Activité débranchée : Comprendre les algorithmes de tri avec des cartes

Objectifs pédagogiques

- Comprendre le fonctionnement de différents algorithmes de tri.
- Relier les manipulations concrètes à leur représentation en pseudo-code et en langage informatique.
- Calculer et analyser la complexité des algorithmes de tri.
- Comparer les algorithmes et en tirer une synthèse.

Matériel nécessaire

- Un jeu de cartes mélangé (par groupe).
- Une fiche de consignes détaillant chaque algorithme.
- Une grille de synthèse (voir plus bas).
- Un chronomètre (par groupe).

Instructions par algorithme

1. Tri par sélection (*Selection Sort*)

Pseudo-code :

```
POUR i DE 0 À n-1 FAIRE
    min_index ← i
    POUR j DE i+1 À n-1 FAIRE
        SI array[j] < array[min_index] ALORS
            min_index ← j
        FIN SI
    FIN POUR
    ÉCHANGER array[i] ET array[min_index]
FIN POUR
```

Code en C :

```c
#include <stdio.h>

void selection_sort(int array[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int min_index = i;
        for (int j = i + 1; j < n; j++) {
            if (array[j] < array[min_index]) {
```

```
                min_index = j; // Trouve le minimum
            }
        }
        // Echange
        int temp = array[i];
        array[i] = array[min_index];
        array[min_index] = temp;
    }
}
```

Code en OCaml :

```ocaml
let selection_sort arr =
  let n = Array.length arr in
  for i = 0 to n - 2 do
    let min_index = ref i in
    for j = i + 1 to n - 1 do
      if arr.(j) < arr.(!min_index) then
        min_index := j
    done;
    let temp = arr.(i) in
    arr.(i) <- arr.(!min_index);
    arr.(!min_index) <- temp
  done
```

Analyse de la complexité :

— **Comparaisons :** $n(n-1)/2$, car pour chaque élément, on parcourt les éléments restants.
— **Échanges :** Au maximum $n-1$ échanges, car on effectue un échange par itération extérieure.
— **Complexité temporelle :** $O(n^2)$, car les comparaisons dominent.
— **Complexité spatiale :** $O(1)$, car nous utilisons une mémoire constante.

2. Tri par insertion (*Insertion Sort*)

Pseudo-code :

```
POUR i DE 1 À n-1 FAIRE
    current ← array[i]
    j ← