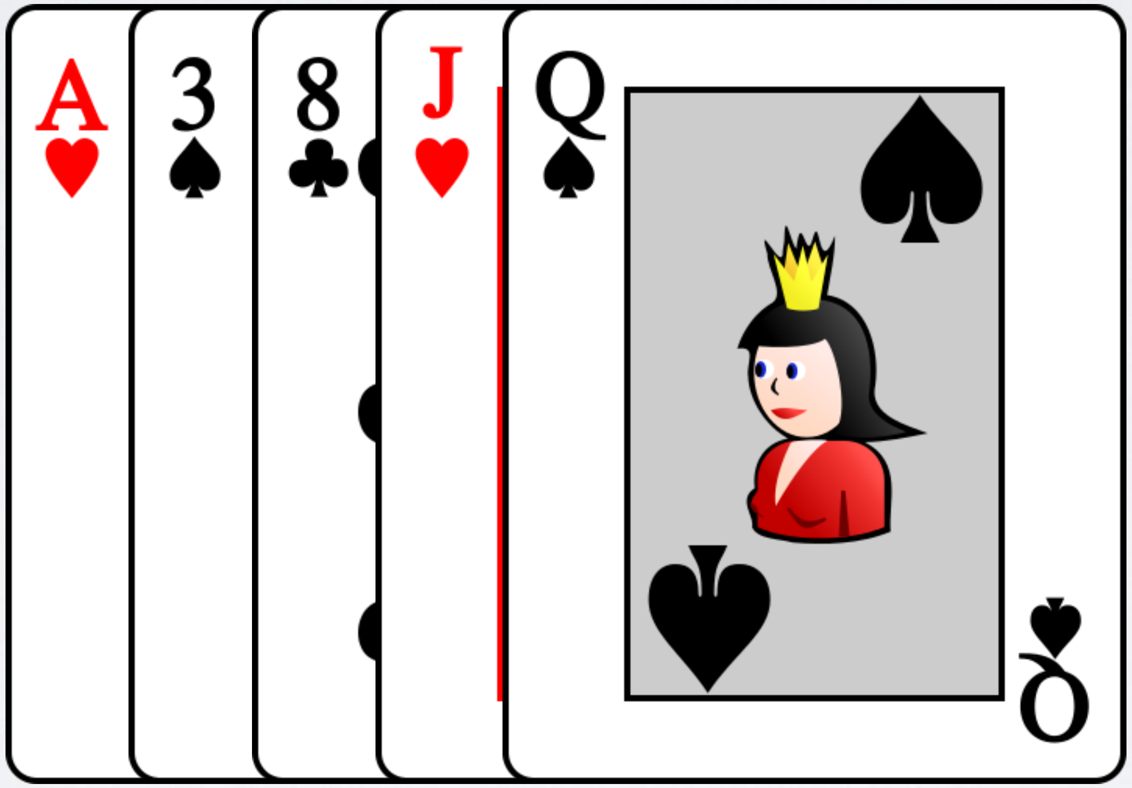
№ operations = 3

LE MEILLEUR DES CAS



№ operations = 4

LE MEILLEUR DES CAS

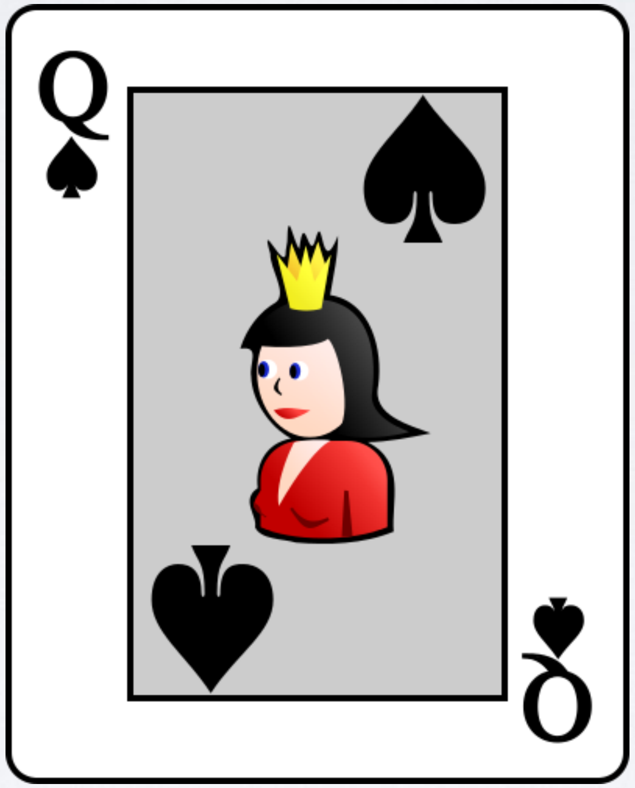


№ operations = 5

LE MEILLEUR DES CAS

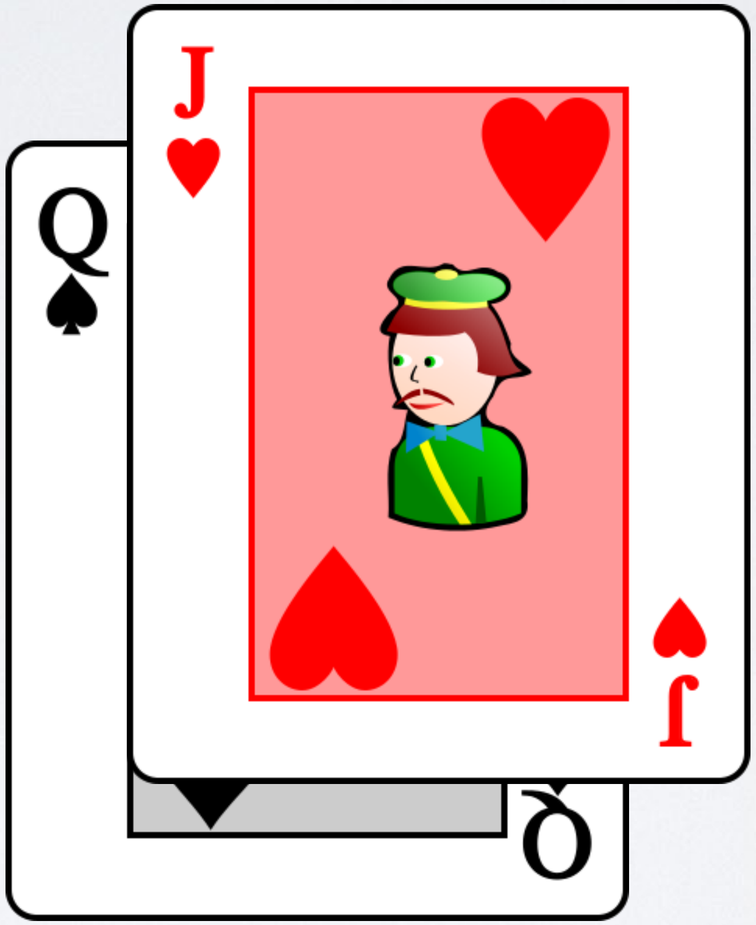
- Les cartes arrivent déjà triées
- On fait n opérations (déplacements de cartes)

LE PIRE DES CAS



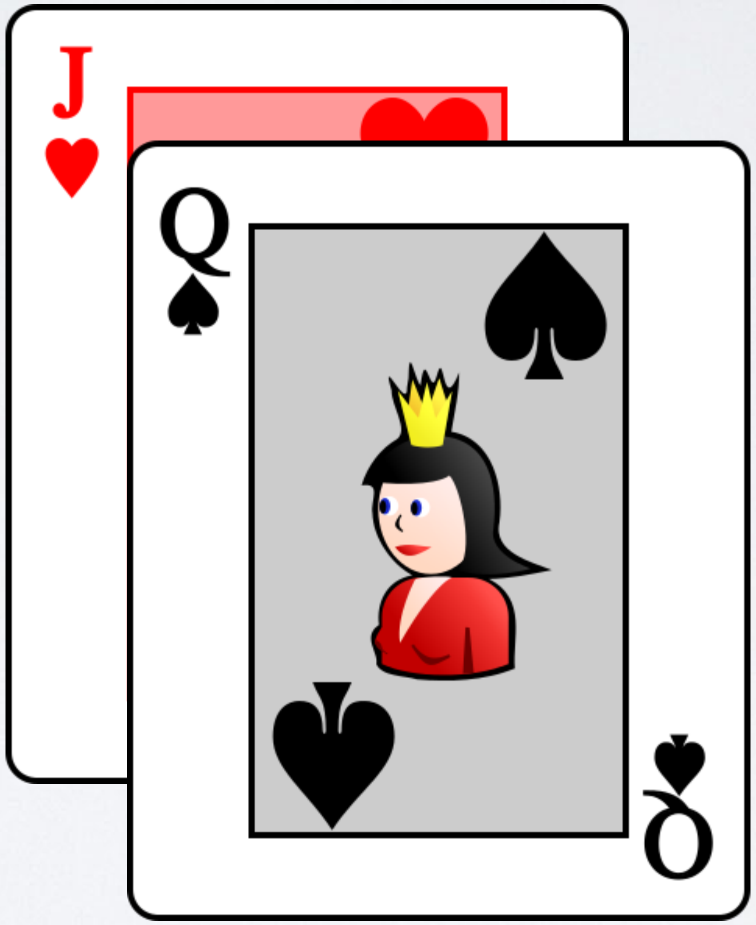
№ operations = 1

LE PIRE DES CAS



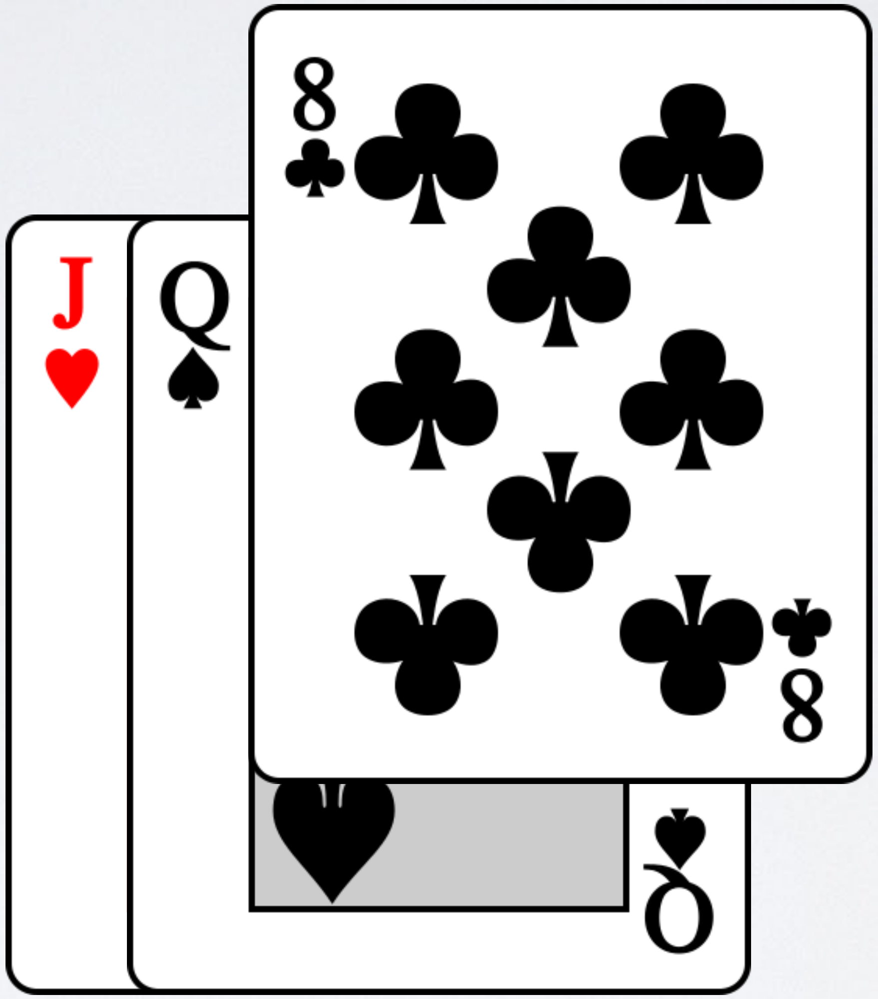
№ operations = 1 + 1

LE PIRE DES CAS



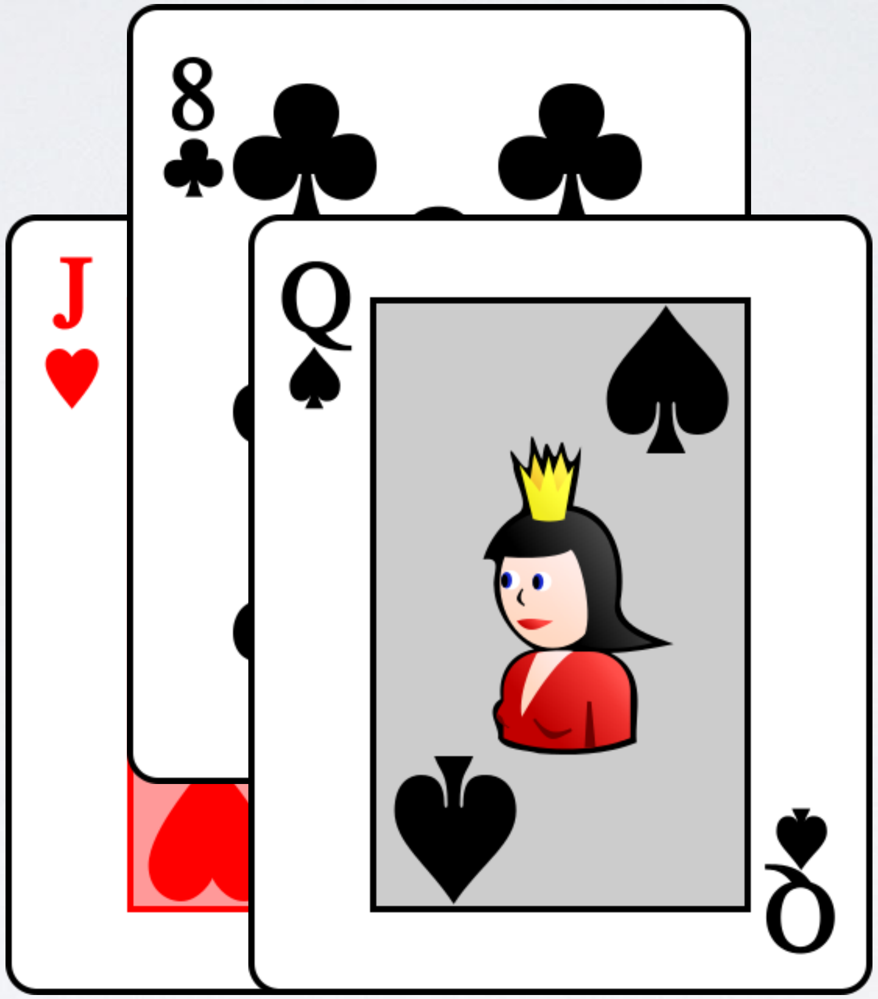
№ operations = 1 + 2

LE PIRE DES CAS



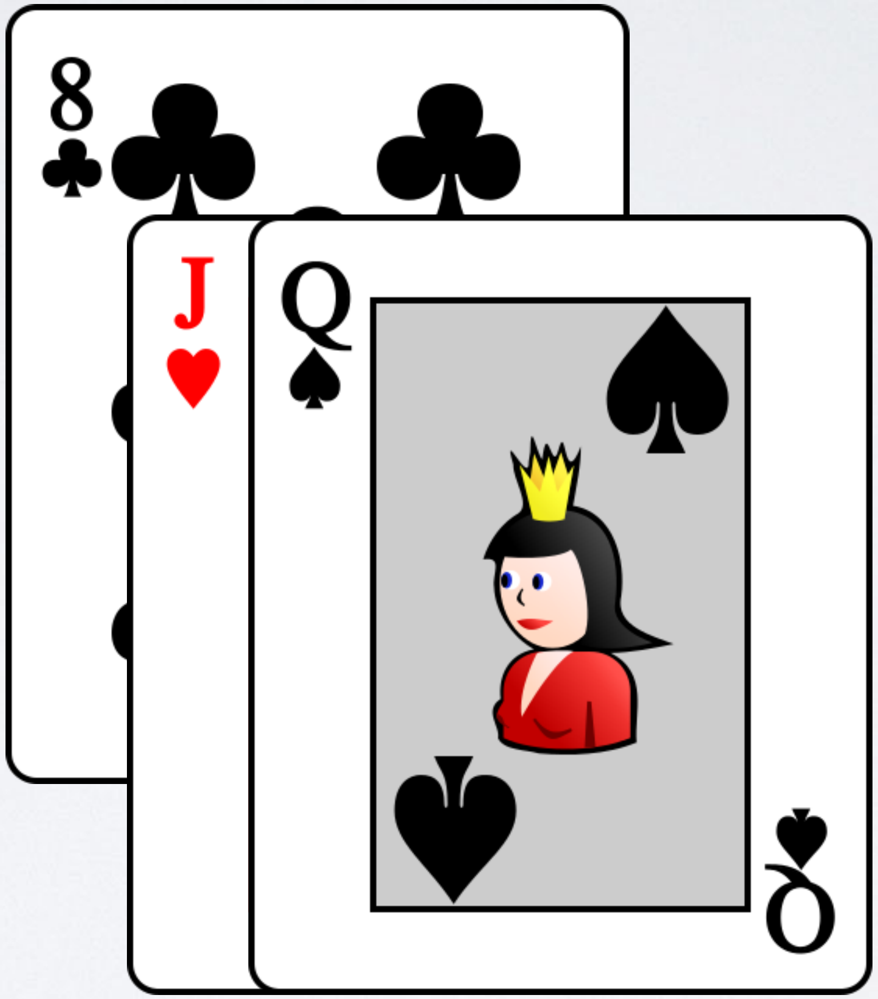
№ operations = 1 + 2 + 1

LE PIRE DES CAS



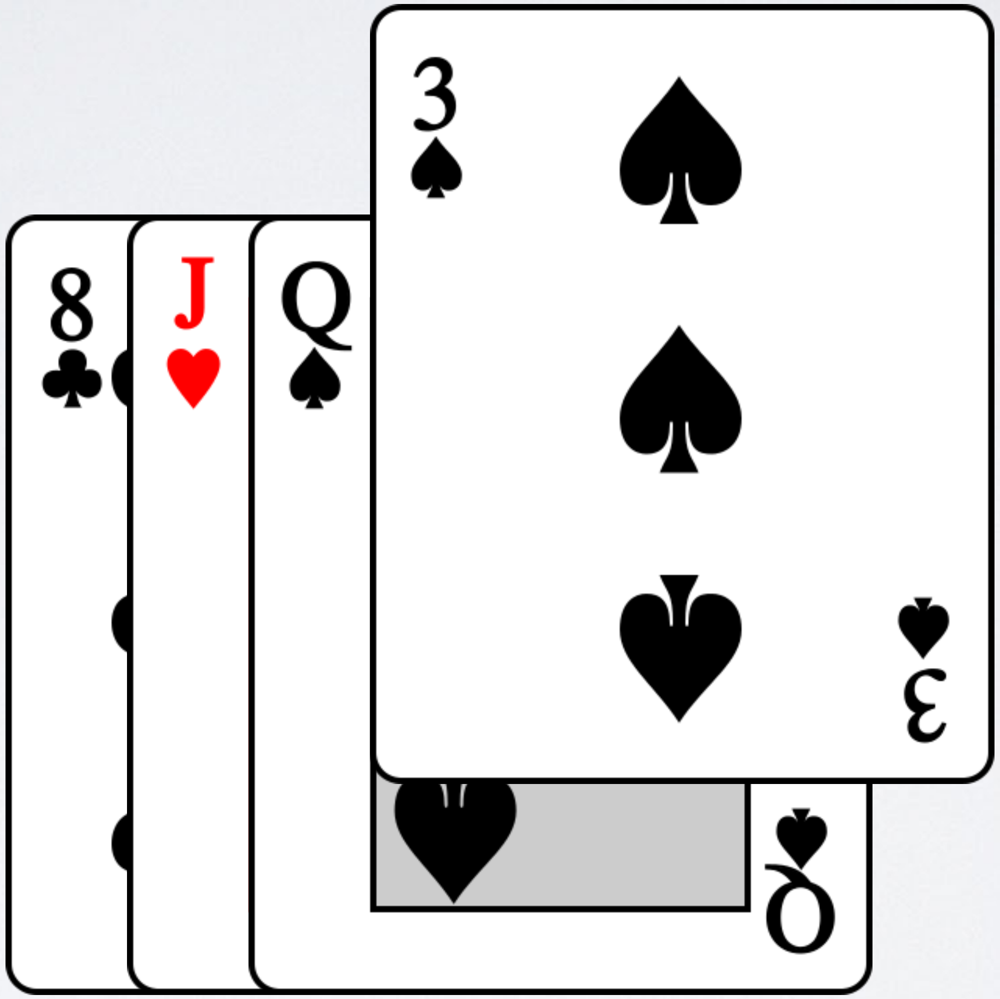
№ operations = 1 + 2 + 2

LE PIRE DES CAS



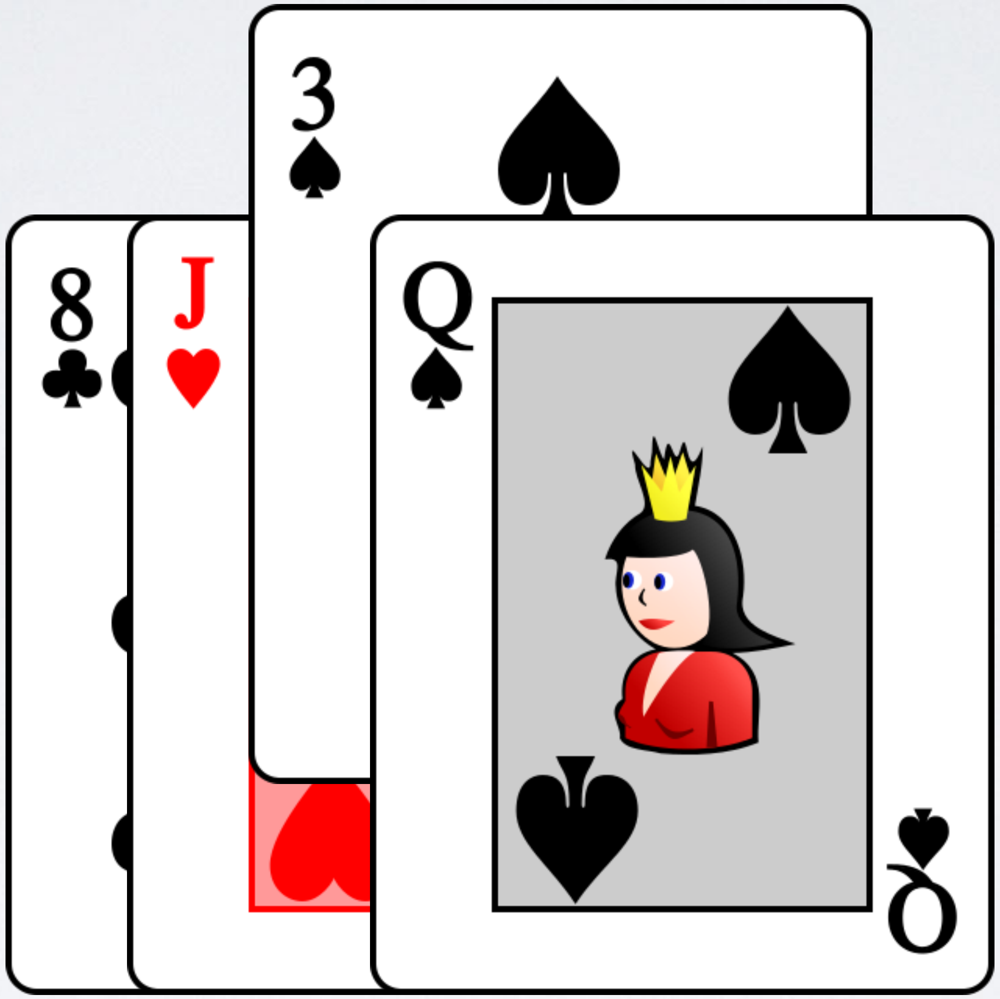
№ operations = 1 + 2 + 3

LE PIRE DES CAS



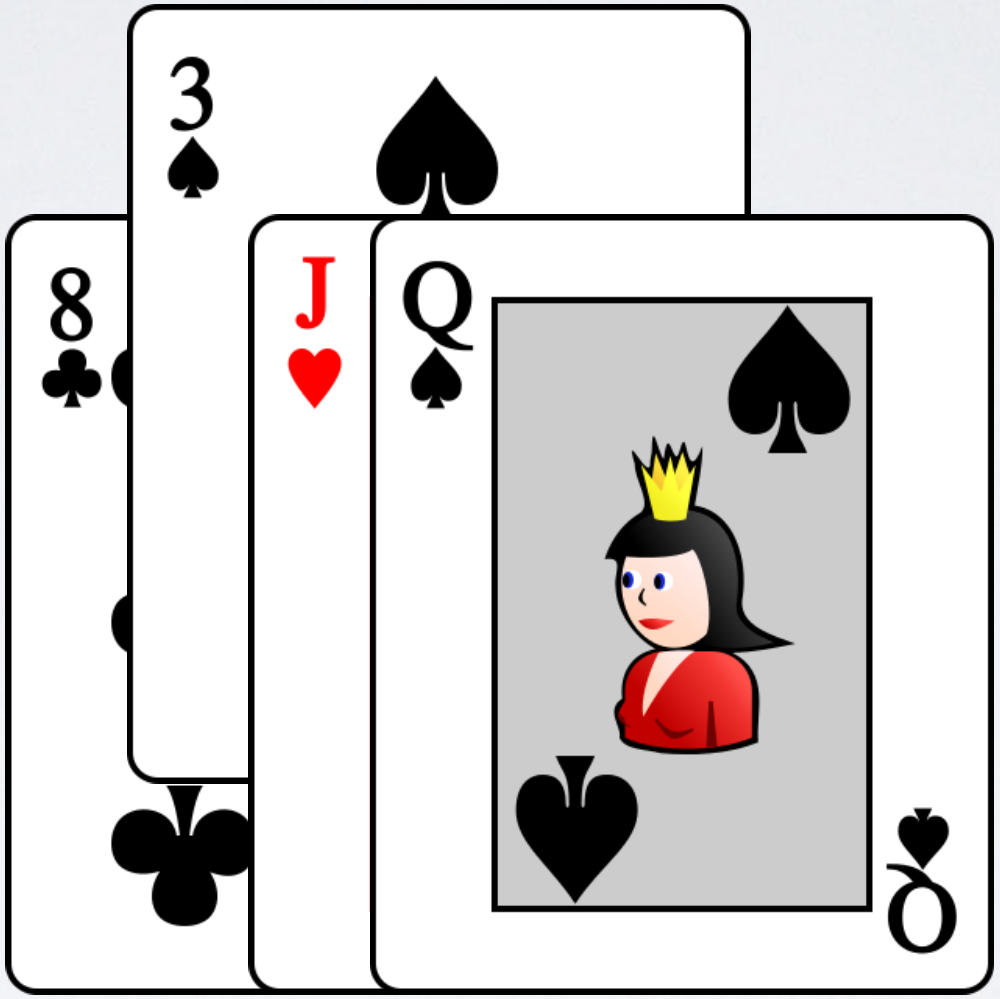
№ operations = 1 + 2 + 3 + 1

LE PIRE DES CAS



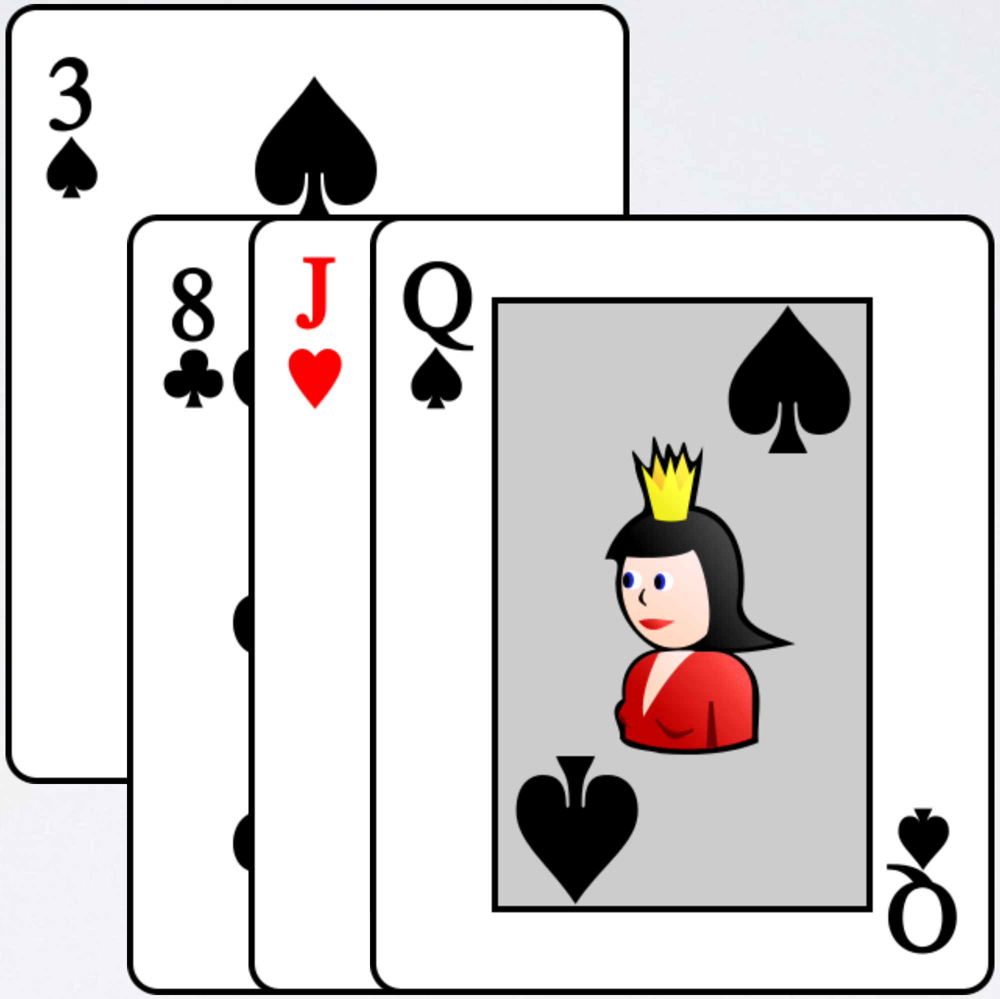
№ operations = 1 + 2 + 3 + 2

LE PIRE DES CAS



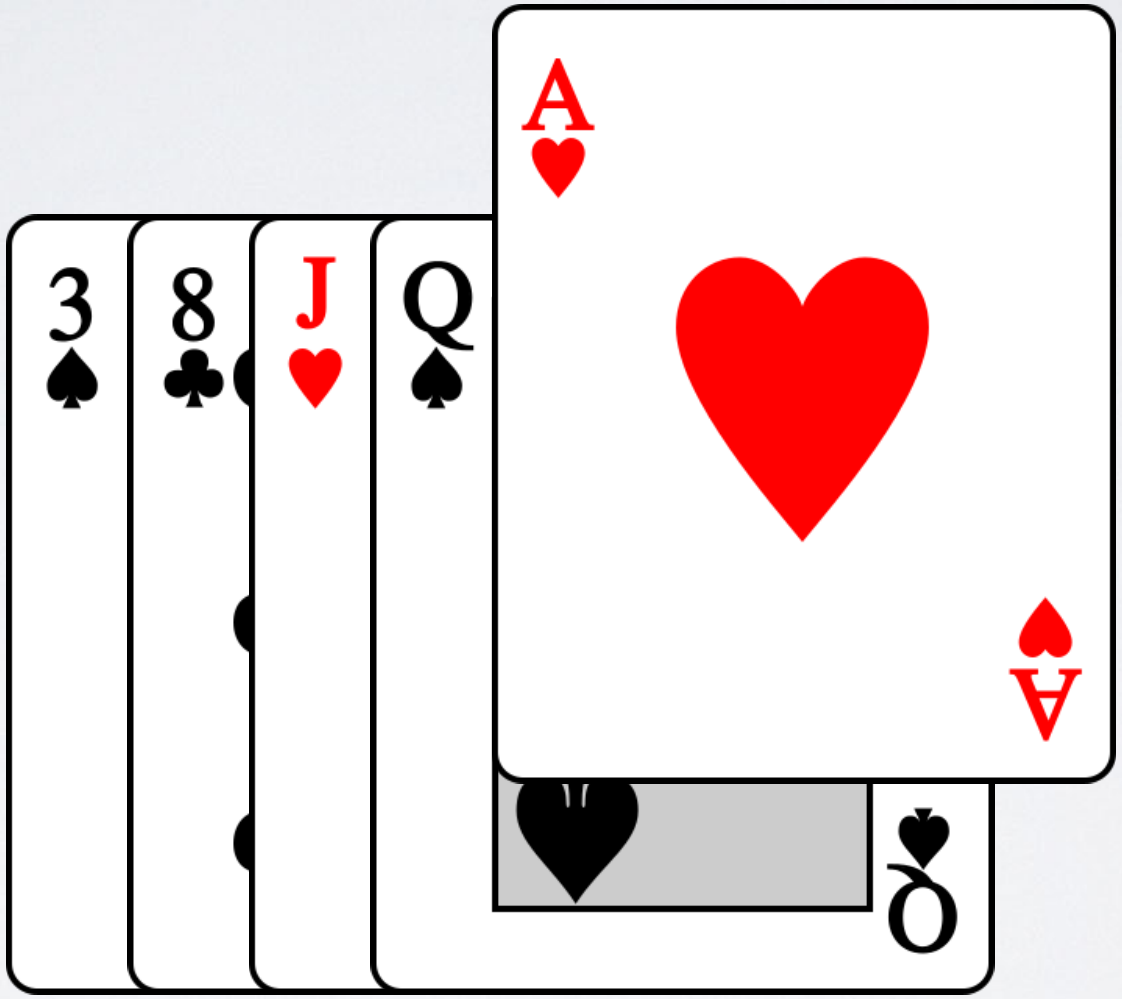
№ operations = 1 + 2 + 3 + 3

LE PIRE DES CAS



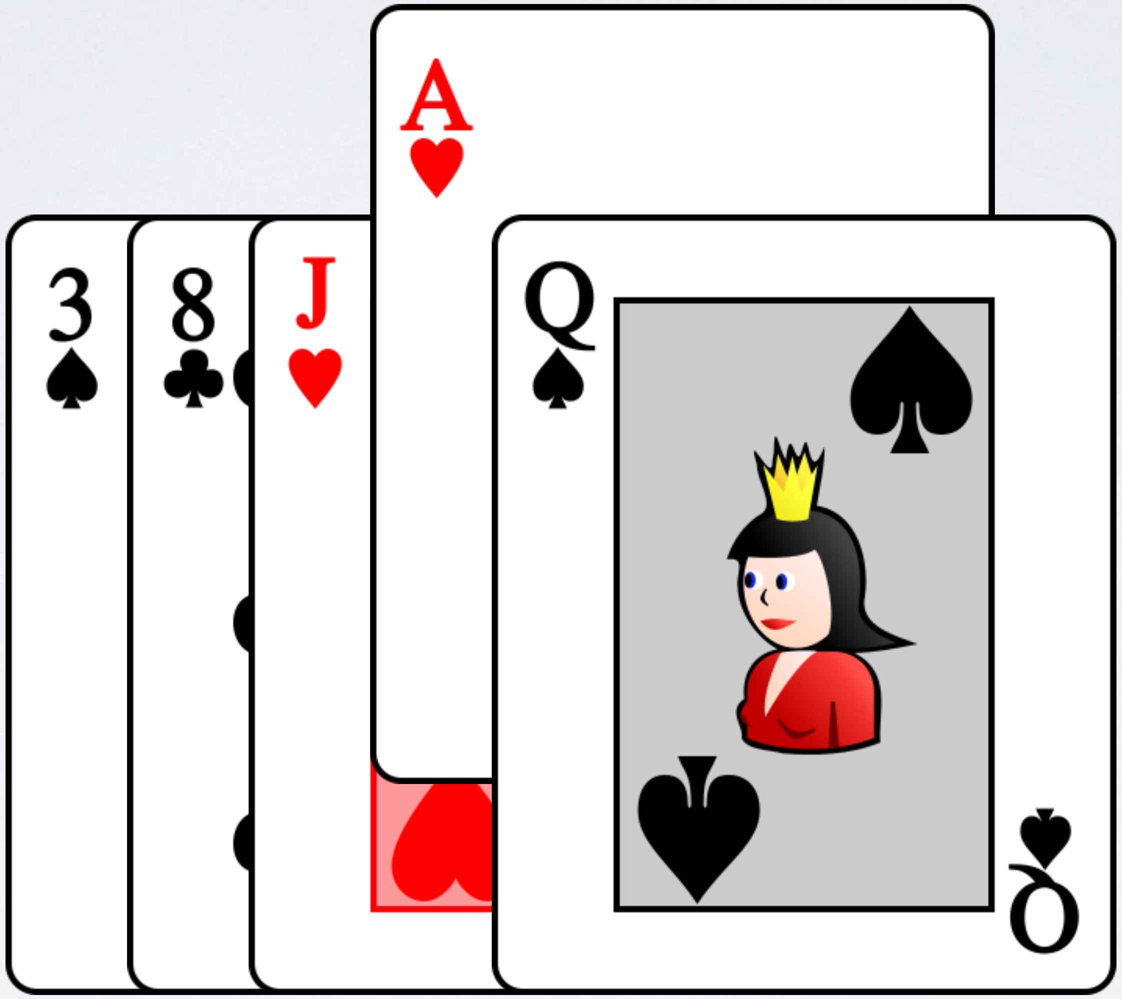
№ operations = 1 + 2 + 3 + 4

LE PIRE DES CAS



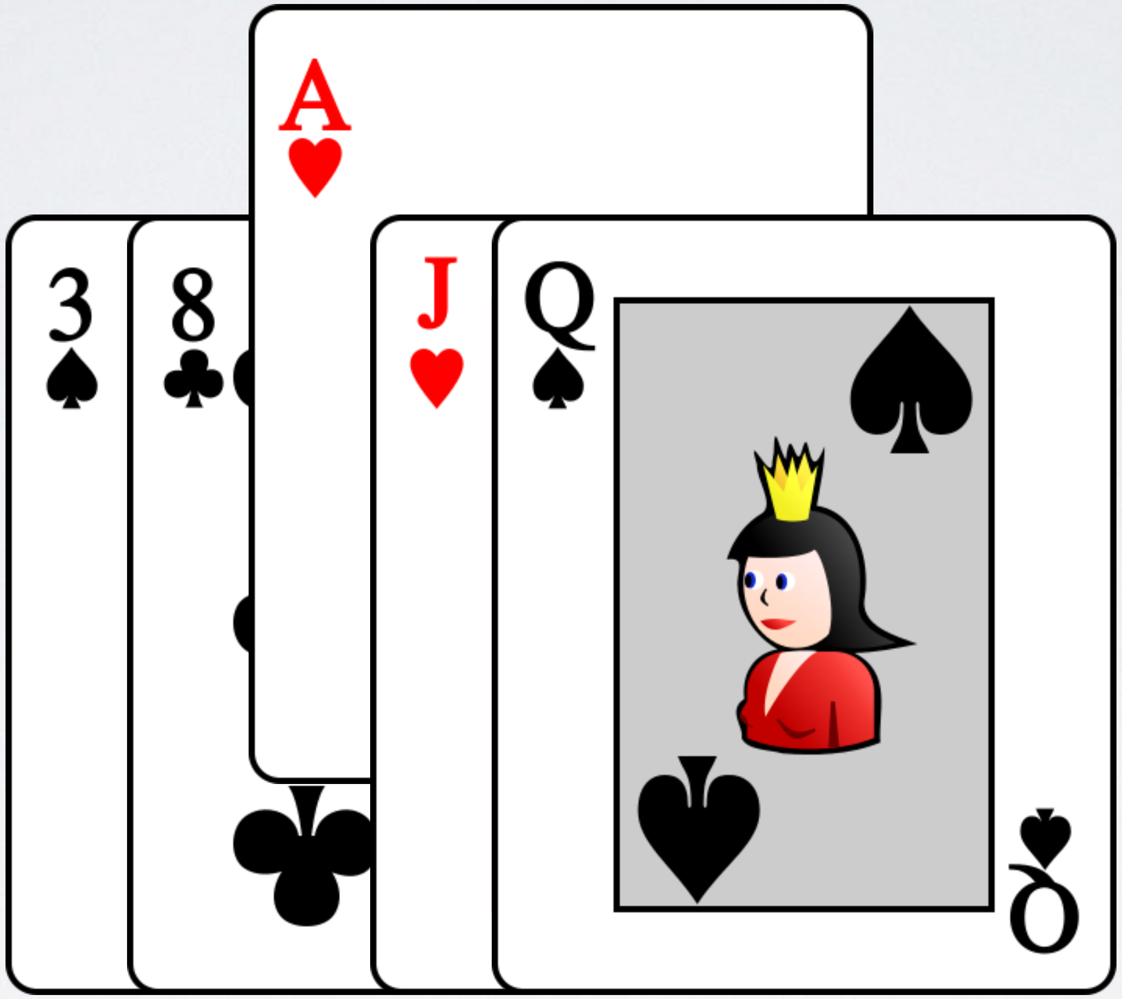
№ operations = 1 + 2 + 3 + 4 + 1

LE PIRE DES CAS



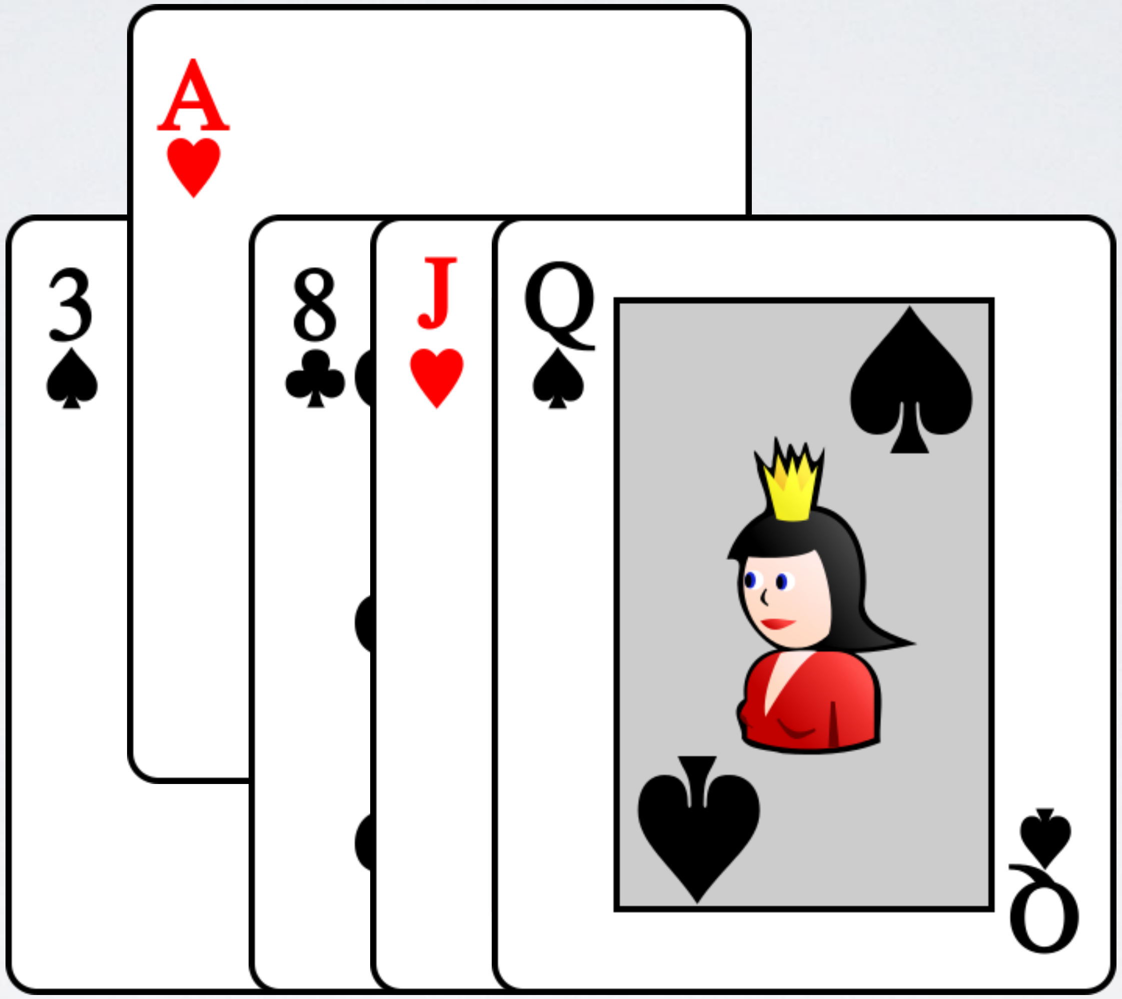
№ operations = 1 + 2 + 3 + 4 + 2

LE PIRE DES CAS



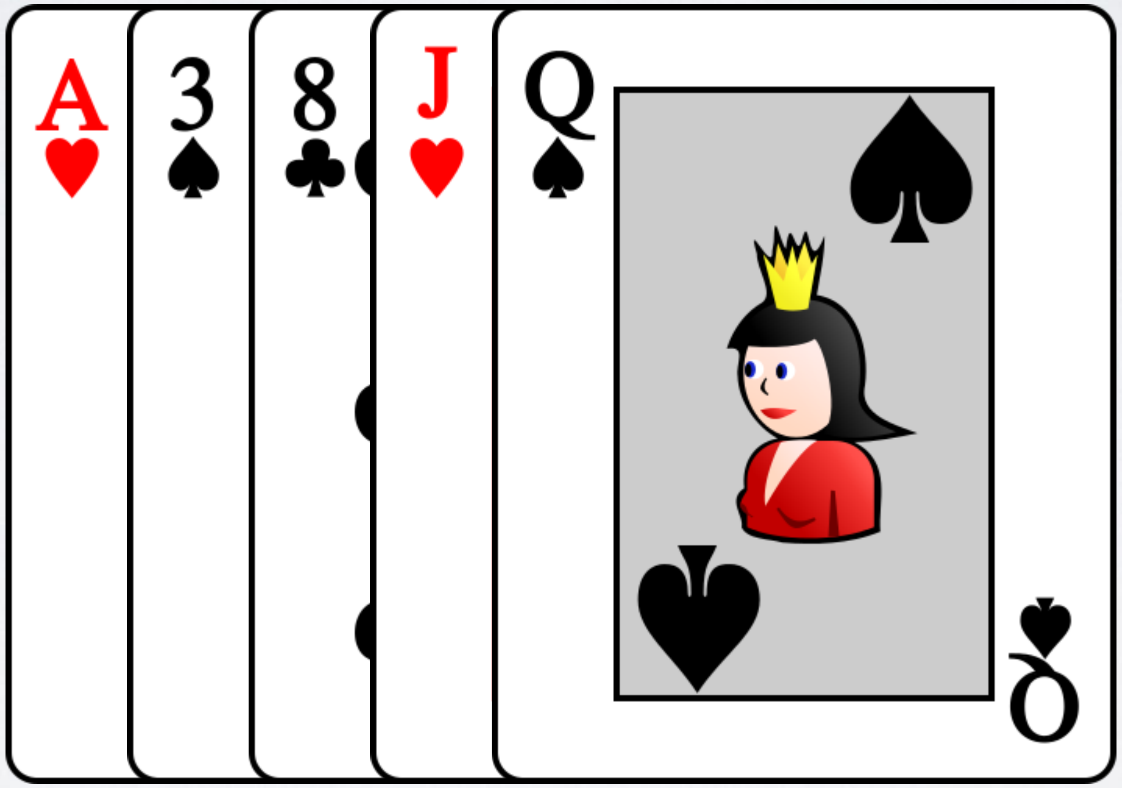
№ operations = 1 + 2 + 3 + 4 + 3

LE PIRE DES CAS



№ operations = 1 + 2 + 3 + 4 + 4

LE PIRE DES CAS



№ operations = 1 + 2 + 3 + 4 + 5

LE PIRE DES CAS

- Les cartes arrivent en ordre décroissant
- On fait i opérations pour la i-ème carte
- Le nombre totale est 1 + 2 + 3 + ⋯ + n

$$\sum_{i=1}^{n} i = \frac{1}{2}n(n+1) = \frac{1}{2}(n^2+n) \in O(n^2)$$

LA BOUCLE « POUR »

```
pour i ≔ x à y faire
    quelque chose
fin
```

=

```
i ≔ x
tant que i ≤ y faire
    quelque chose
    i ≔ i + 1
fin
```

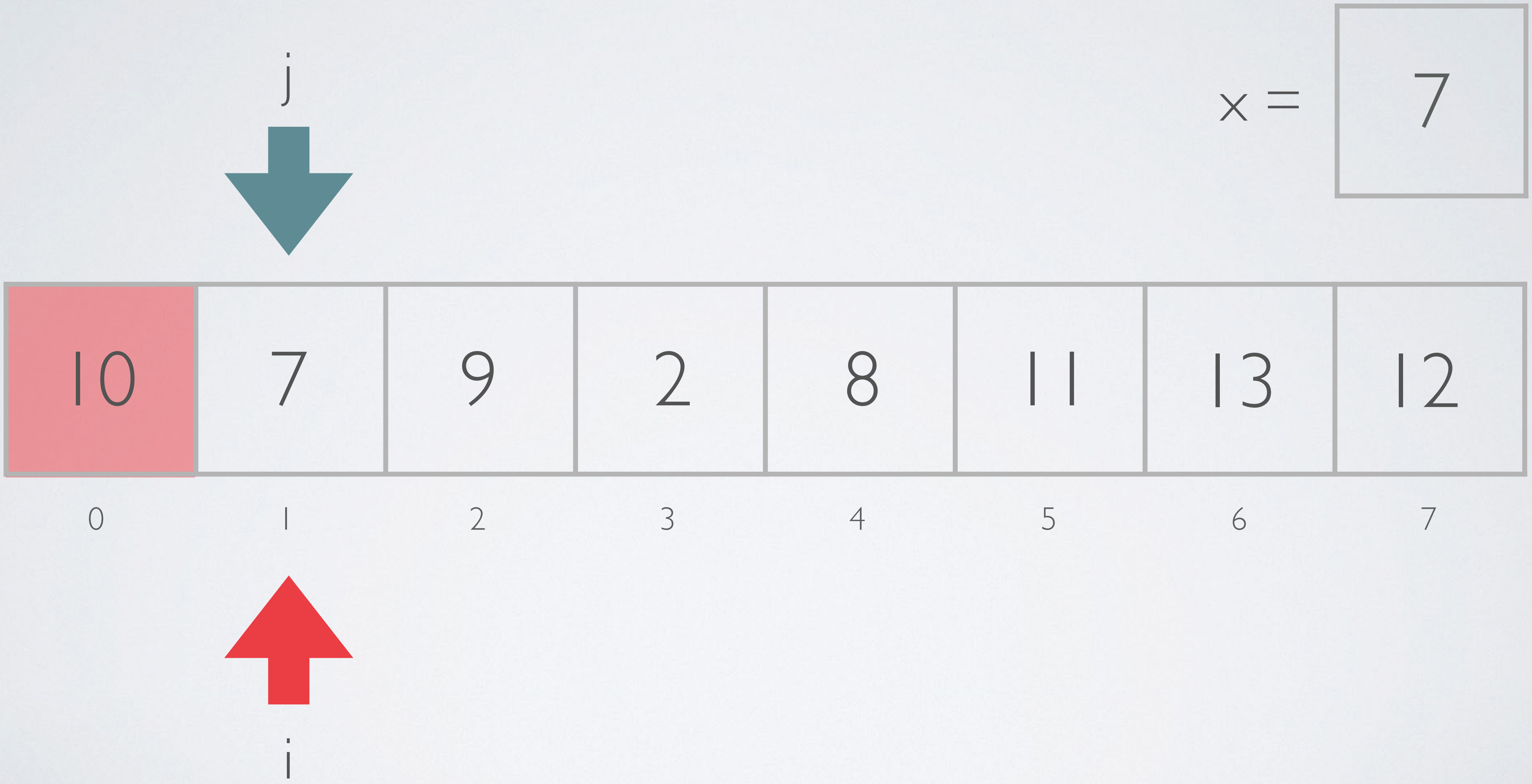
TRI D'UN TABLEAU PAR INSERTION

```
procedure trier-par-insertion(T)
    n ≔ longueur(T)
    pour i ≔ 1 à n – 1 faire
        x ≔ T[i]
        j ≔ i
        tant que j > 0 et x < T[j – 1] faire
            (décaler d'un élément)
            T[j] ≔ T[j – 1]
            j ≔ j – 1
        (ici x ≥ T[j – 1] ou bien j = 0)
        T[j] ≔ x
```

TRI PAR INSERTION

10	7	9	2	8	11	13	12
0	1	2	3	4	5	6	7

TRI PAR INSERTION



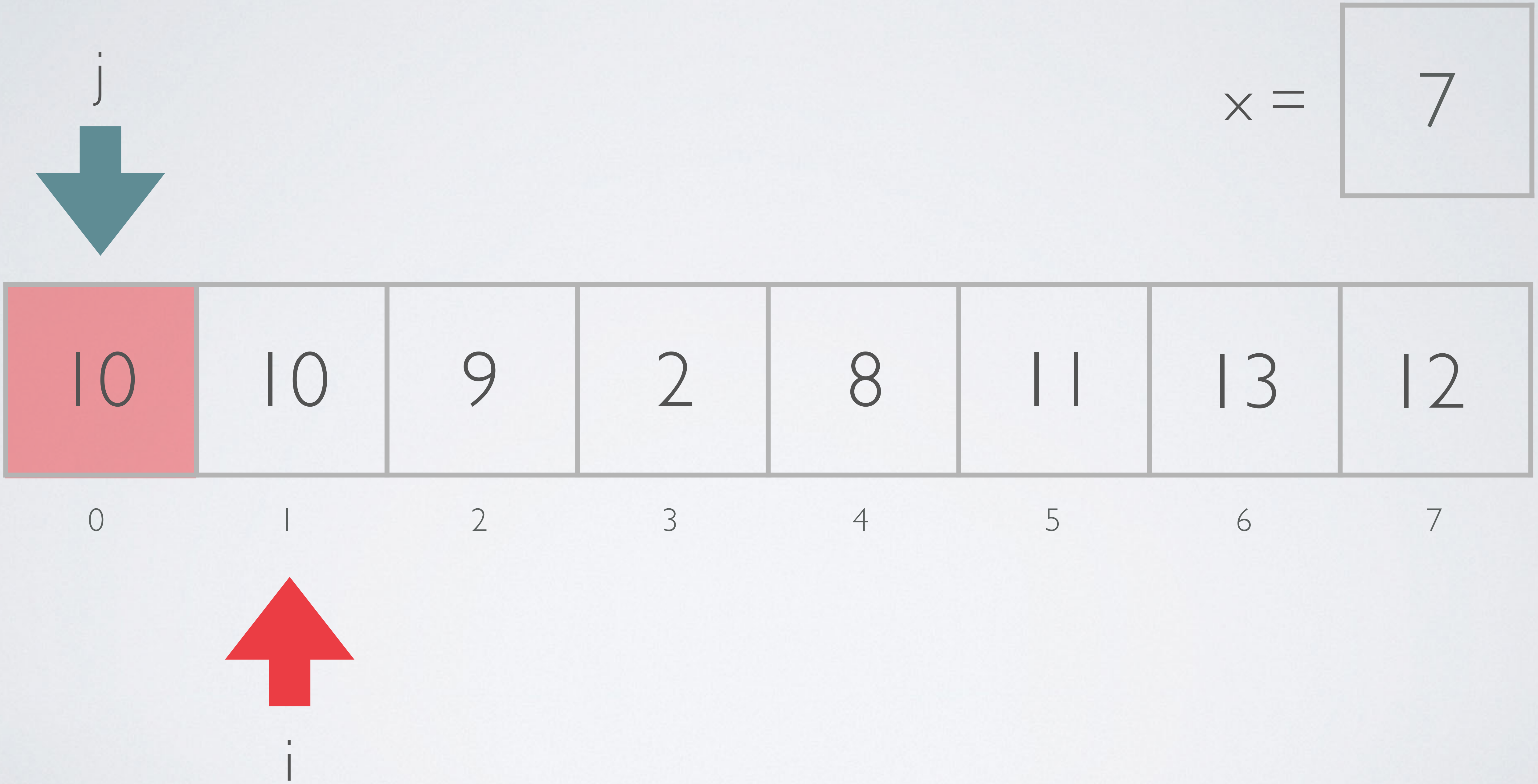
j

x = 7

10	7	9	2	8	11	13	12
0	1	2	3	4	5	6	7

i

TRI PAR INSERTION



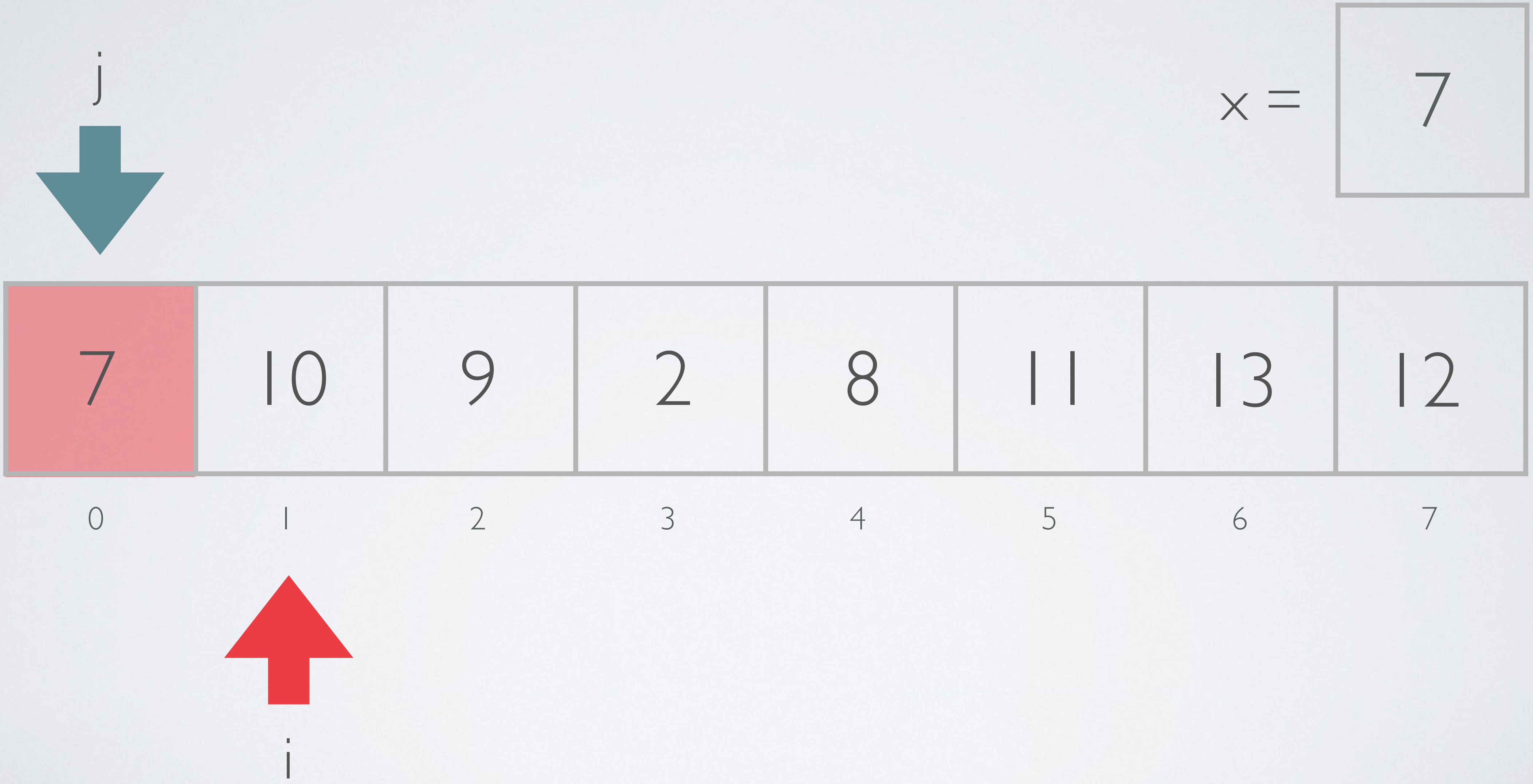
j

x = 7

0	1	2	3	4	5	6	7
10	10	9	2	8	11	13	12

i

TRI PAR INSERTION



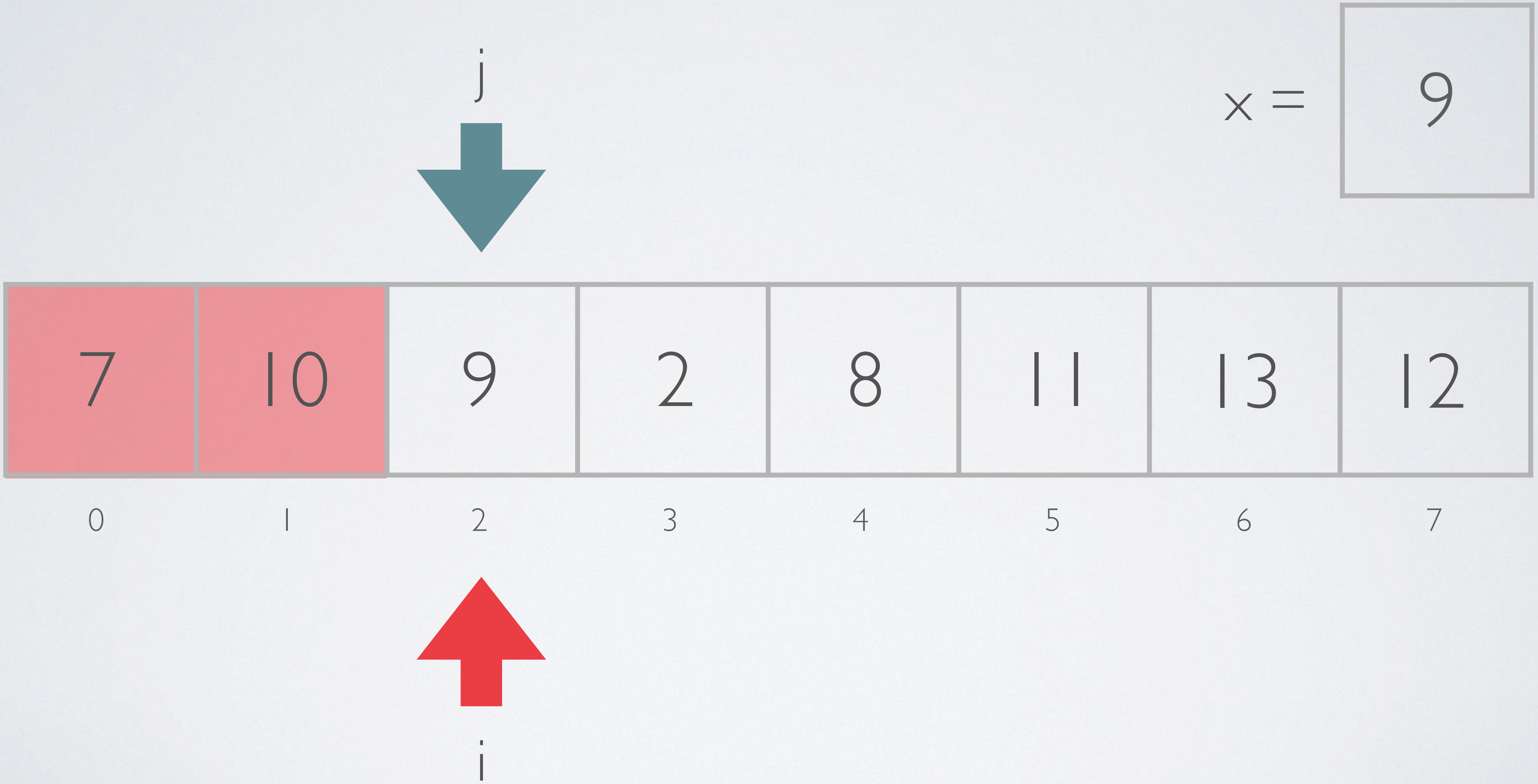
j

x = 7

0	1	2	3	4	5	6	7
7	10	9	2	8	11	13	12

i

TRI PAR INSERTION



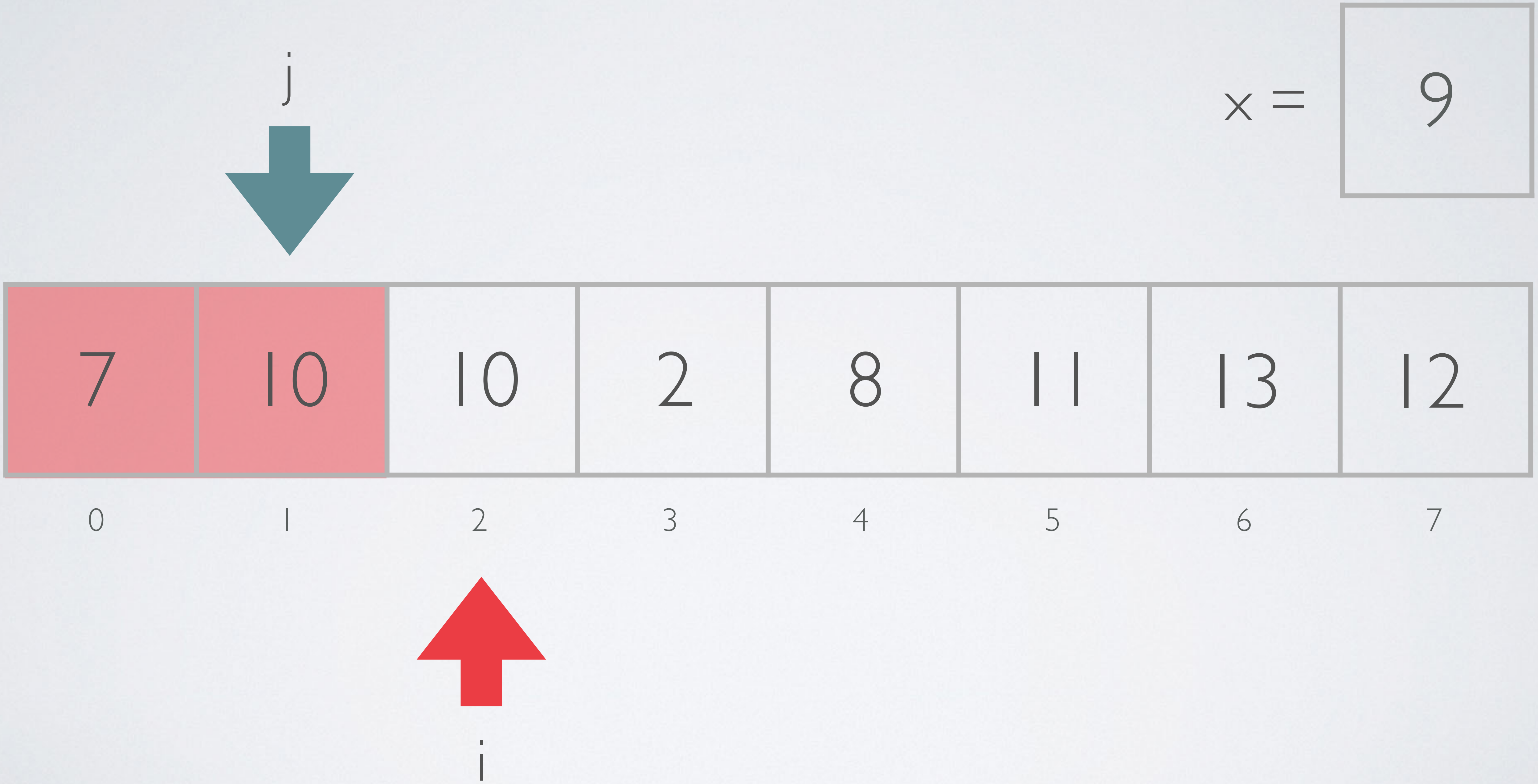
j

x = 9

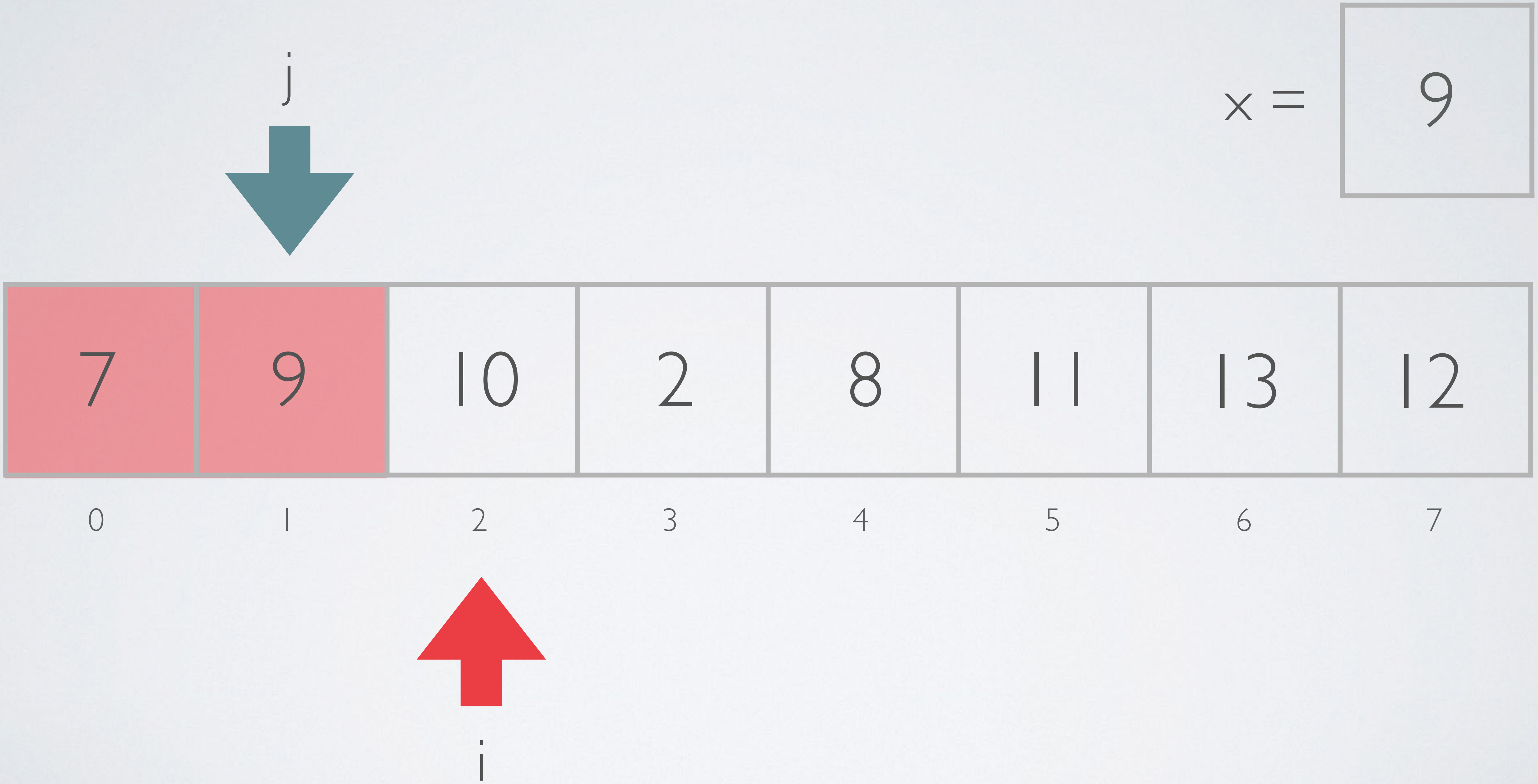
0	1	2	3	4	5	6	7
7	10	9	2	8	11	13	12

i

TRI PAR INSERTION



TRI PAR INSERTION



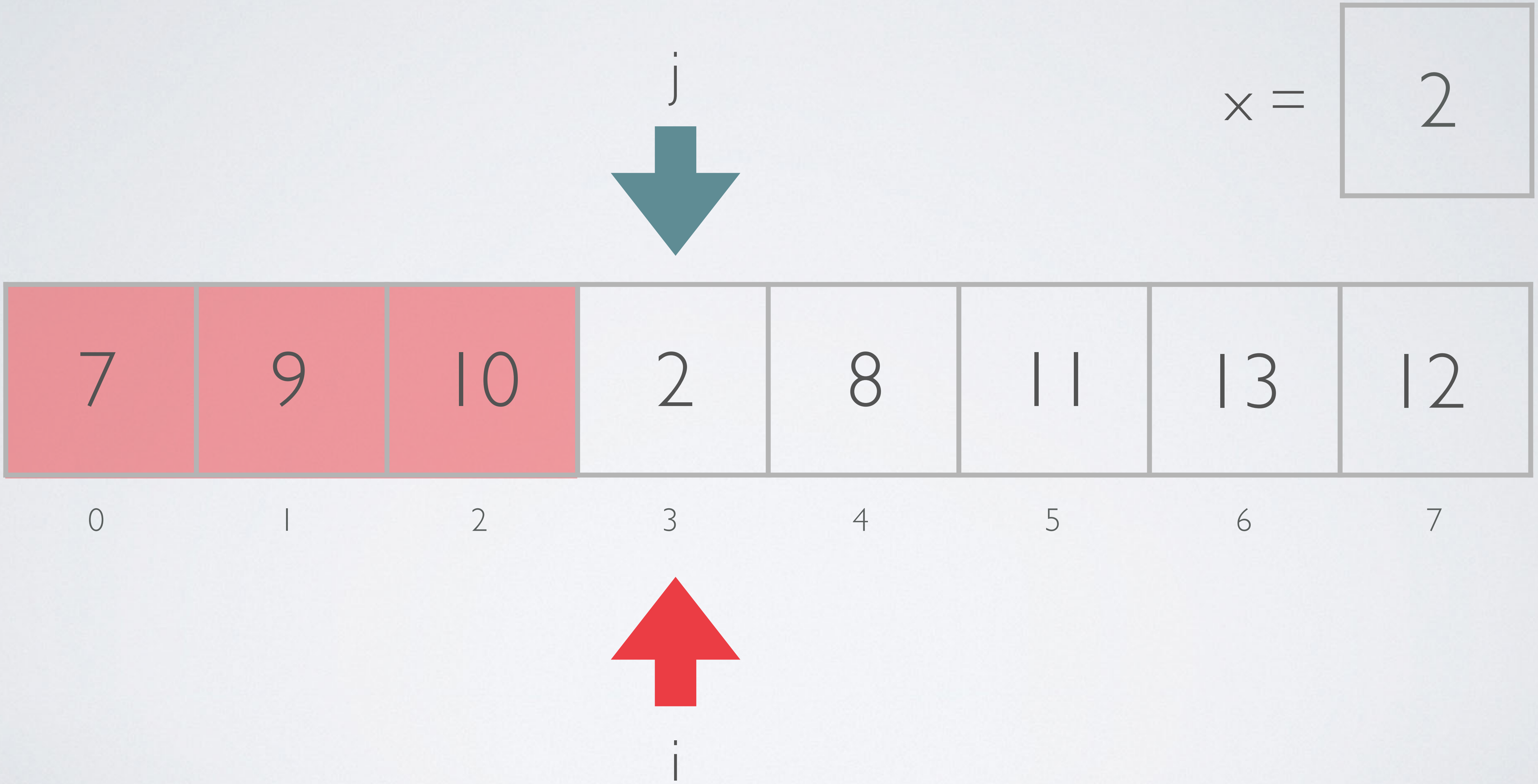
j

x = 9

7	9	10	2	8	11	13	12
0	1	2	3	4	5	6	7

i

TRI PAR INSERTION



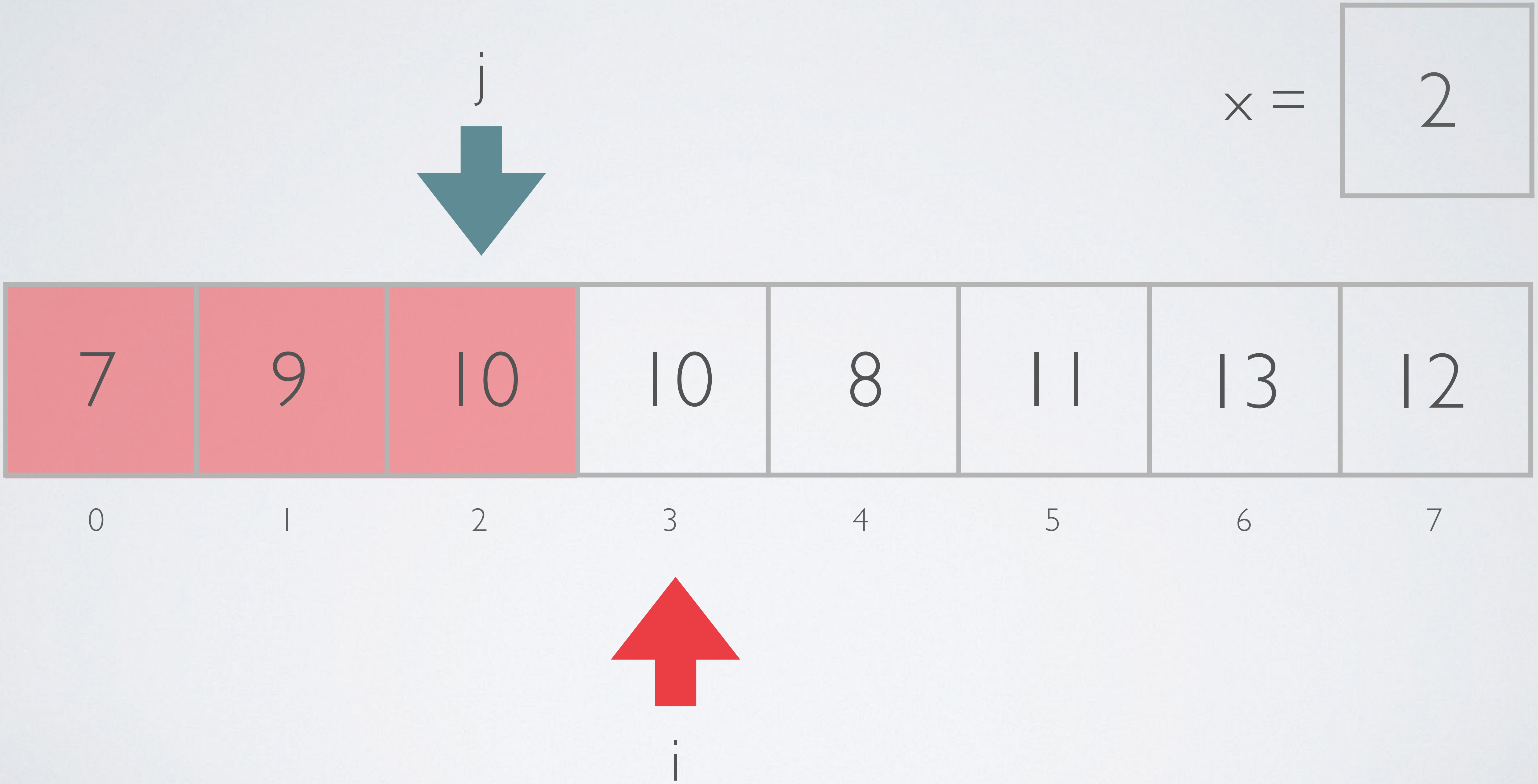
j

x = 2

7	9	10	2	8	11	13	12
0	1	2	3	4	5	6	7

i

TRI PAR INSERTION



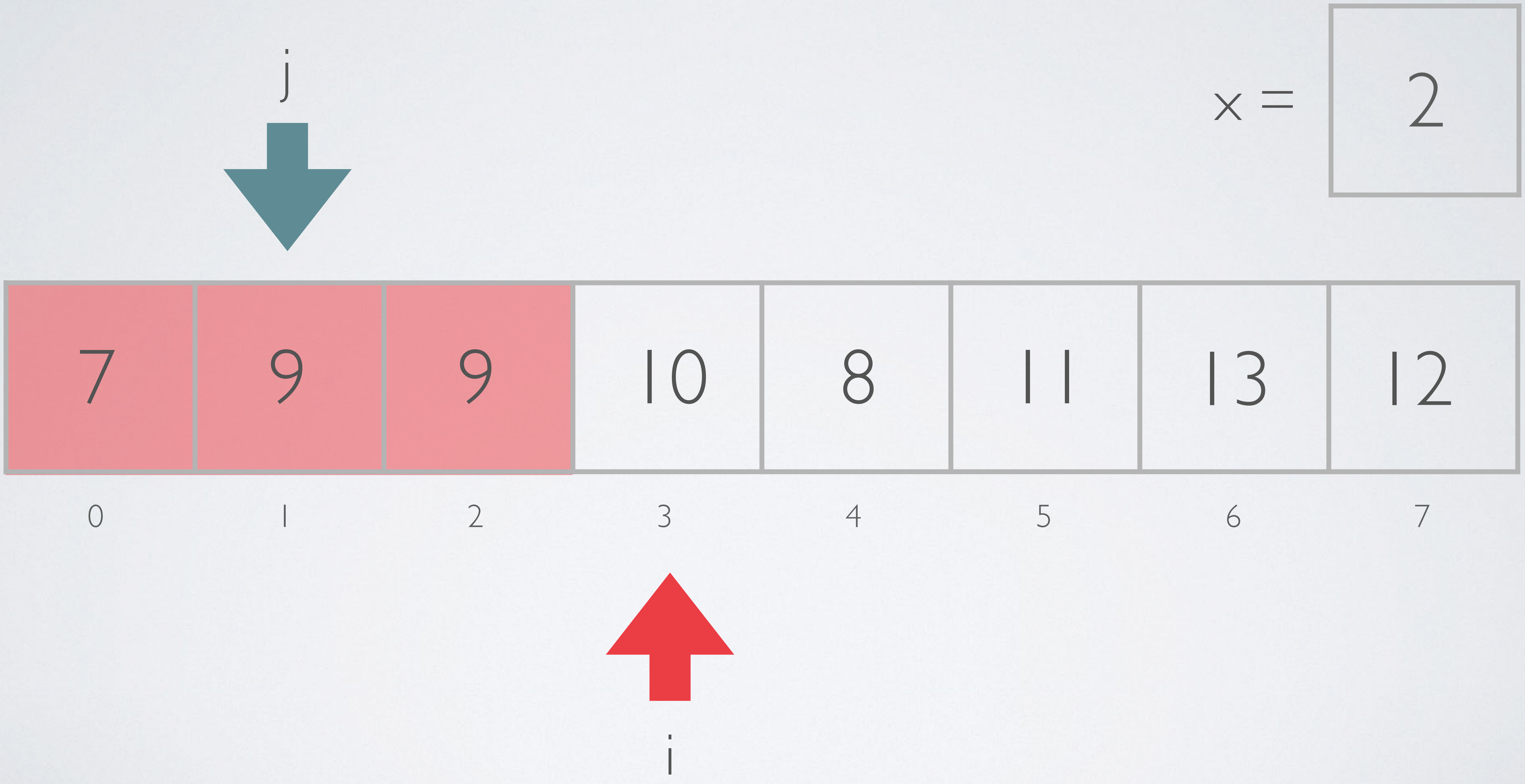
j

x = 2

7	9	10	10	8	11	13	12
0	1	2	3	4	5	6	7

i

TRI PAR INSERTION



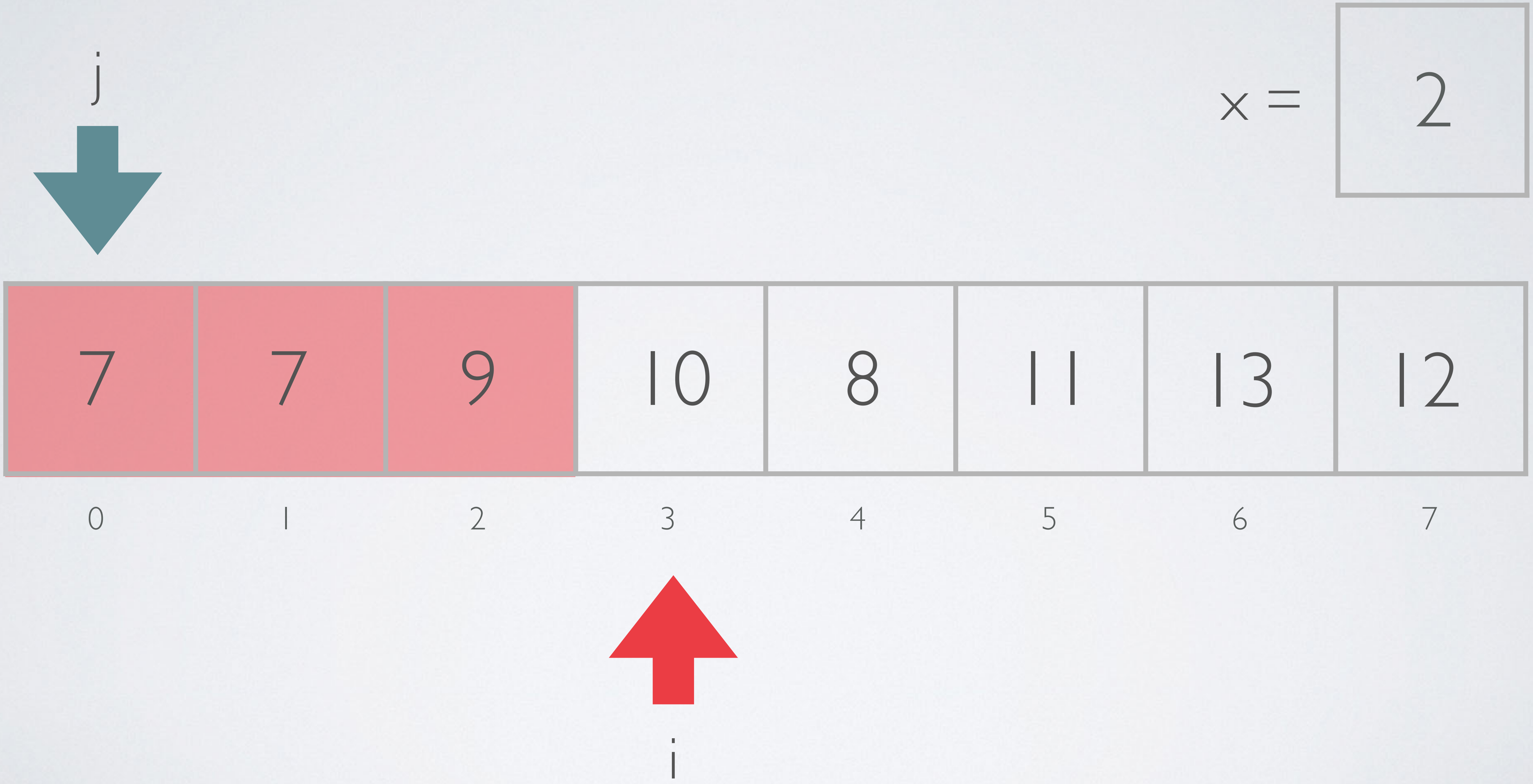
j

x = 2

7	9	9	10	8	11	13	12
0	1	2	3	4	5	6	7

i

TRI PAR INSERTION



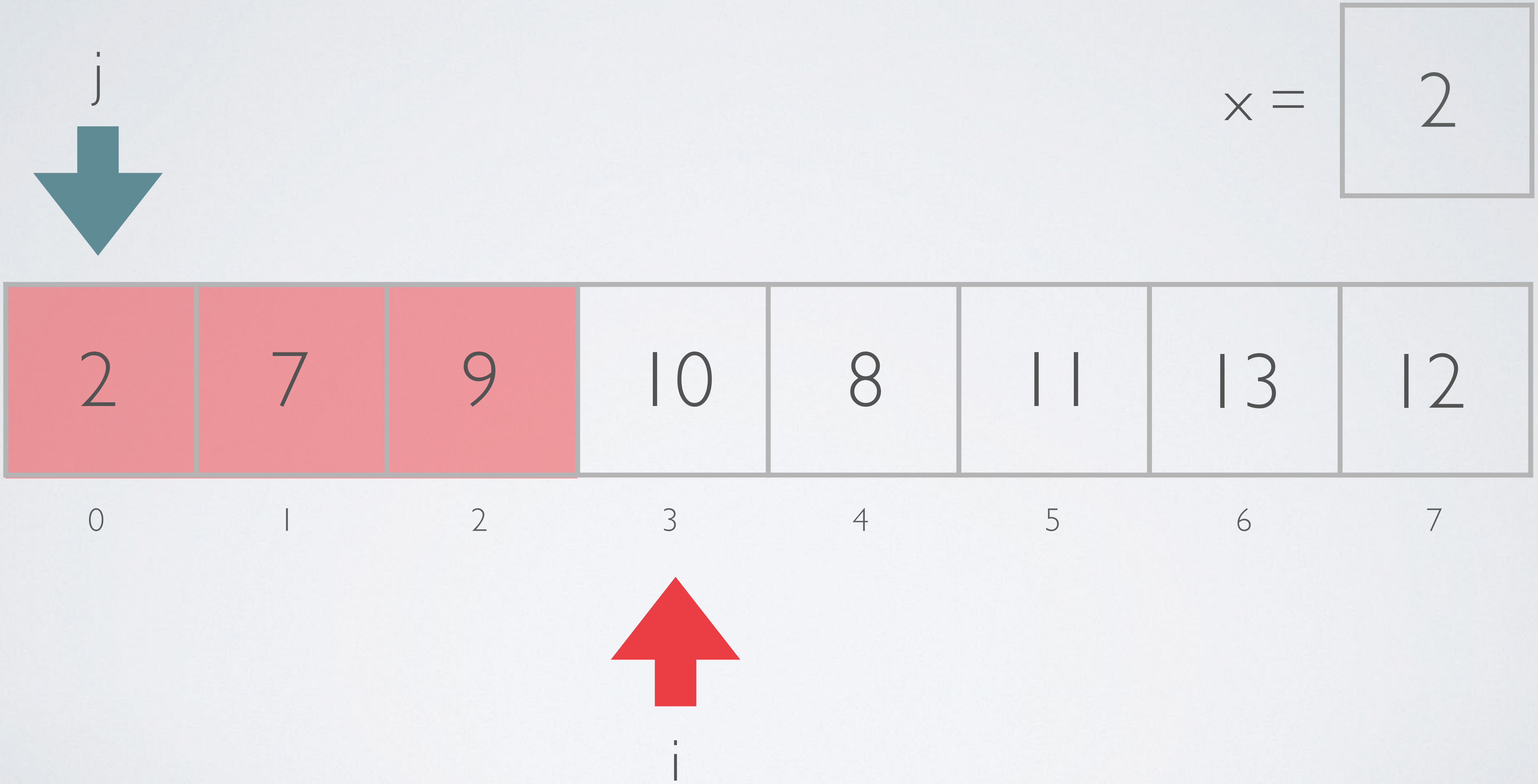
j

x = 2

7	7	9	10	8	11	13	12
0	1	2	3	4	5	6	7

i

TRI PAR INSERTION



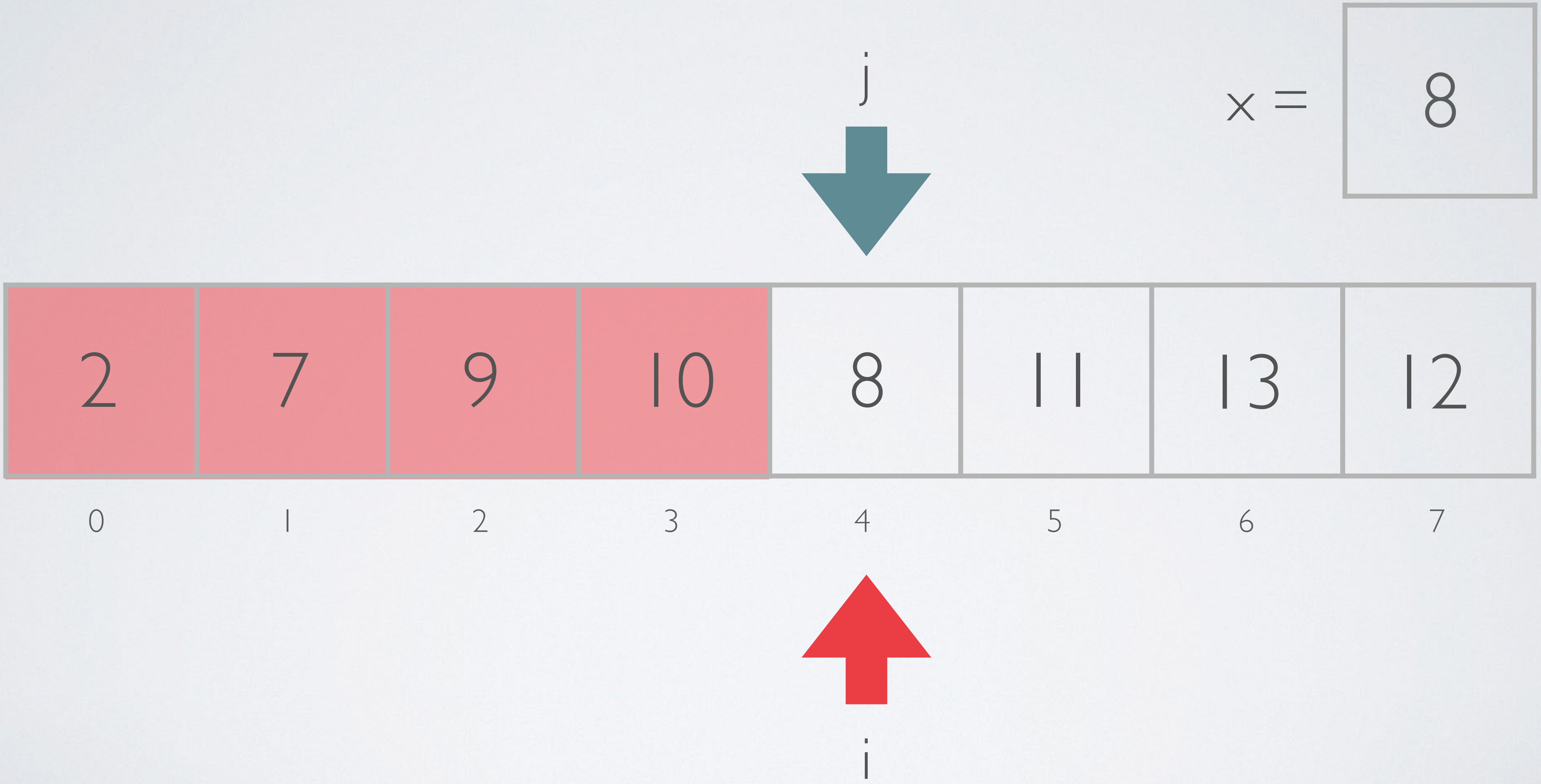
j

x = 2

2	7	9	10	8	11	13	12
0	1	2	3	4	5	6	7

i

TRI PAR INSERTION



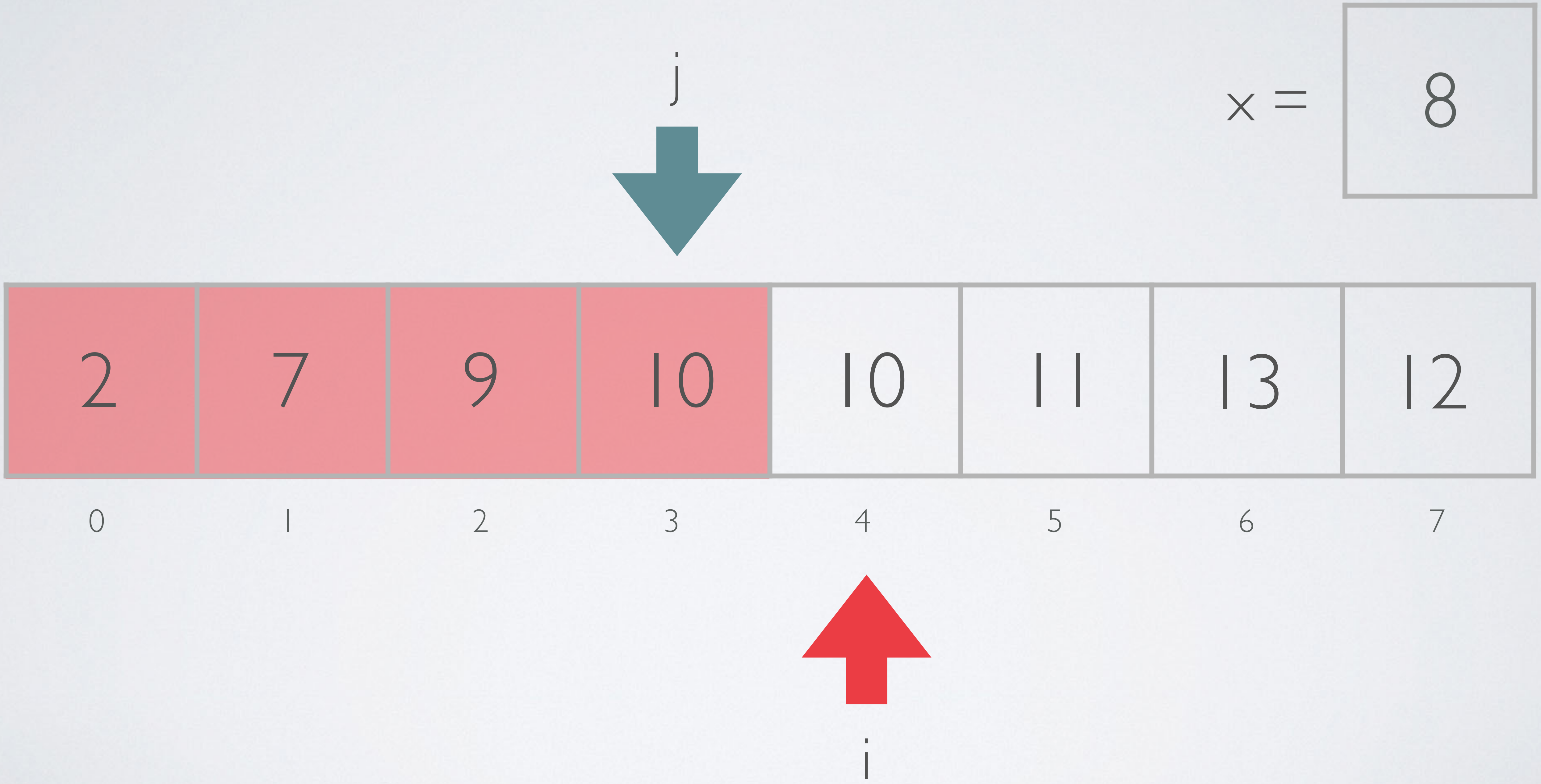
j

x = 8

2	7	9	10	8	11	13	12
0	1	2	3	4	5	6	7

i

TRI PAR INSERTION



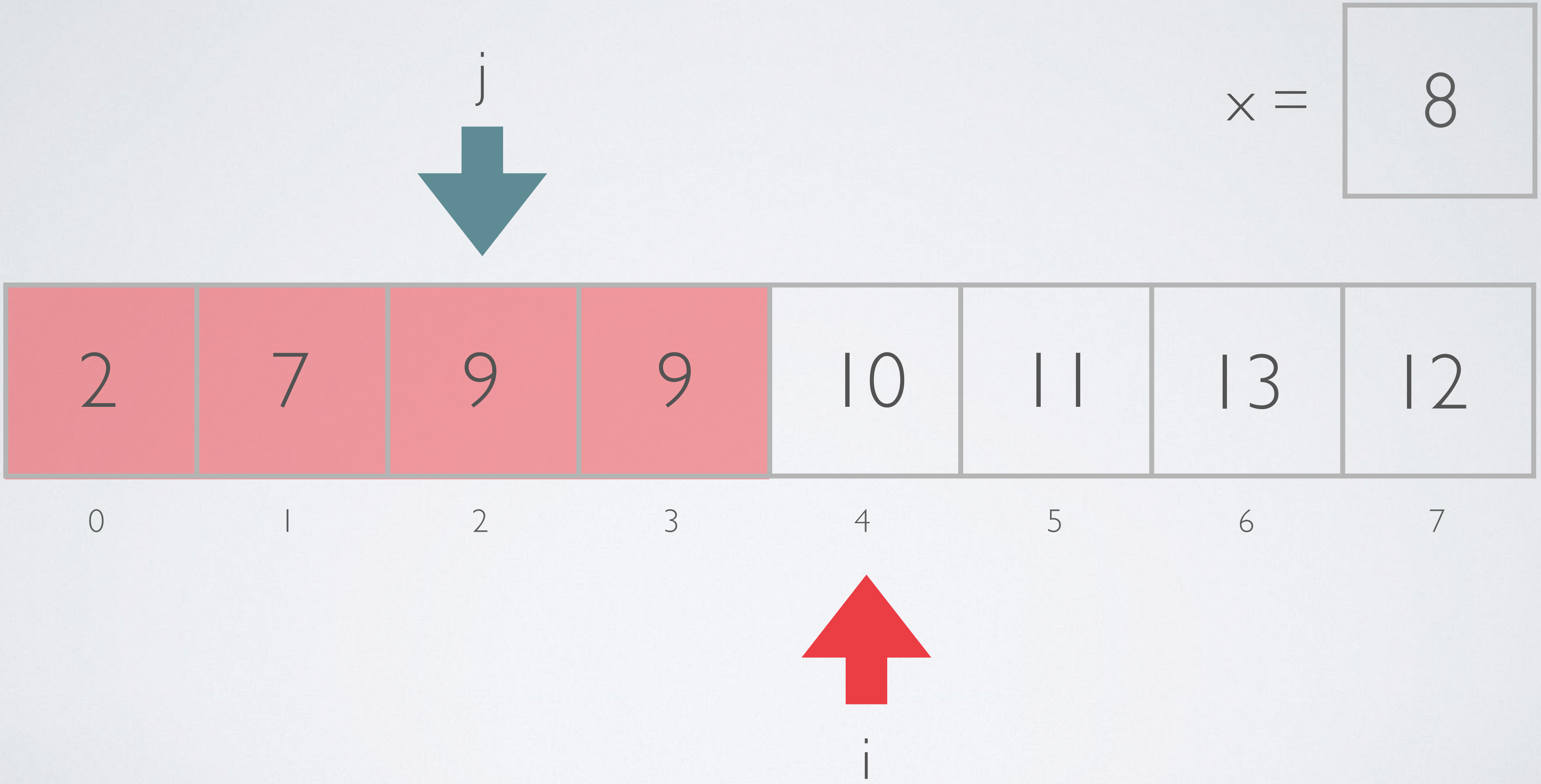
j

x = 8

2	7	9	10	10	11	13	12
0	1	2	3	4	5	6	7

i

TRI PAR INSERTION



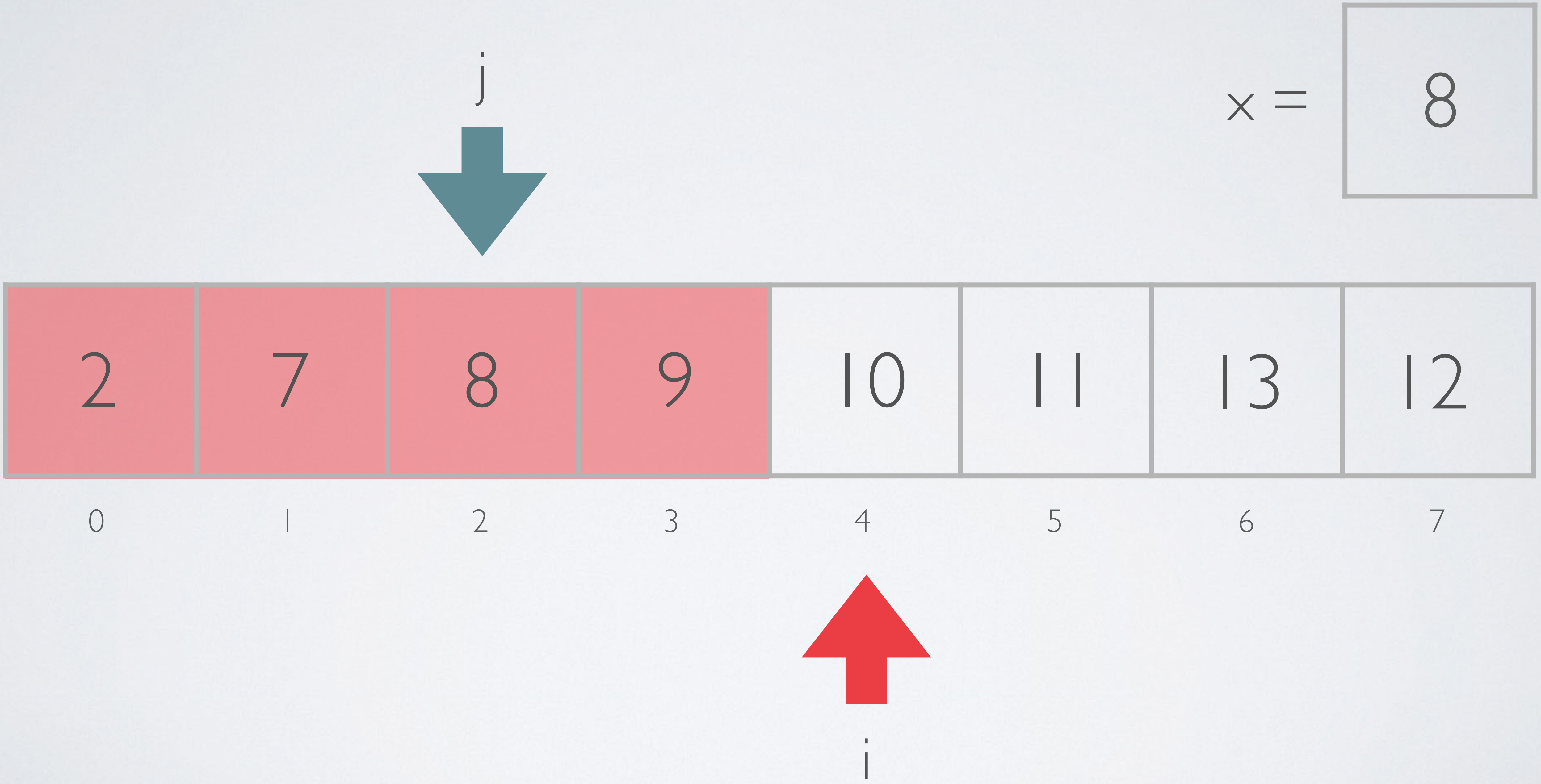
j

x = 8

2	7	9	9	10	11	13	12
0	1	2	3	4	5	6	7

i

TRI PAR INSERTION



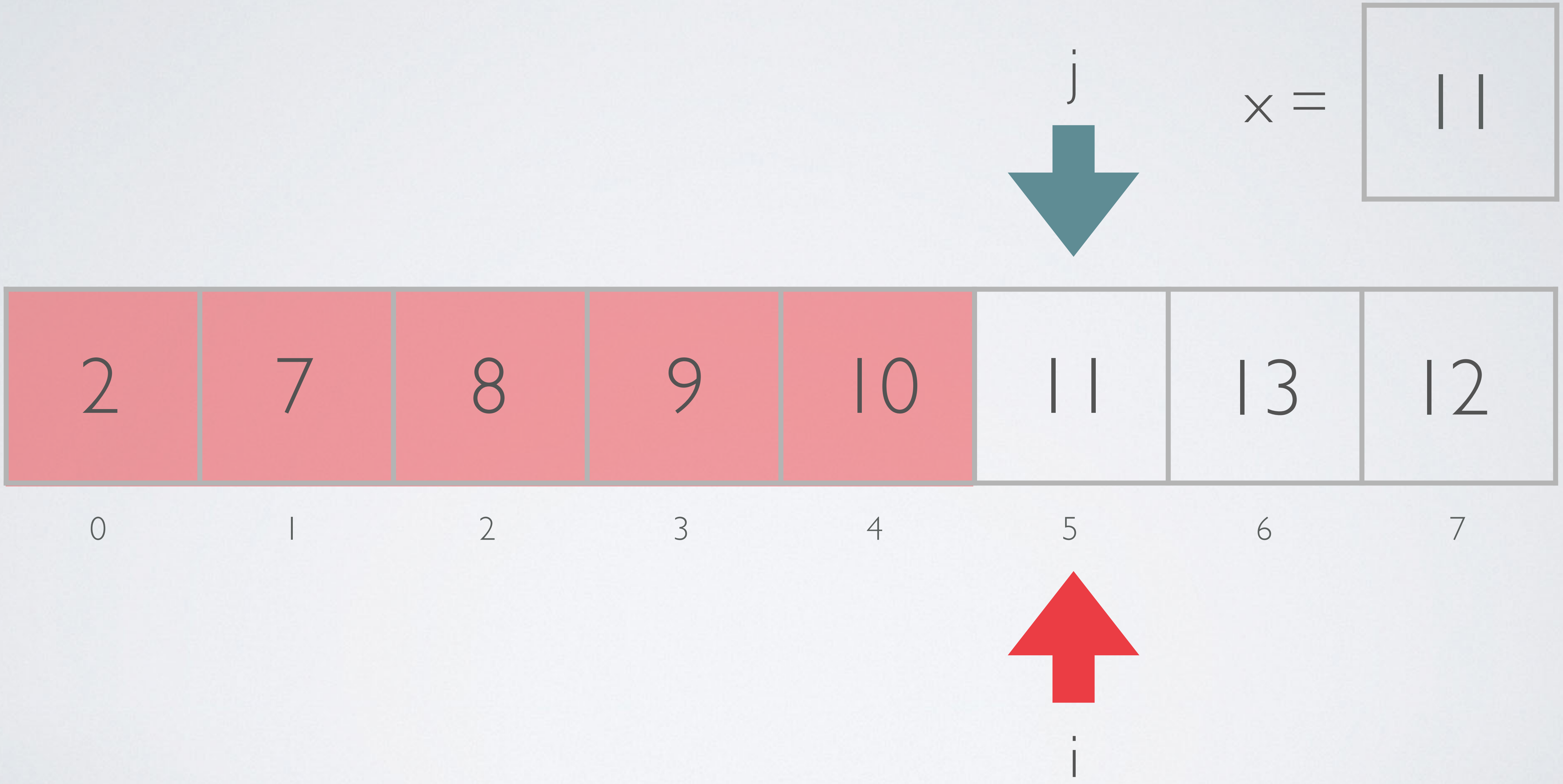
x = 8

j

2	7	8	9	10	11	13	12
0	1	2	3	4	5	6	7

i

TRI PAR INSERTION



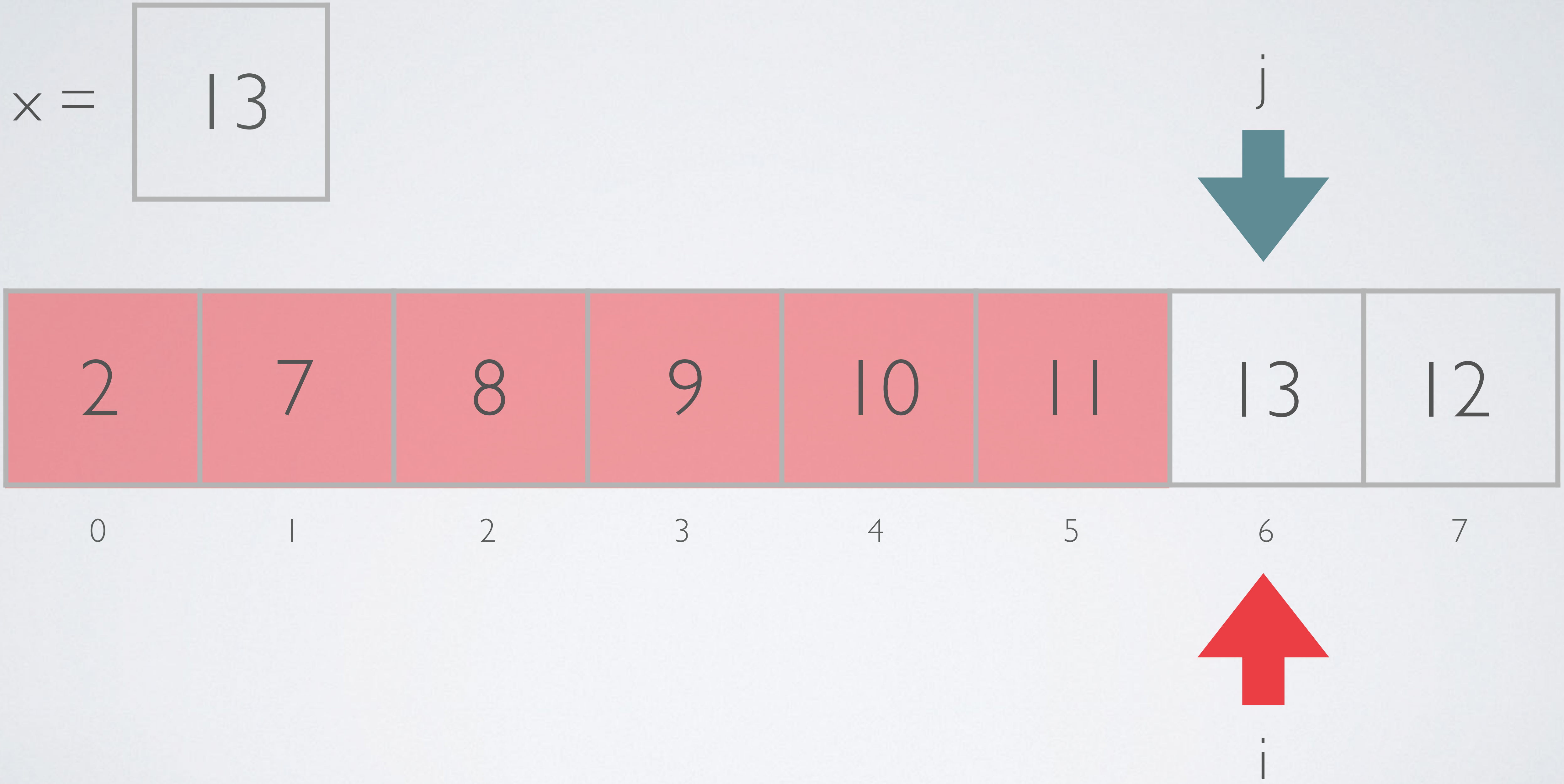
j

x = 11

2	7	8	9	10	11	13	12
0	1	2	3	4	5	6	7

i

TRI PAR INSERTION



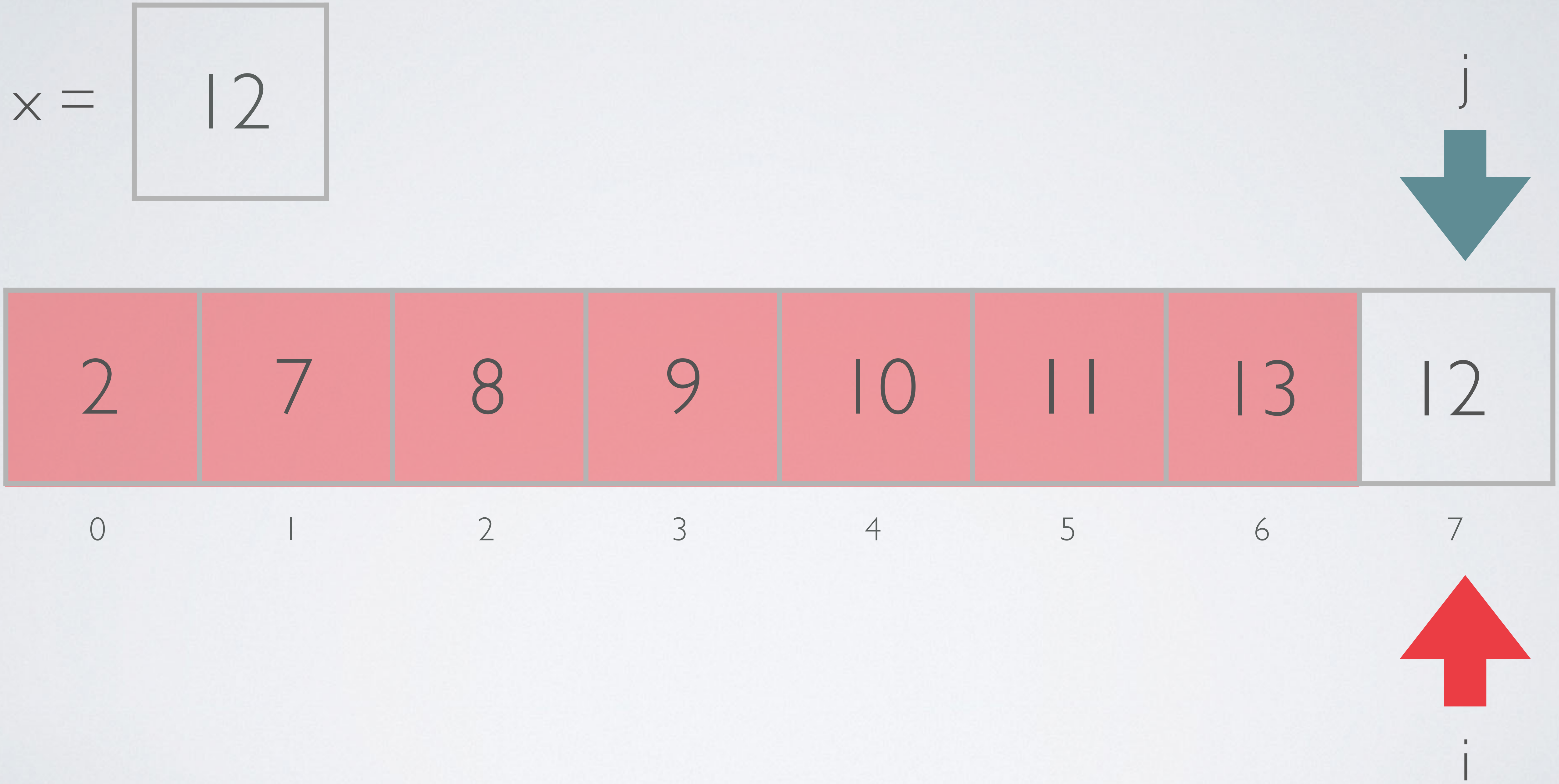
x = 13

j

0	1	2	3	4	5	6	7
2	7	8	9	10	11	13	12

i

TRI PAR INSERTION



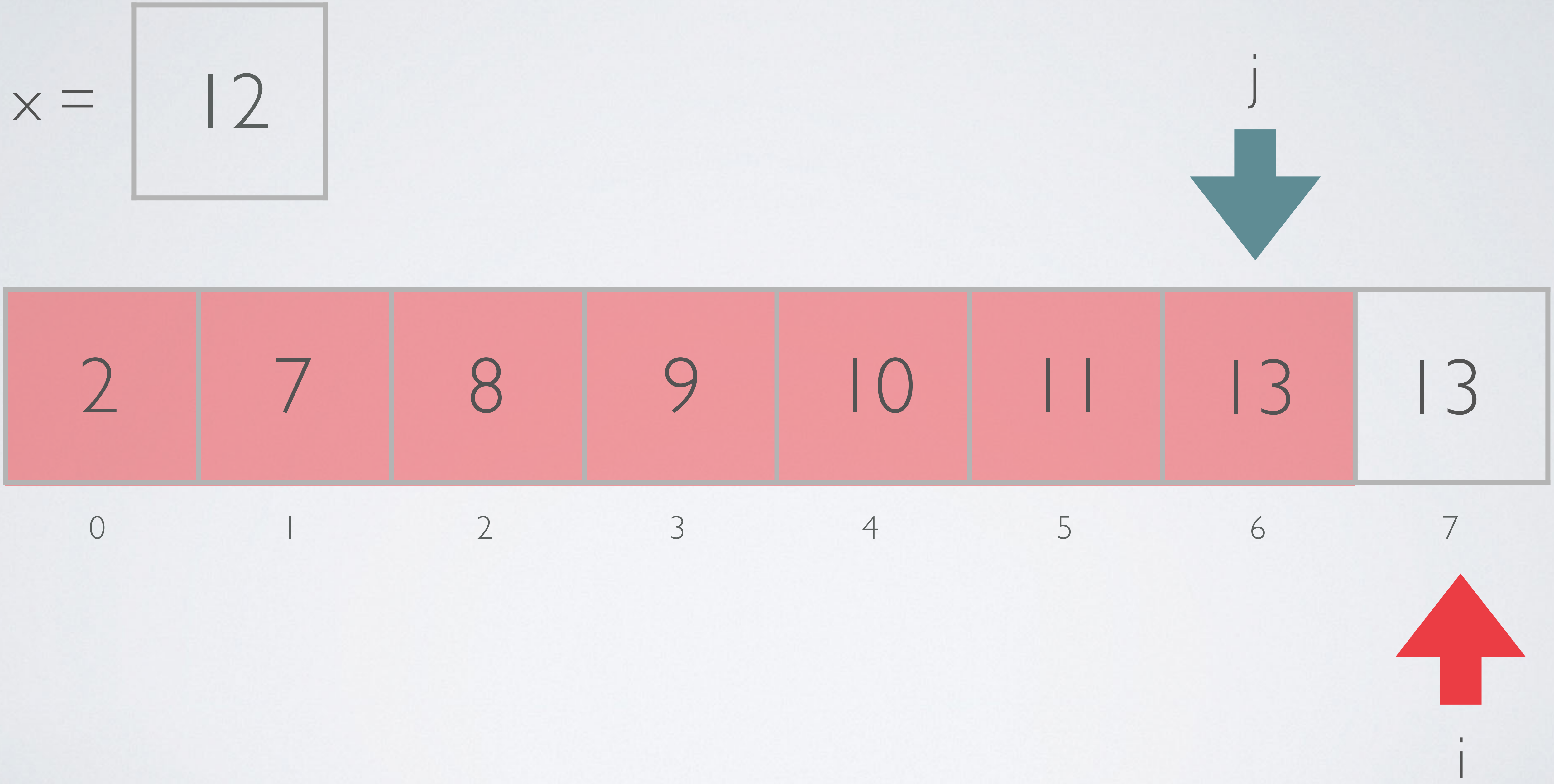
x = 12

j

2	7	8	9	10	11	13	12
0	1	2	3	4	5	6	7

i

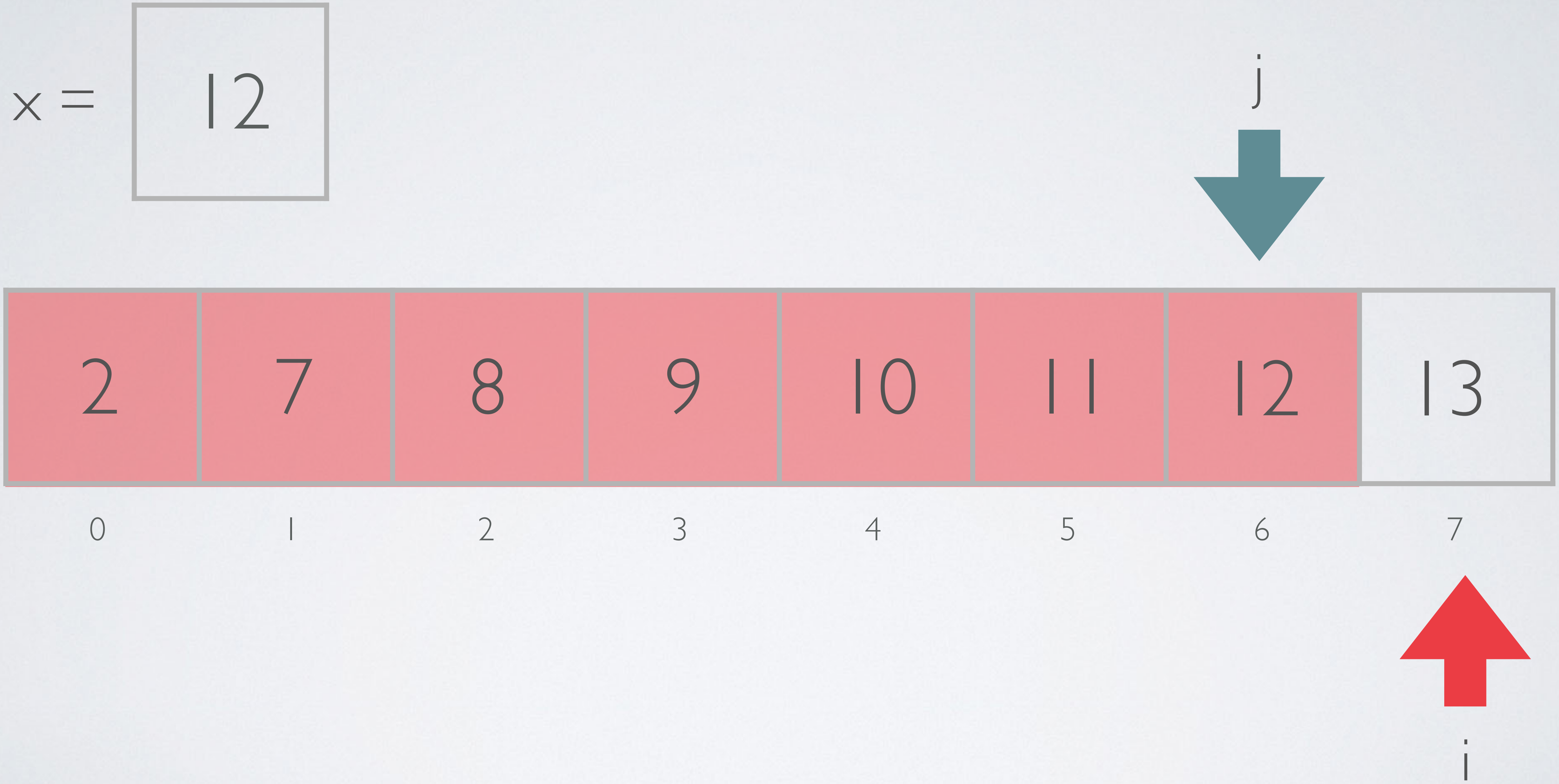
TRI PAR INSERTION



x = 12

0	1	2	3	4	5	6	7
2	7	8	9	10	11	13	13

TRI PAR INSERTION



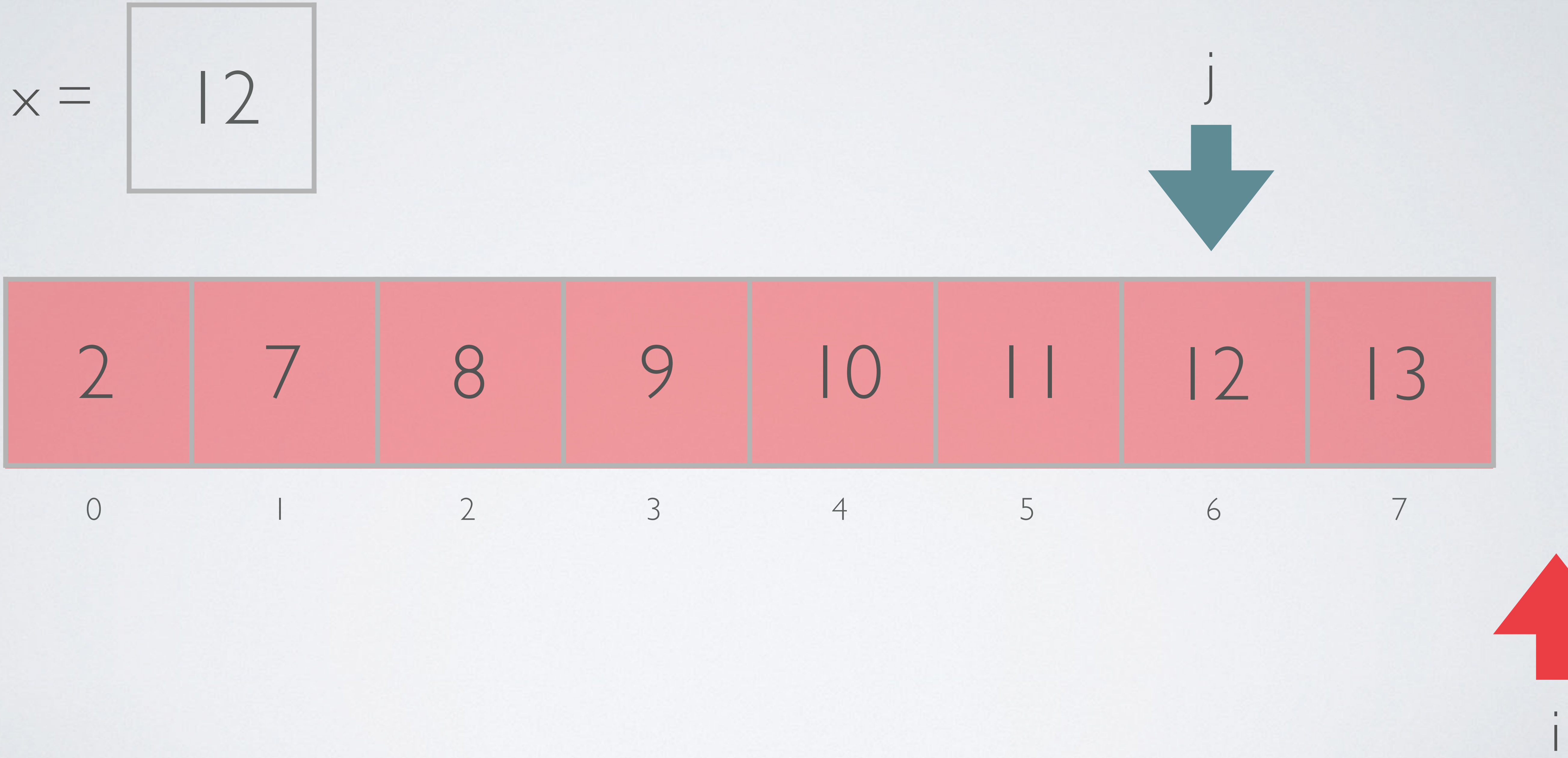
x = 12

j

2	7	8	9	10	11	12	13
0	1	2	3	4	5	6	7

i

TRI PAR INSERTION



x = 12

j

2	7	8	9	10	11	12	13
0	1	2	3	4	5	6	7

i

TERMINAISON

```
procedure trier-par-insertion(T)
    n ≔ longueur(T)
    pour i ≔ 1 à n − 1 faire
        x ≔ T[i]
        j ≔ i
        tant que j > 0
                et x < T[j − 1] faire
            (décaler d'un élément)
            T[j] ≔ T[j − 1]
            j ≔ j − 1
        (ici x ≥ T[j − 1] ou bien j = 0)
        T[j] ≔ x
```

- La boucle **pour** termine toujours
- La boucle **tant que** termine (au pire) quand $j = 0$

CORRECTION

```
procedure trier-par-insertion(T)
    n ≔ longueur(T)
    pour i ≔ 1 à n – 1 faire
        x ≔ T[i]
        j ≔ i
        tant que j > 0
                et x < T[j – 1] faire
            (décaler d’un élément)
            T[j] ≔ T[j – 1]
            j ≔ j – 1
        (ici x ≥ T[j – 1] ou bien j = 0)
        T[j] ≔ x
```

- Le sous-tableau T[0, …, i – 1] est trié au début de la boucle **pour**

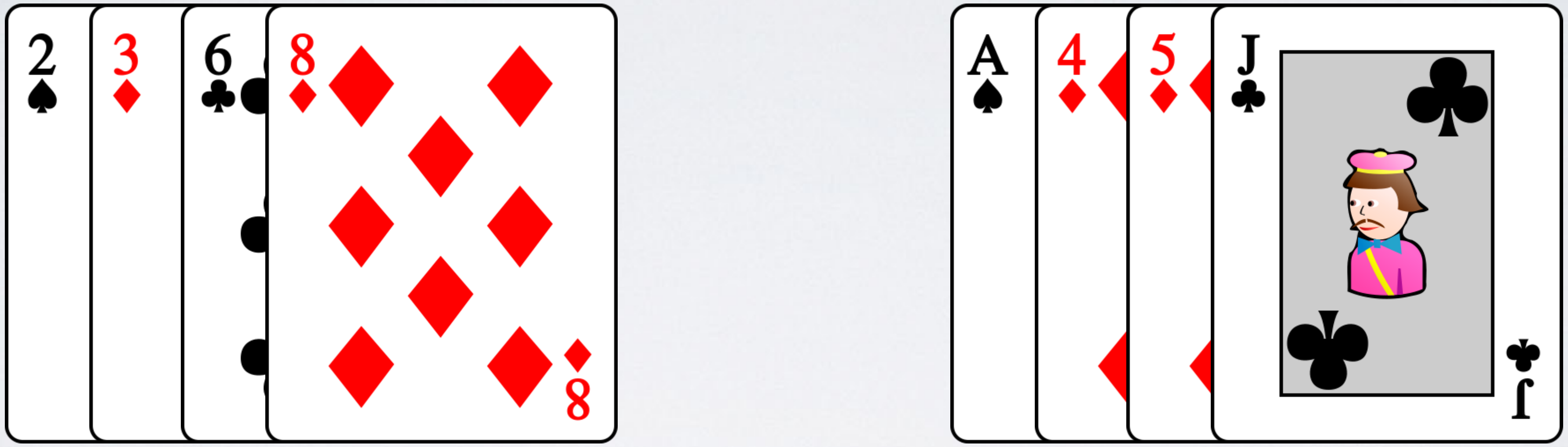
EFFICACITÉ

```
procedure trier-par-insertion(T)
    n ≔ longueur(T)
    pour i ≔ 1 à n – 1 faire
        x ≔ T[i]
        j ≔ i
        tant que j > 0
                et x < T[j – 1] faire
            (décaler d'un élément)
            T[j] ≔ T[j – 1]
            j ≔ j – 1
        (ici x ≥ T[j – 1] ou bien j = 0)
        T[j] ≔ x
```

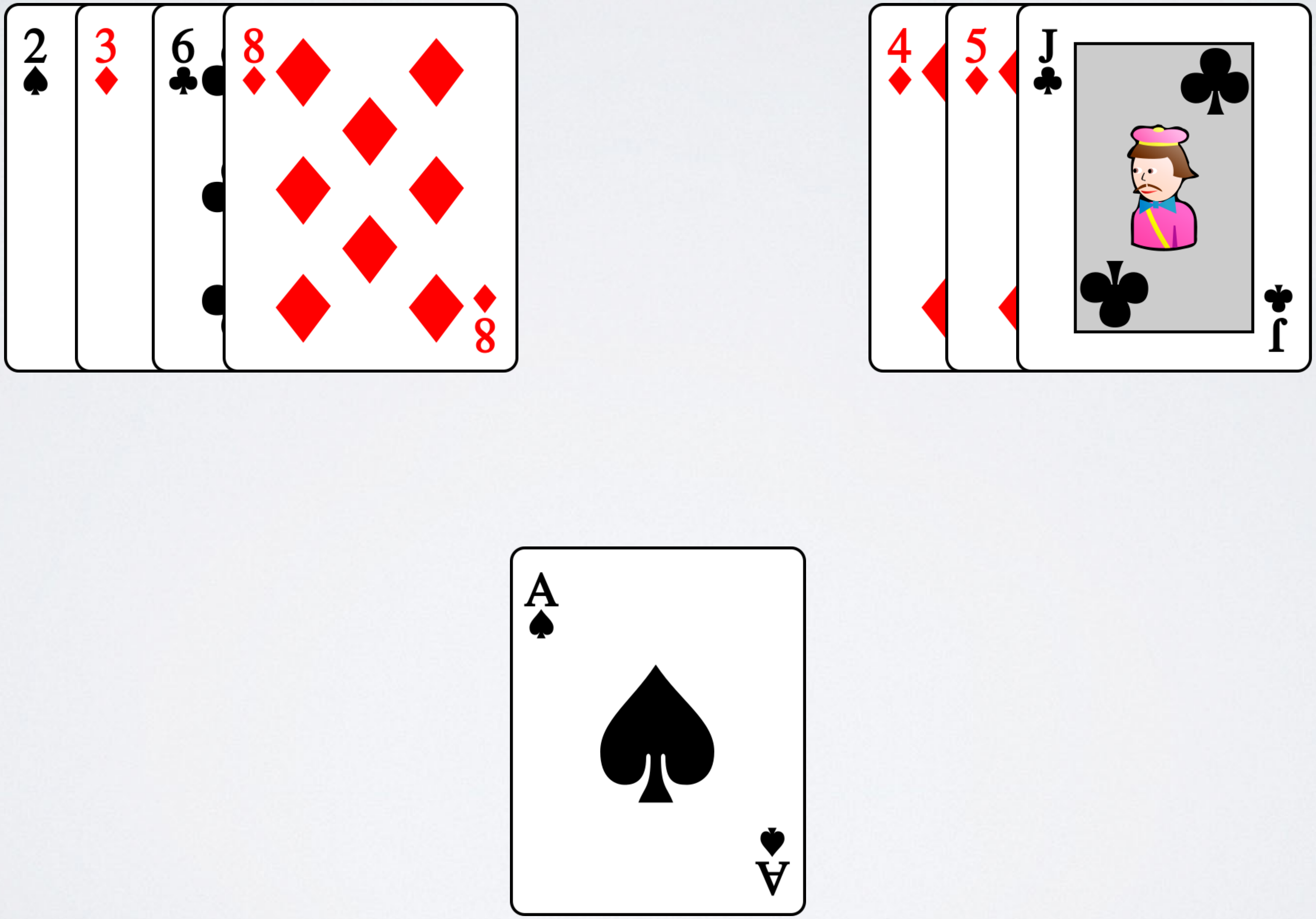
- $O(n)$ opérations dans le meilleur des cas
- $O(n^2)$ opérations dans le pire des cas

EST-CE QU’ON PEUT FAIRE MIEUX QUE ÇA ?

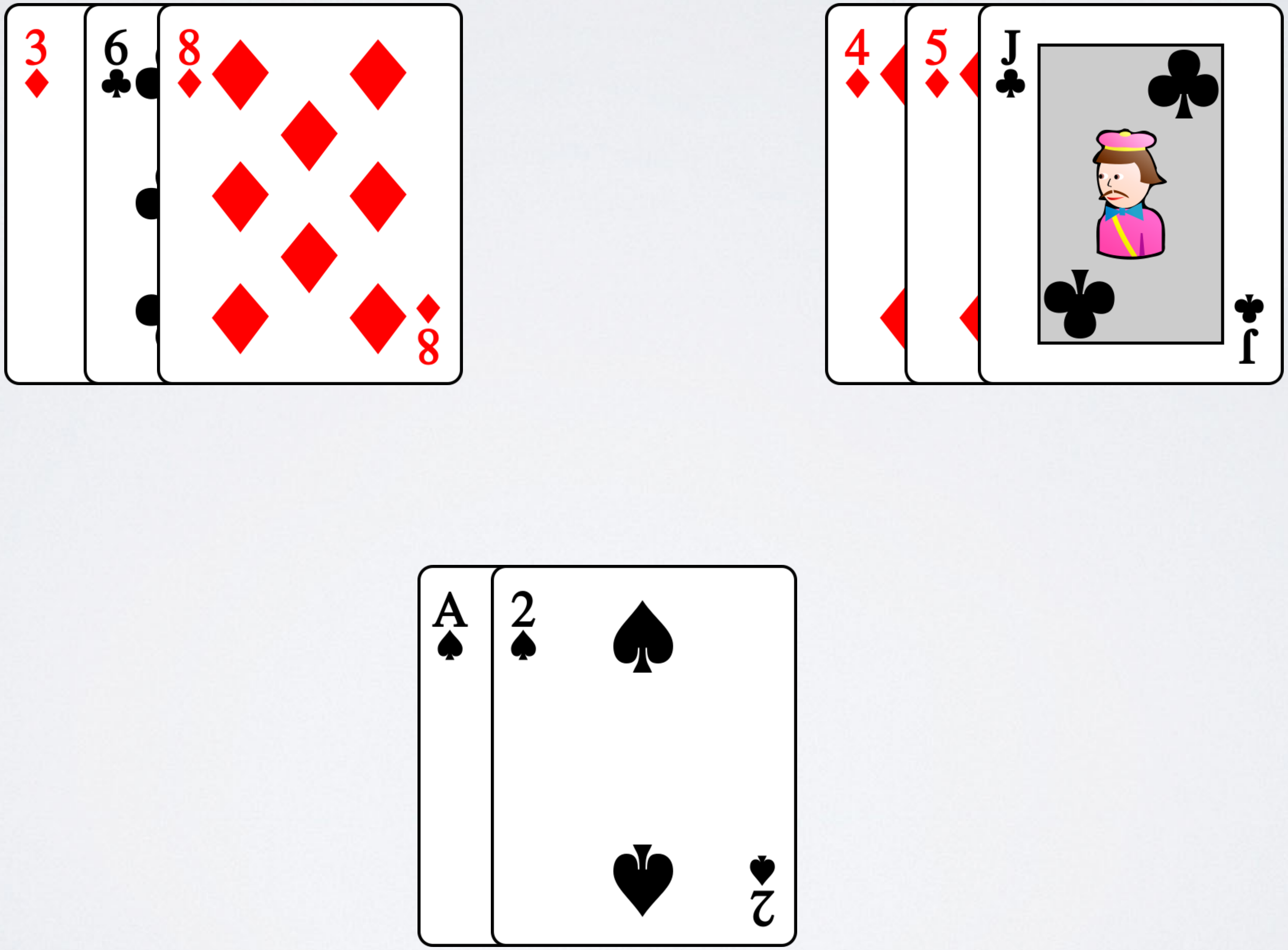
FUSION DE TABLEAUX TRIÉS



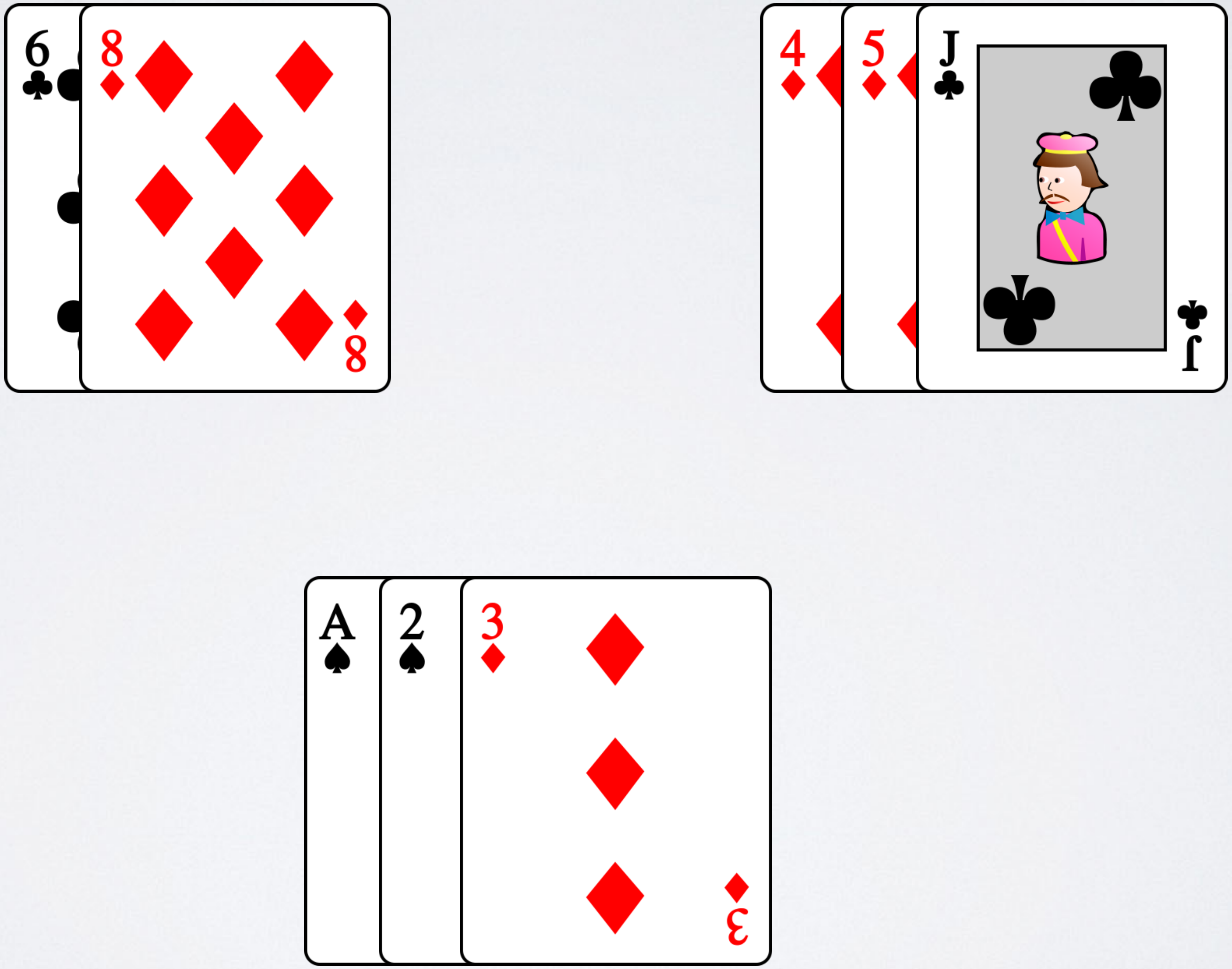
FUSION DE TABLEAUX TRIÉS



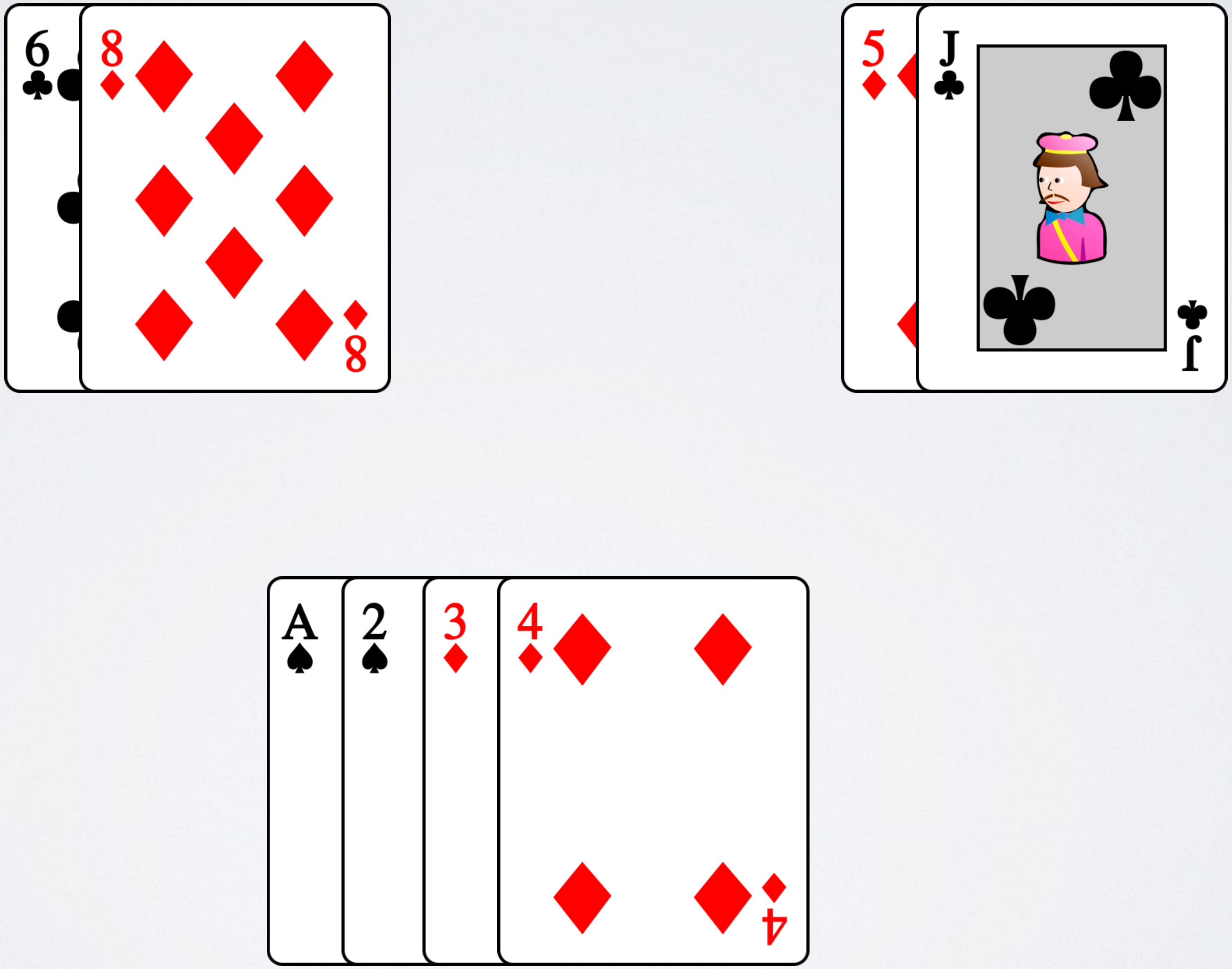
FUSION DE TABLEAUX TRIÉS



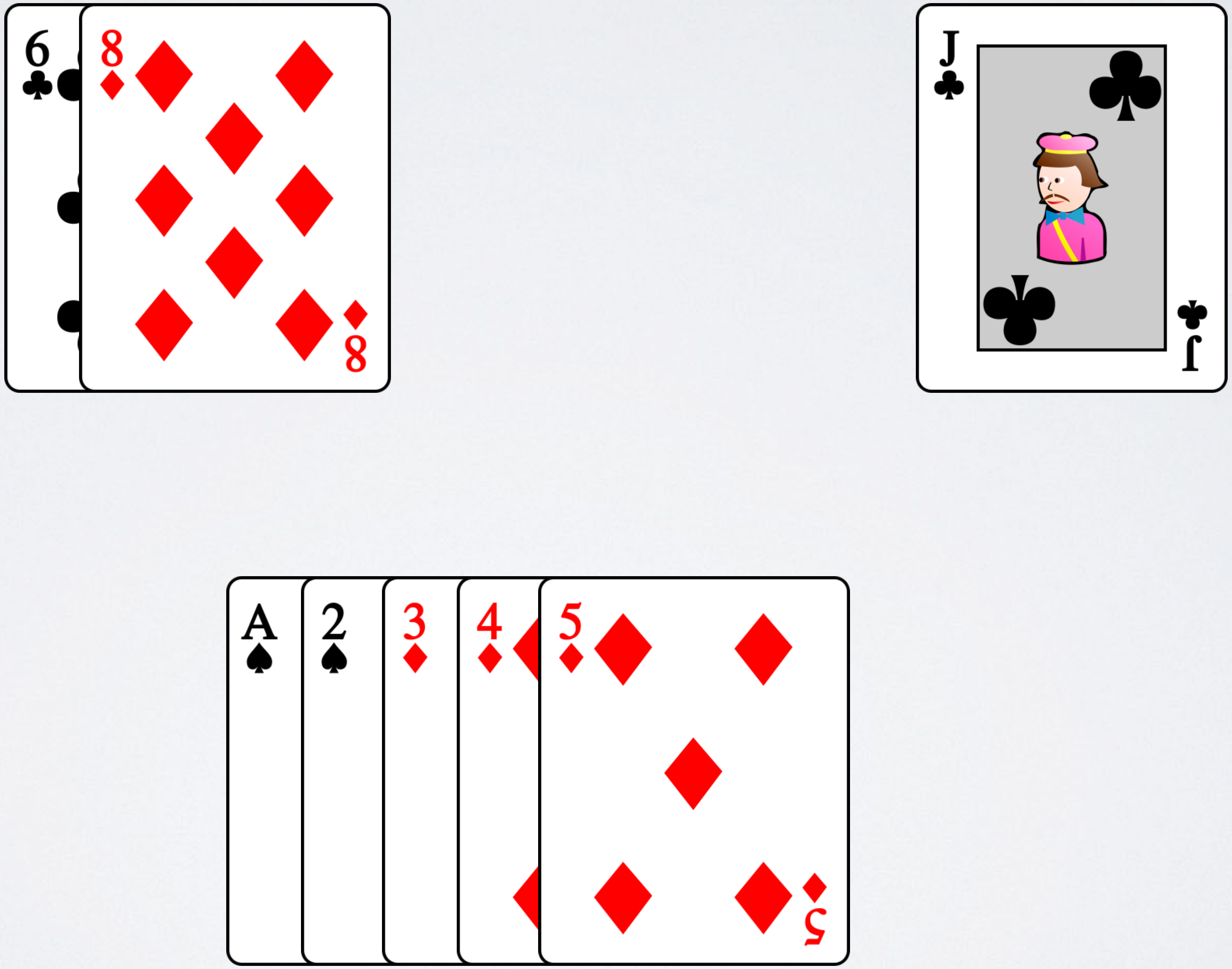
FUSION DE TABLEAUX TRIÉS



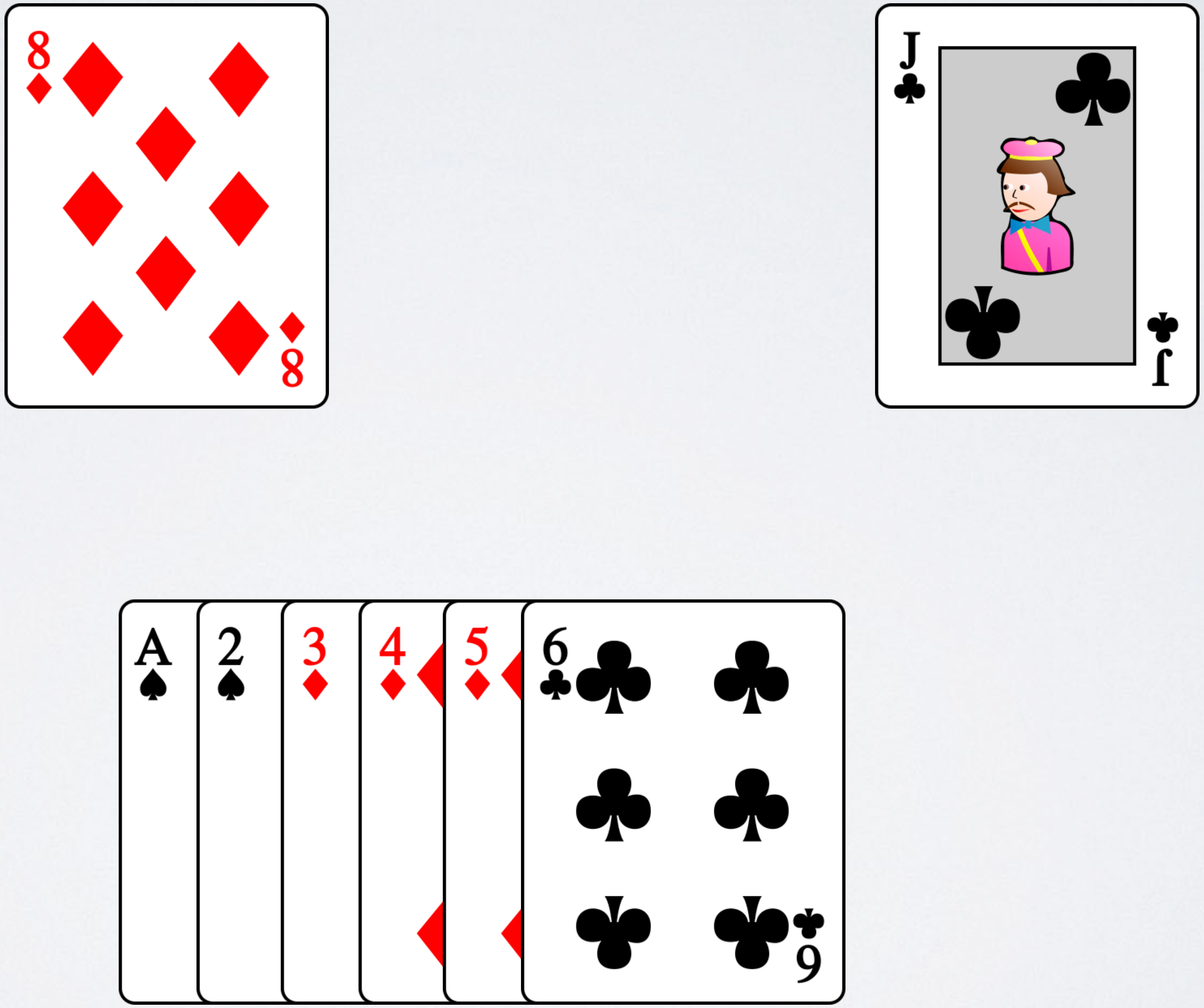
FUSION DE TABLEAUX TRIÉS



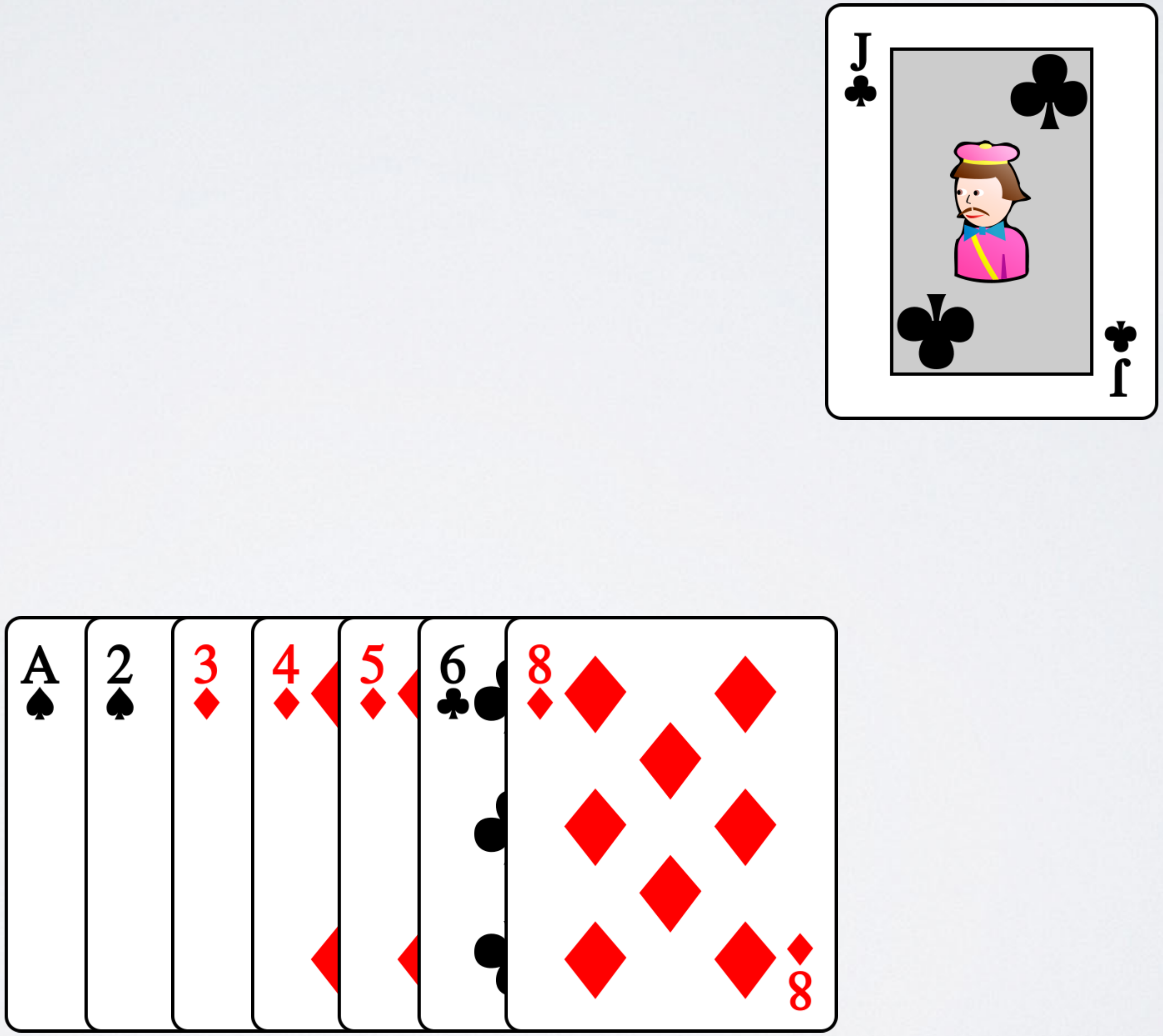
FUSION DE TABLEAUX TRIÉS



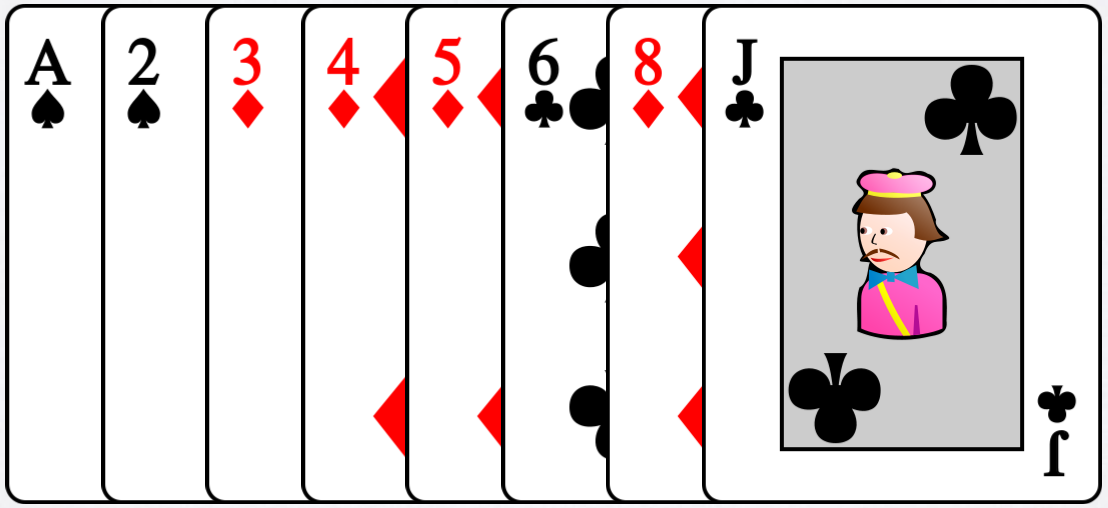
FUSION DE TABLEAUX TRIÉS



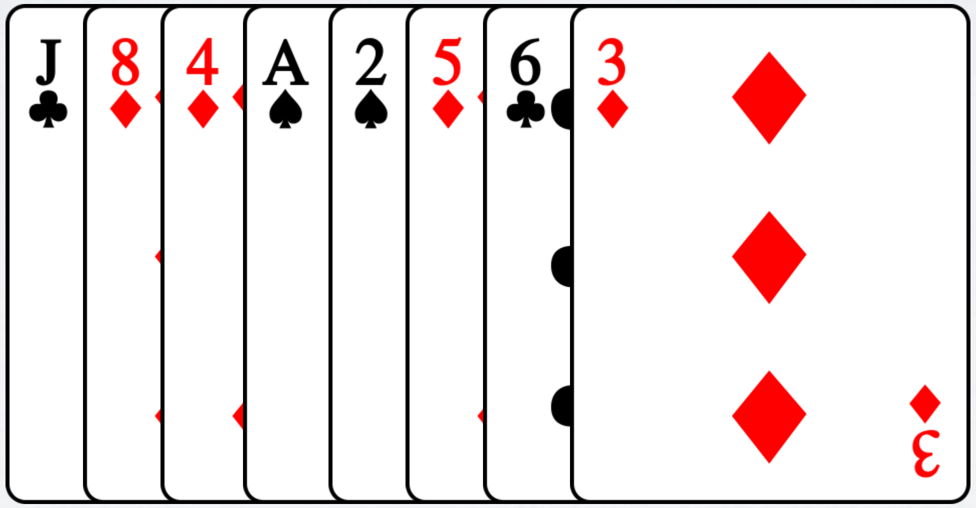
FUSION DE TABLEAUX TRIÉS



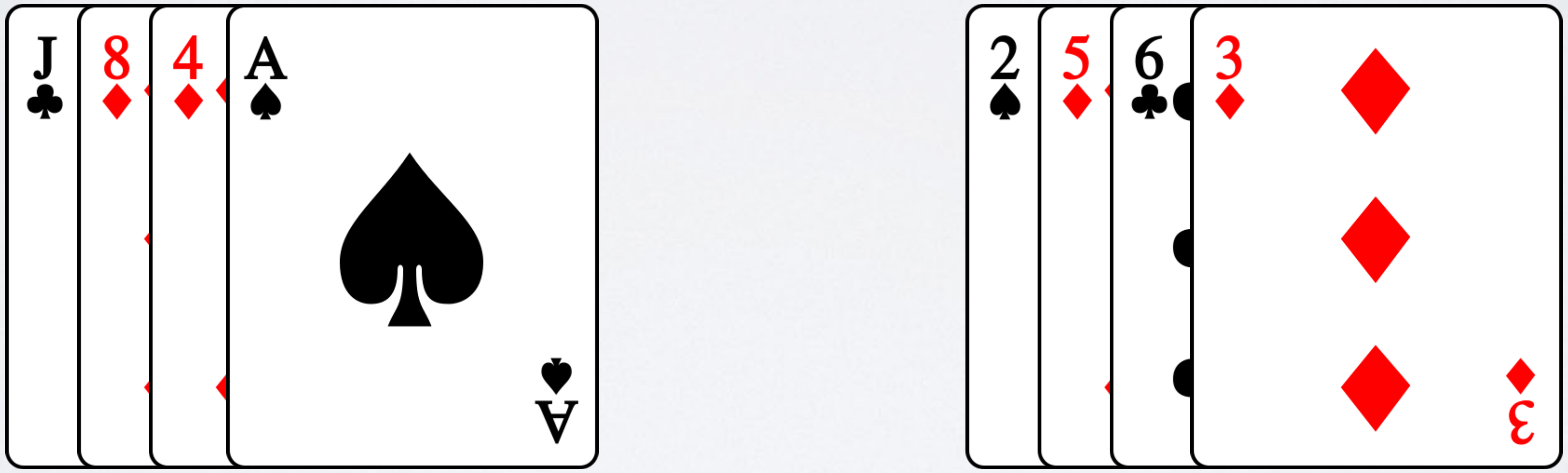
FUSION DE TABLEAUX TRIÉS



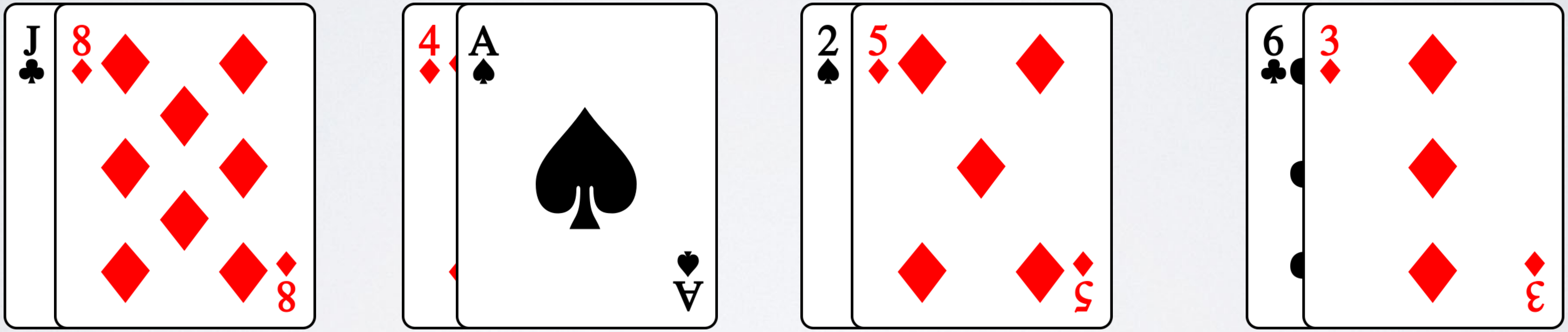
TRI FUSION



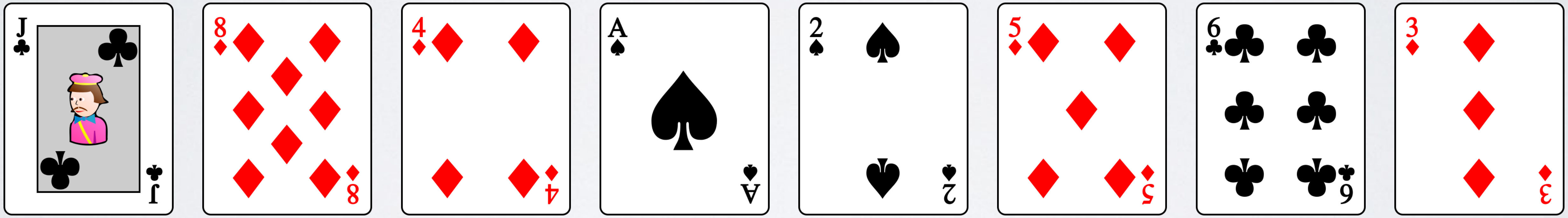
DIVISER



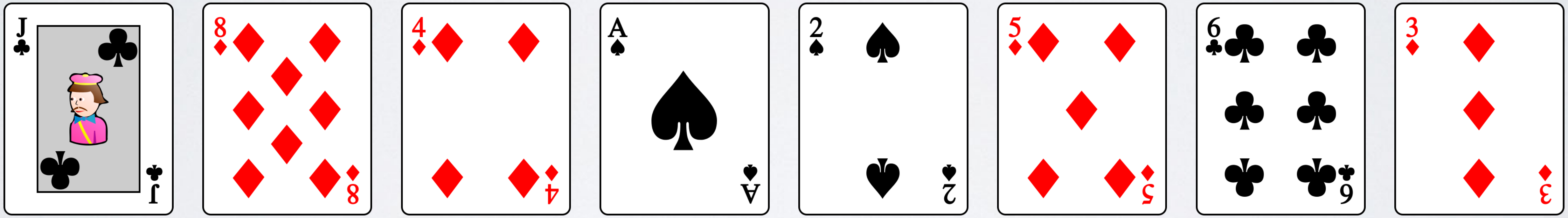
DIVISER



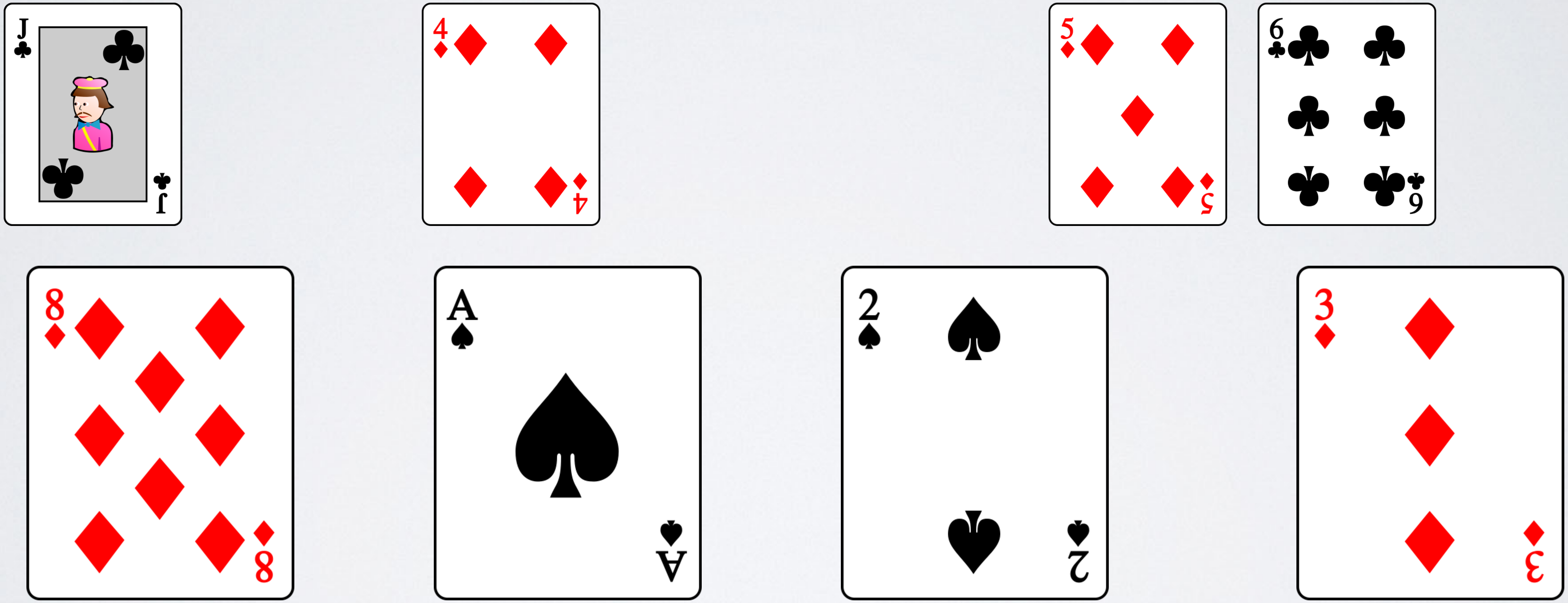
DIVISER



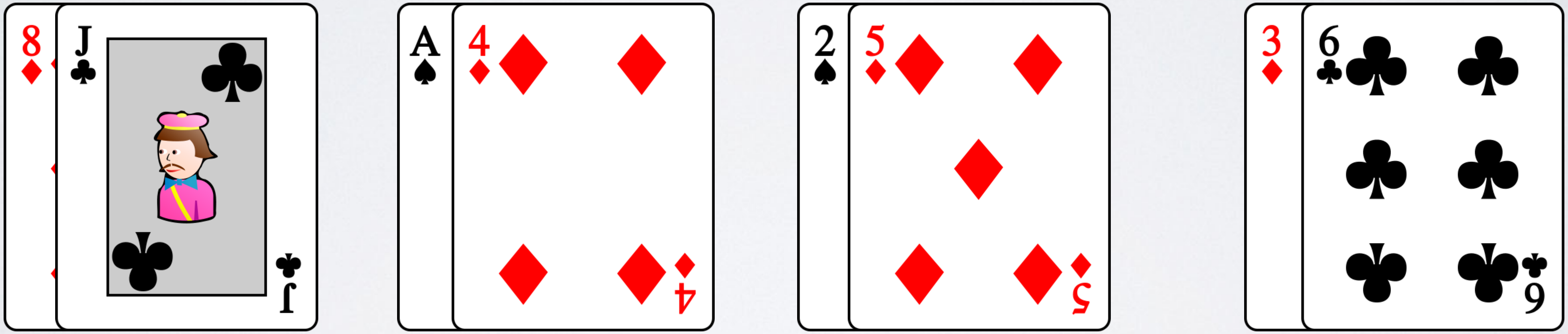
TOUS LES JEUX SONT TRIÉS !



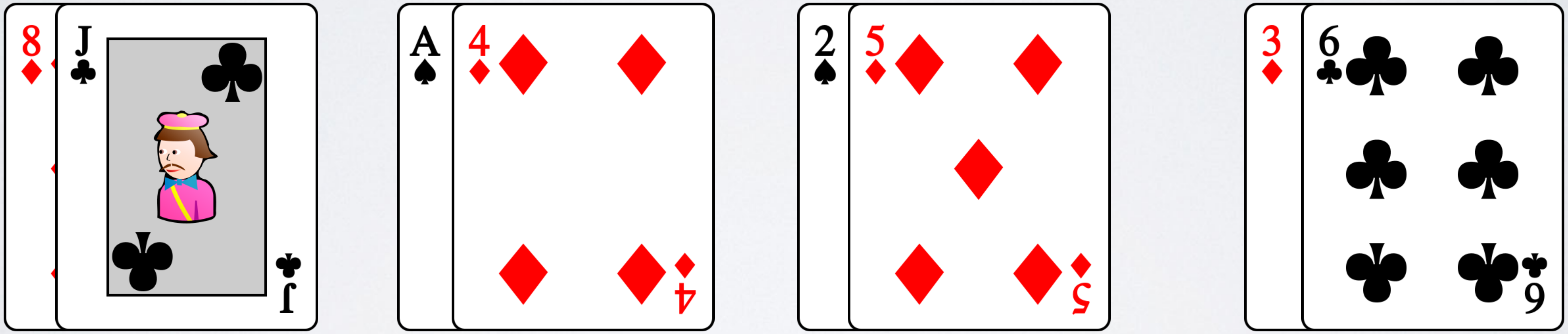
FUSIONNER



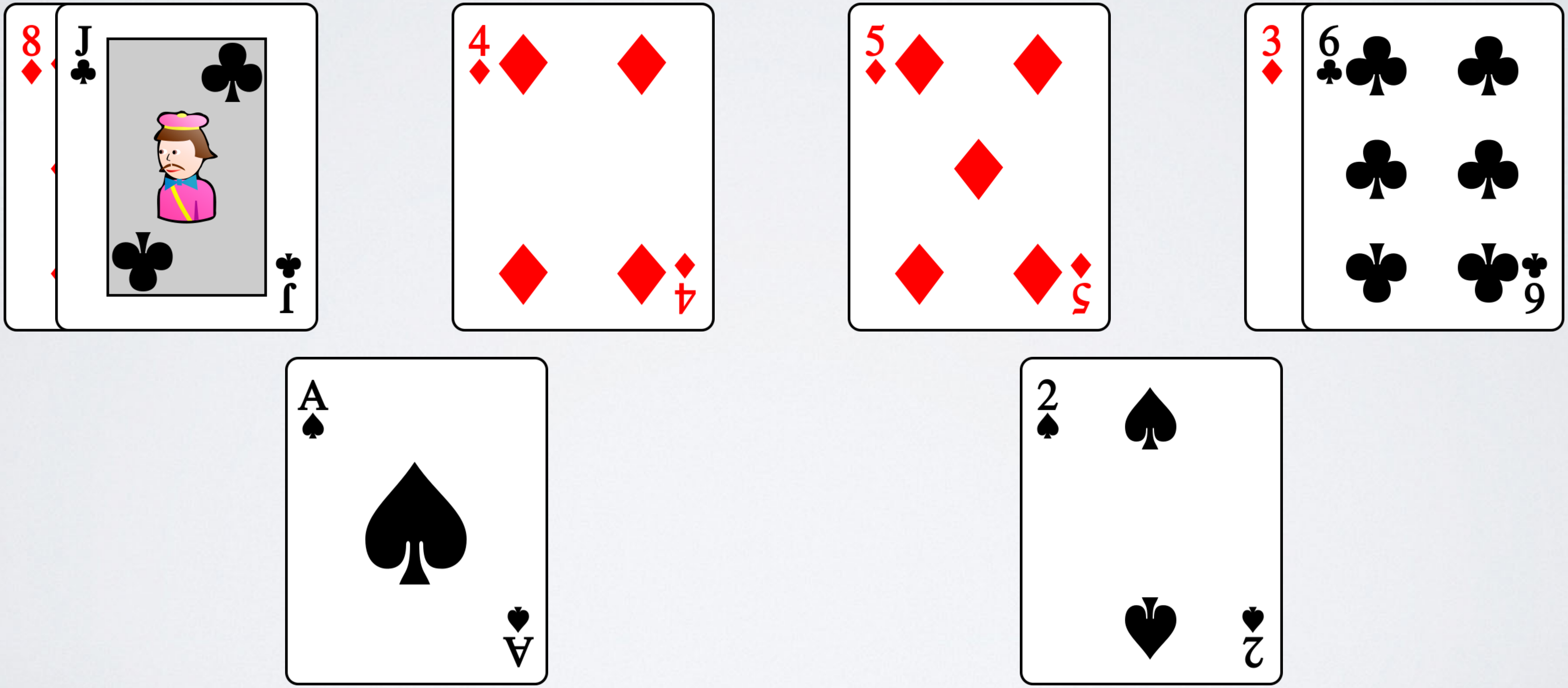
FUSIONNER



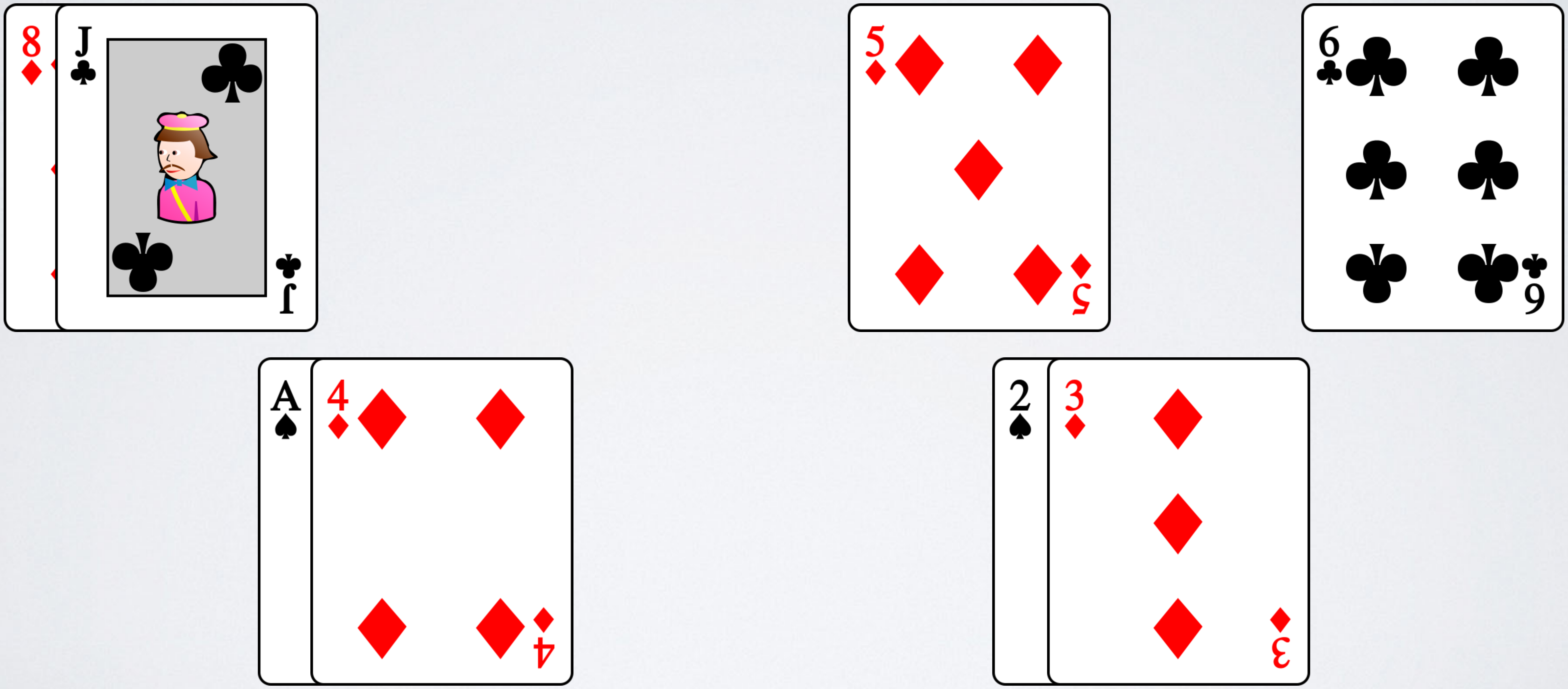
TOUS LES JEUX SONT TRIÉS !



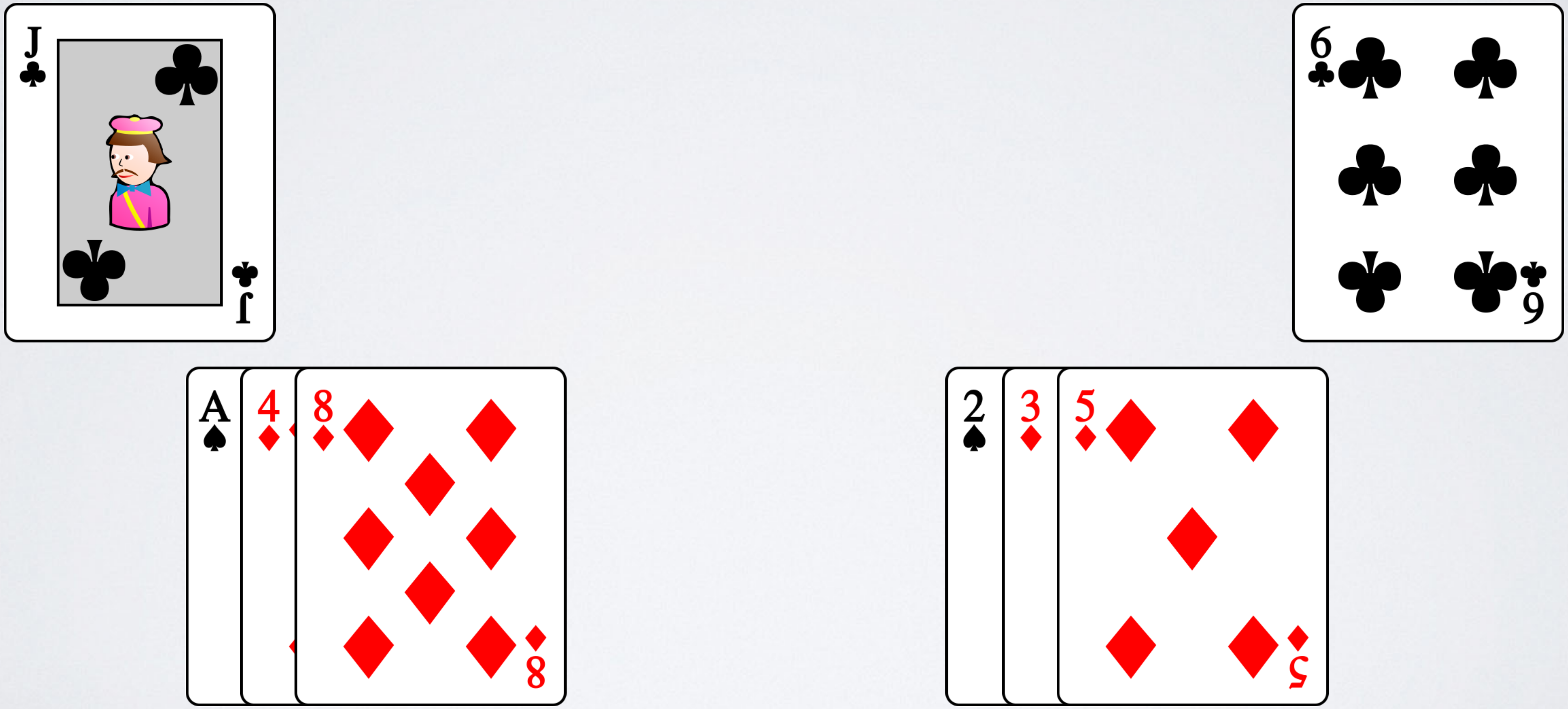
FUSIONNER



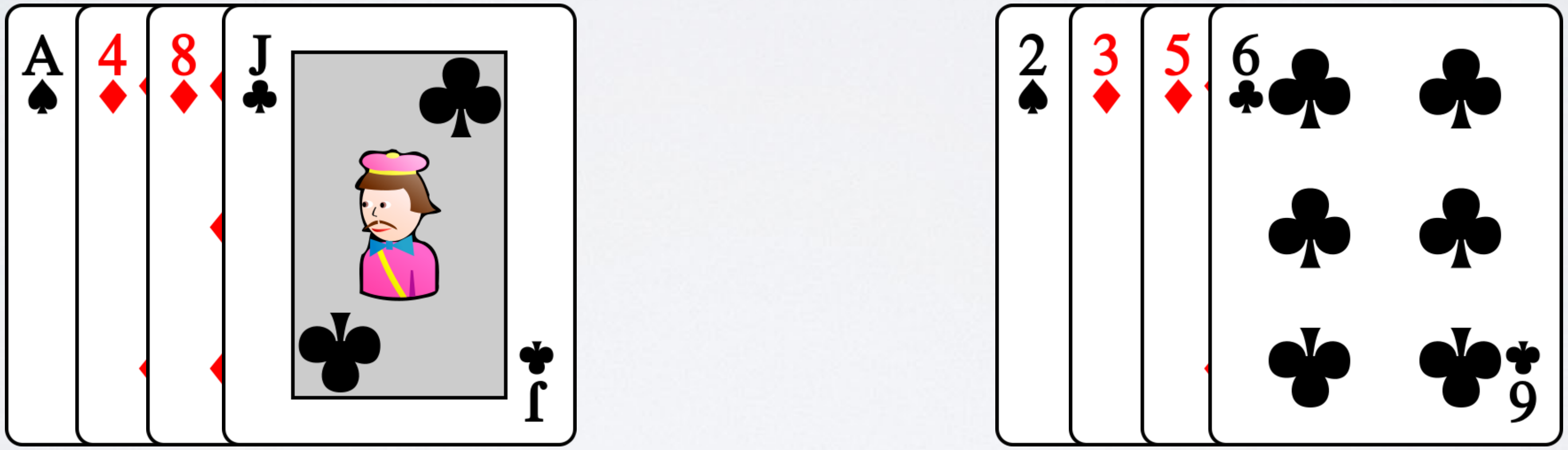
FUSIONNER



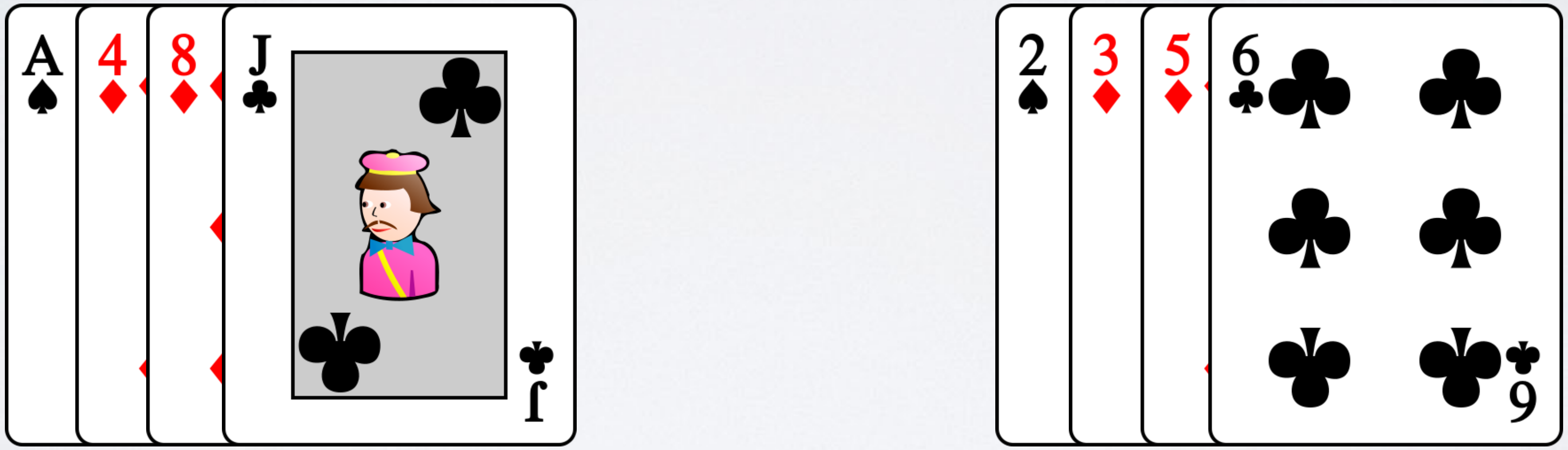
FUSIONNER



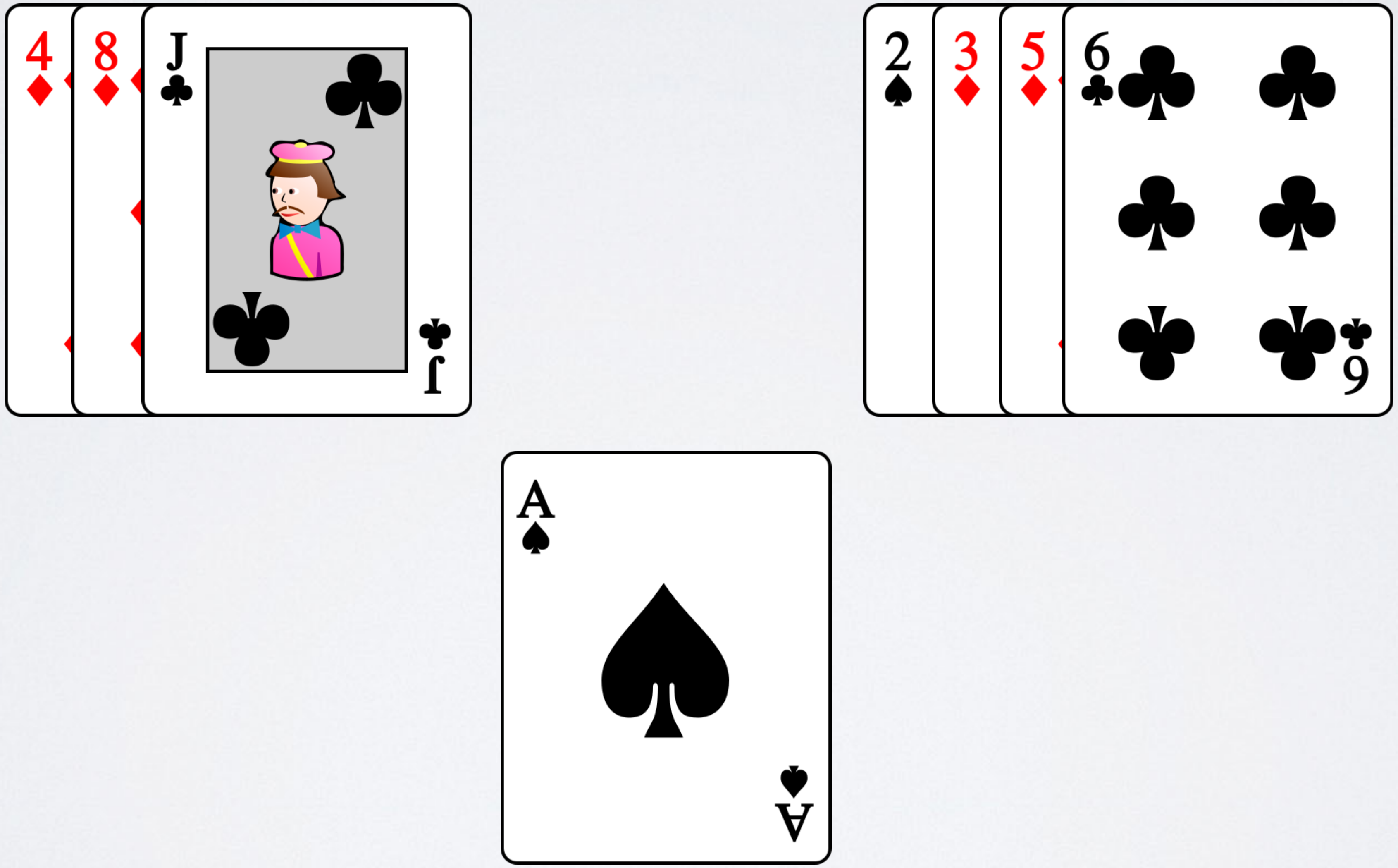
FUSIONNER



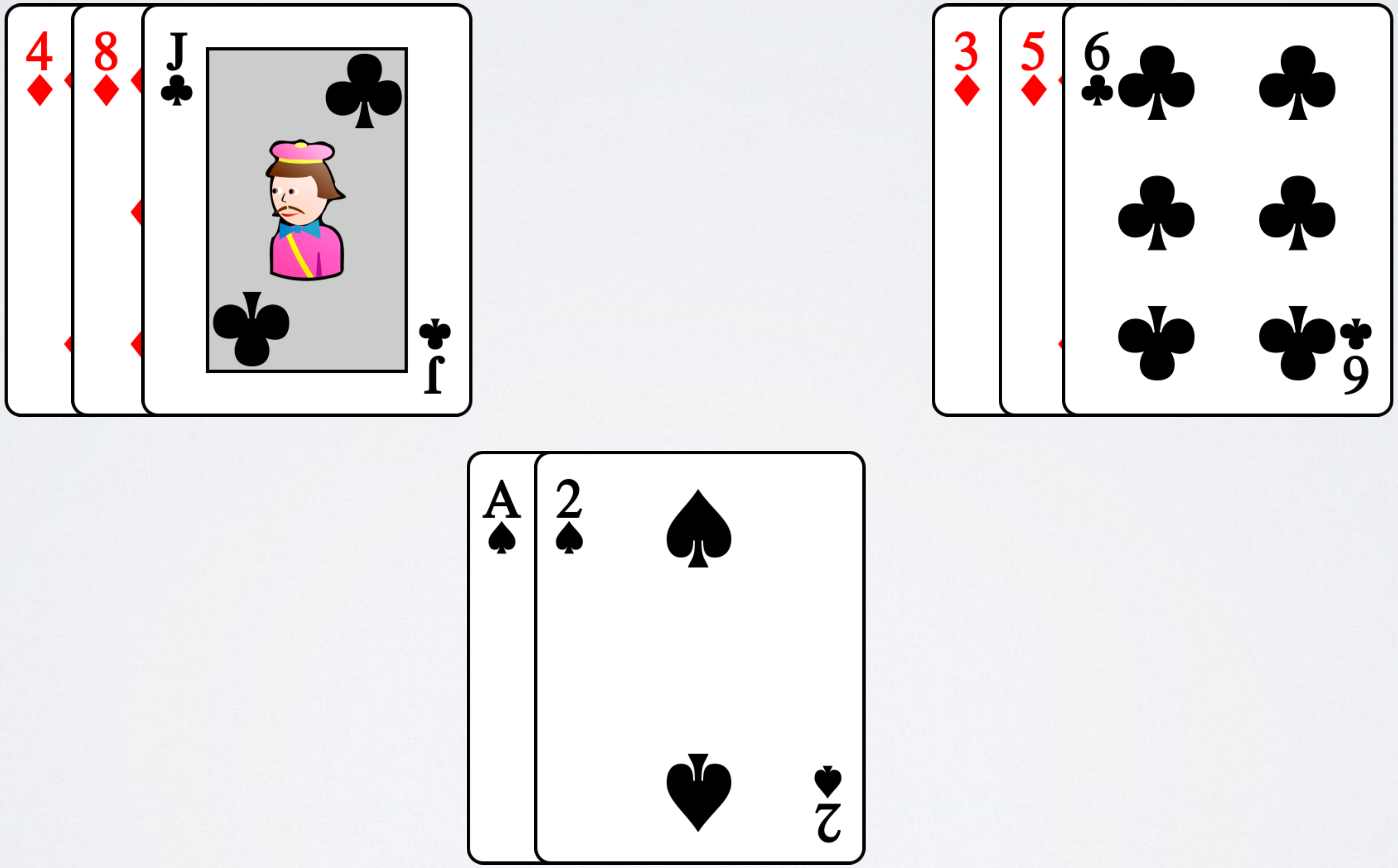
TOUS LES JEUX SONT TRIÉS !



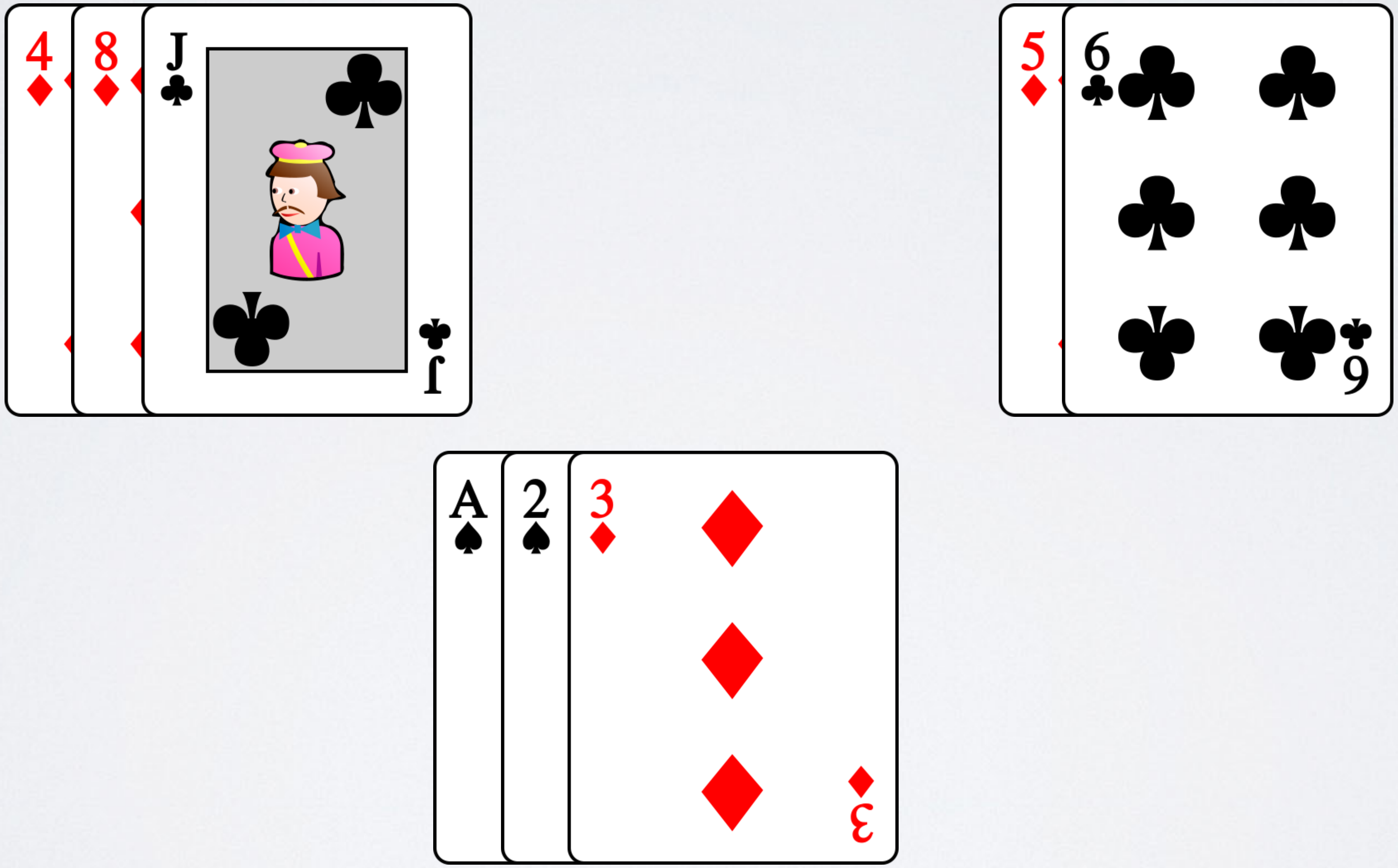
FUSIONNER



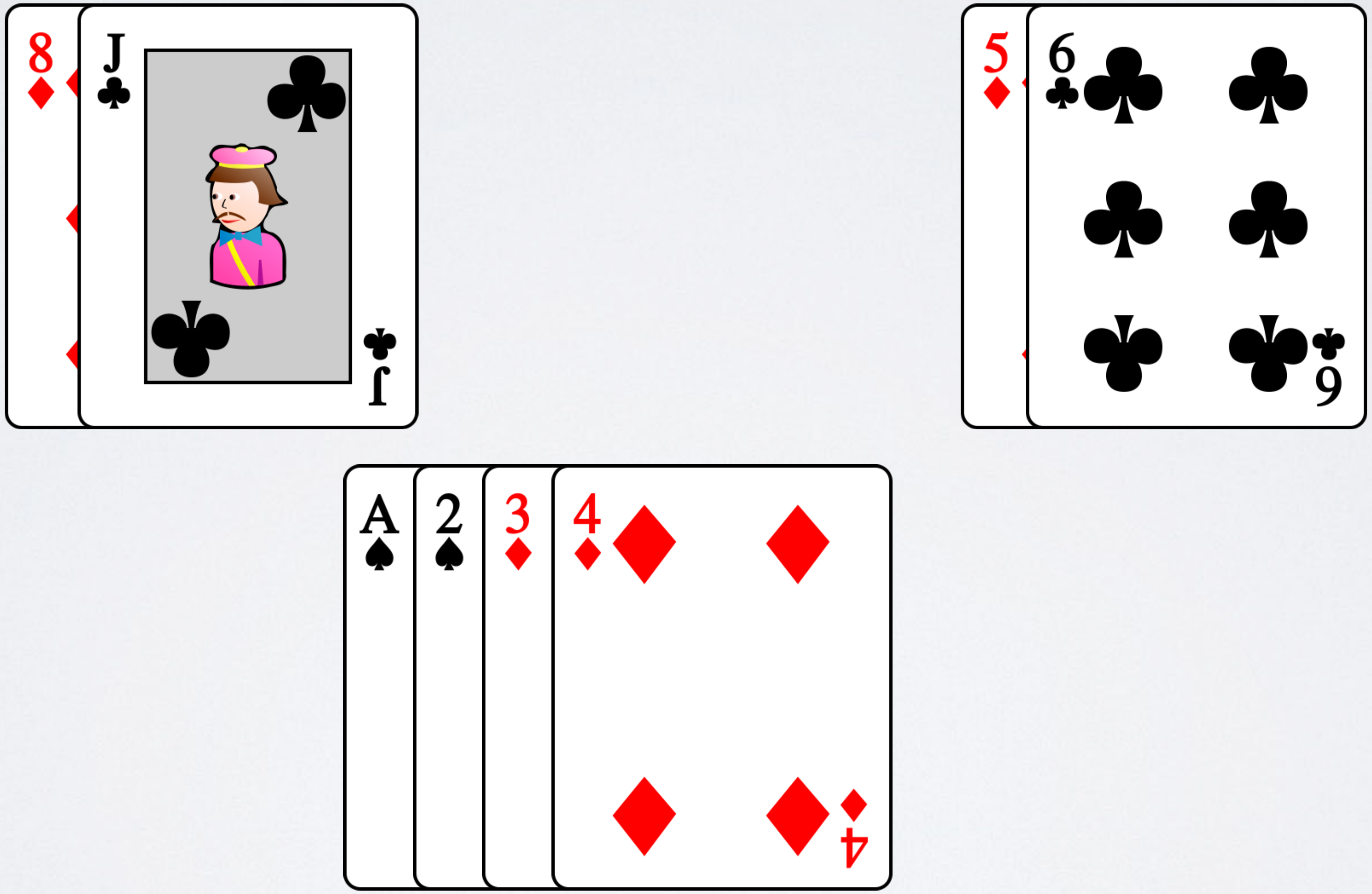
FUSIONNER



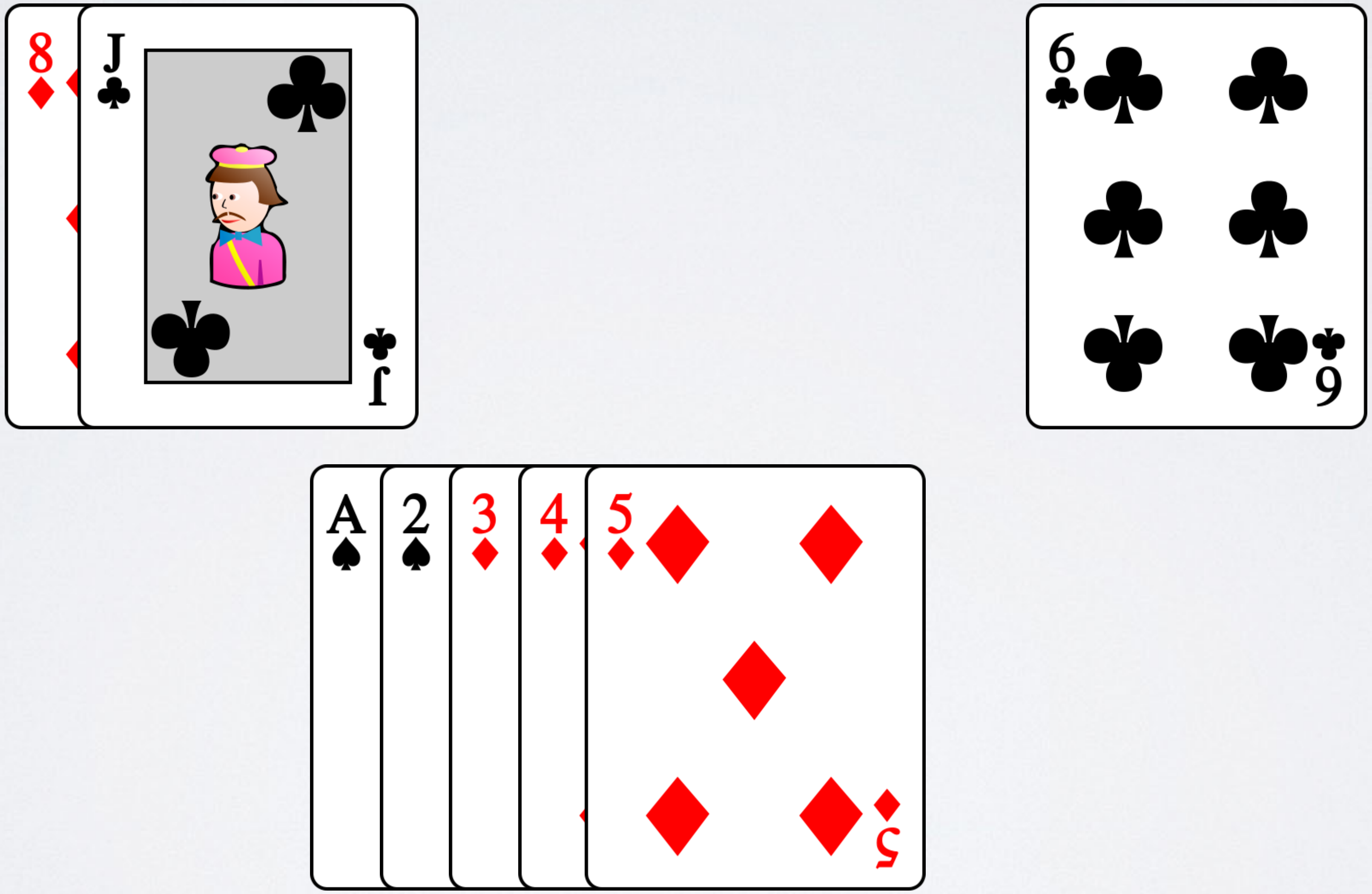
FUSIONNER



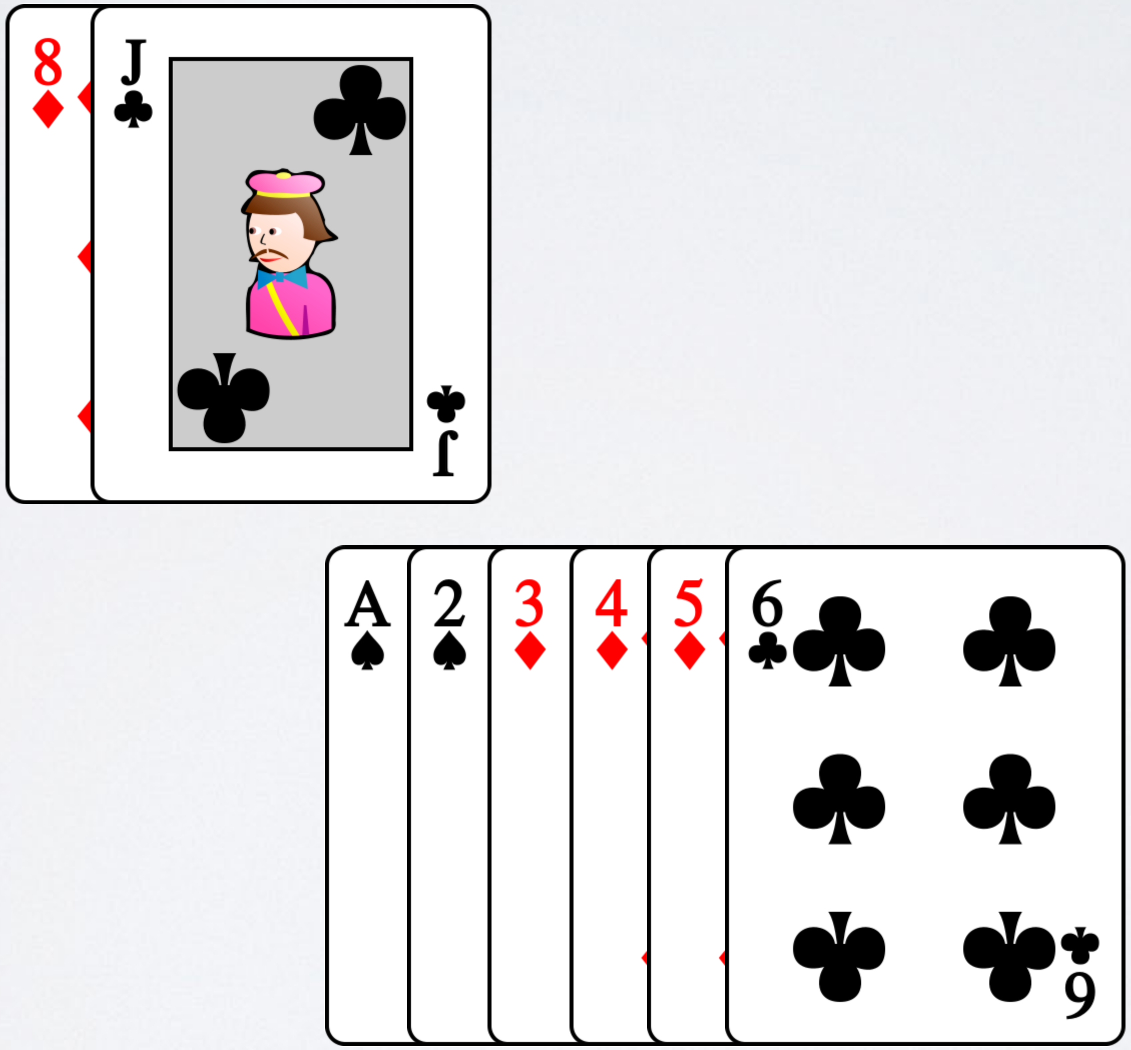
FUSIONNER



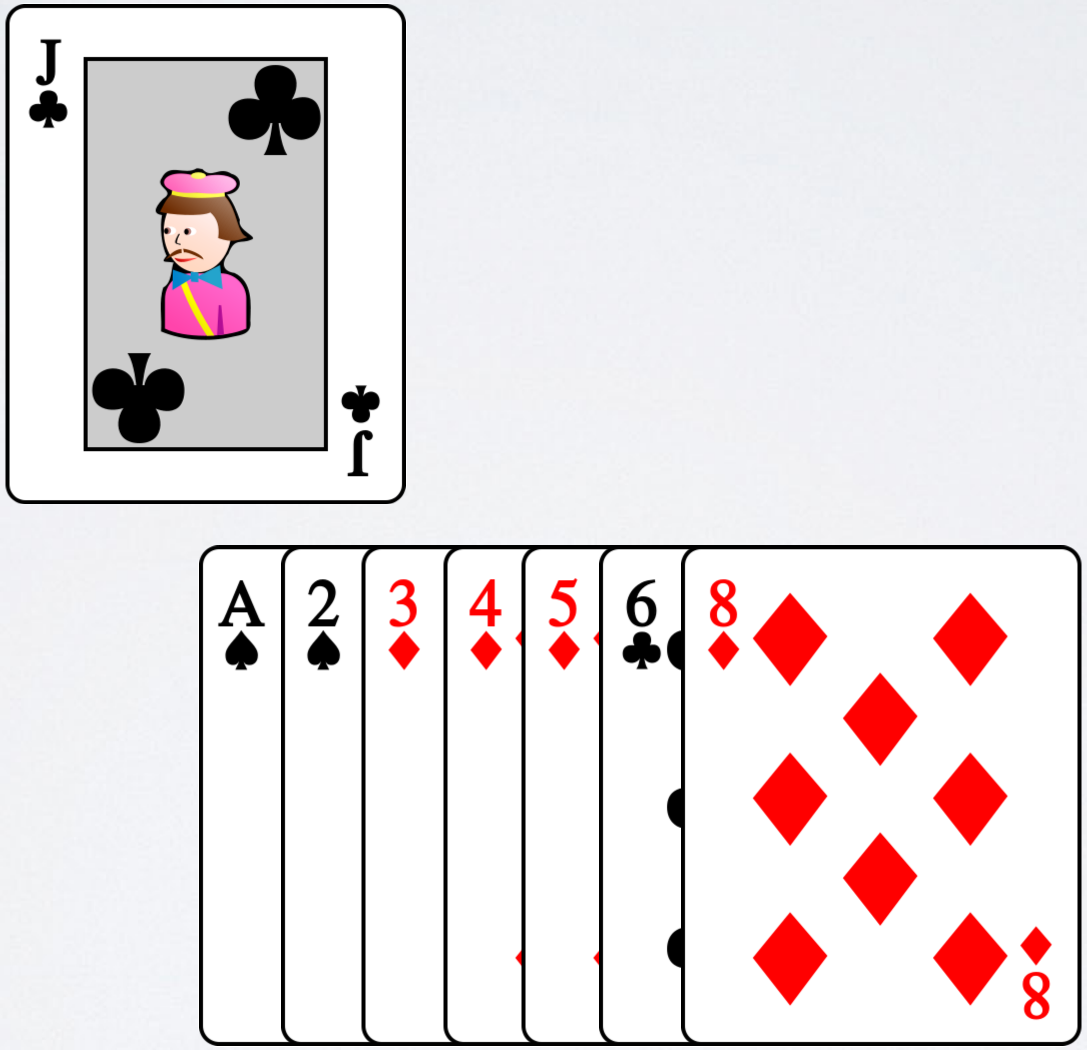
FUSIONNER



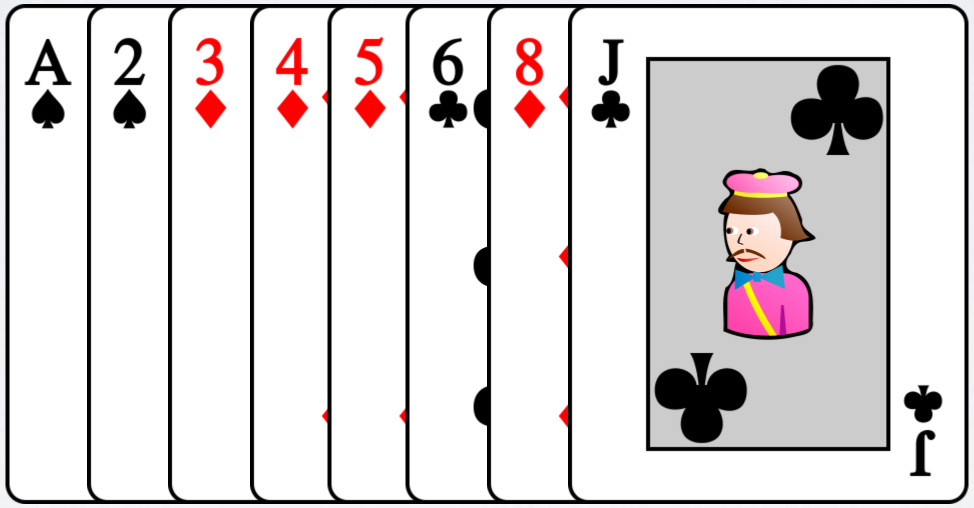
FUSIONNER



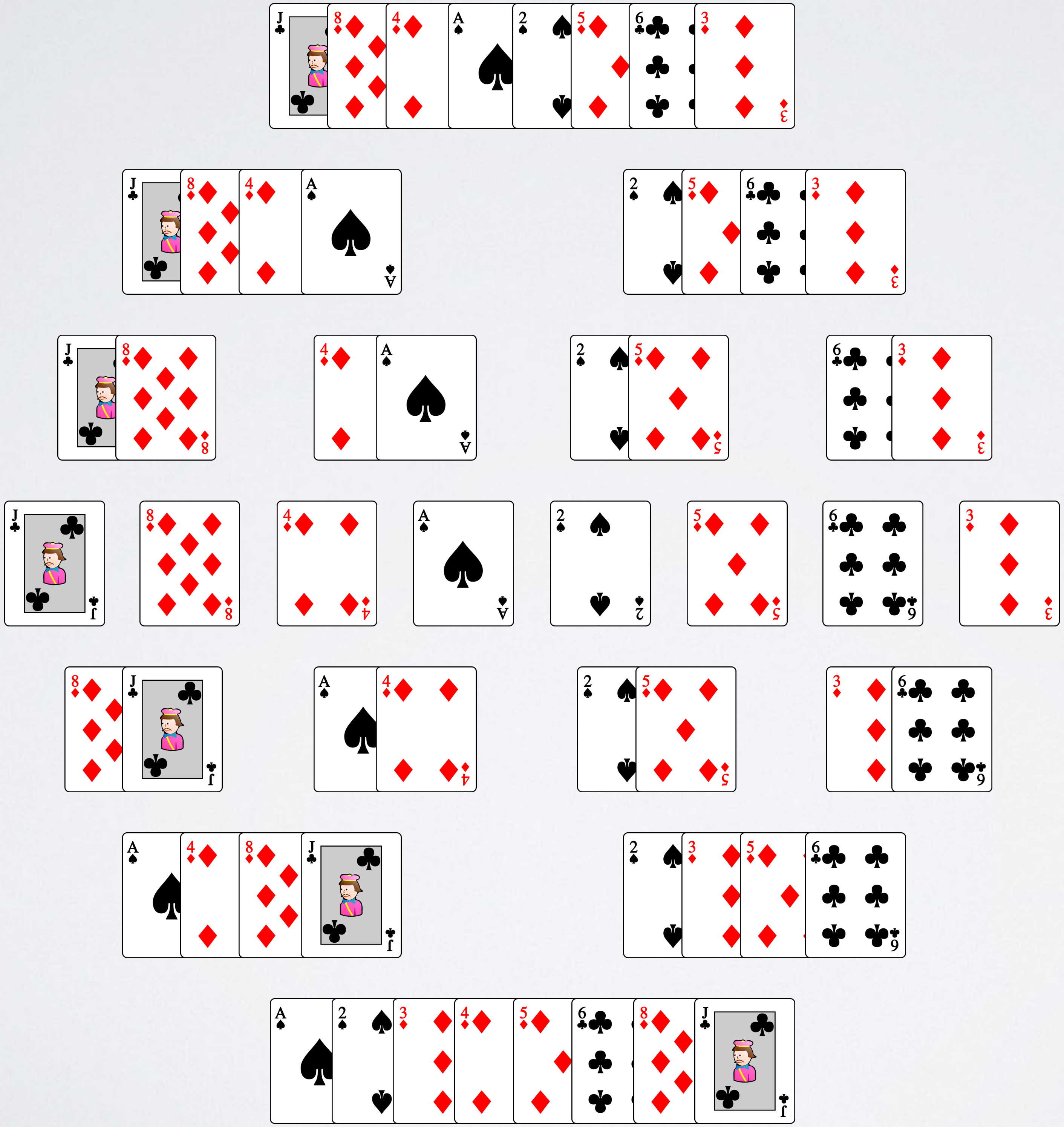
FUSIONNER



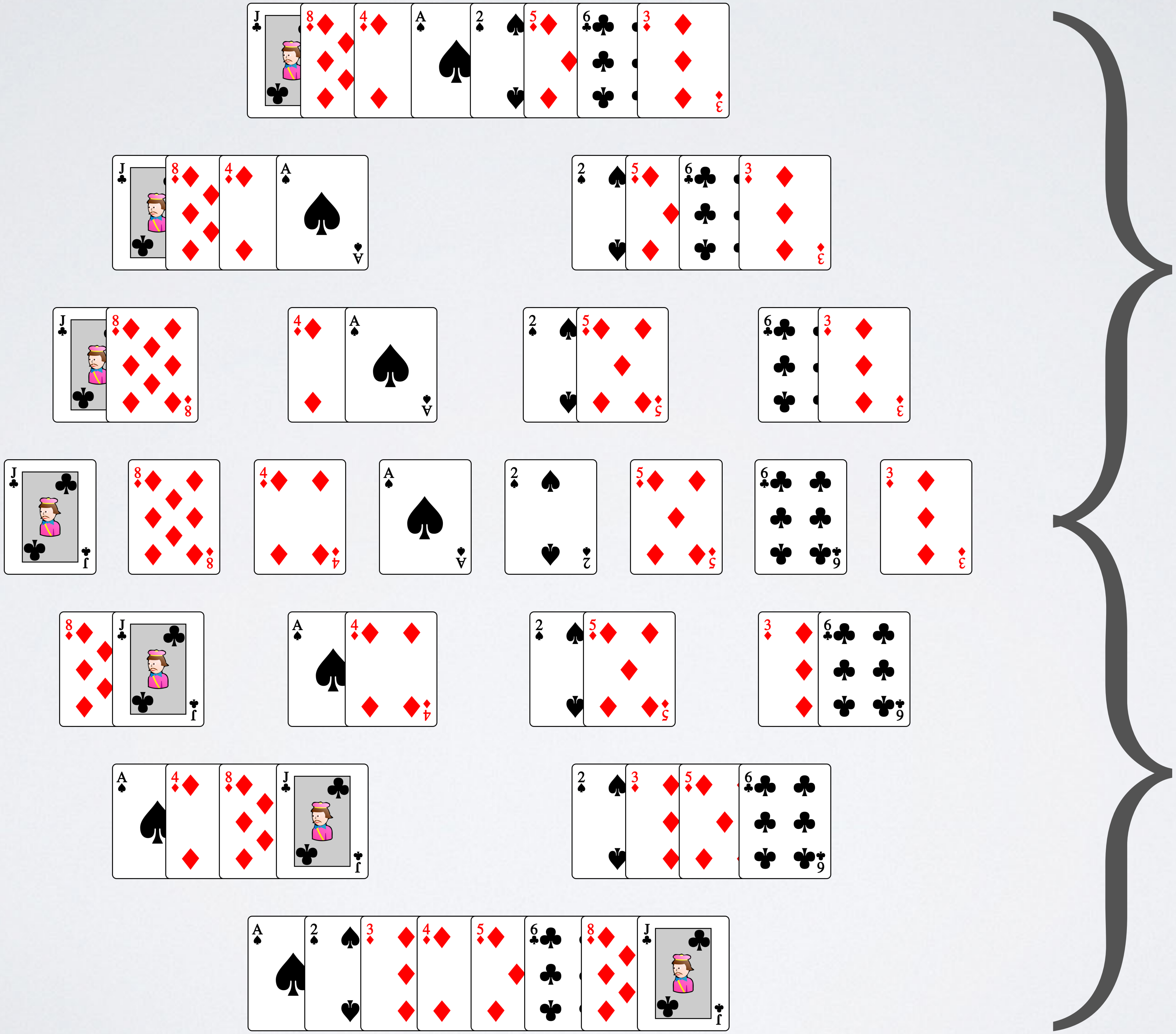
LE JEU EST TRIÉ !



EFFICACITÉ DU TRI FUSION



EFFICACITÉ DU TRI FUSION



n

n

n

n

n

n

n

$\log_2 n$

$\log_2 n$

LA COMPLEXITÉ DU TRI FUSION EST

$$O(n \log_2 n)$$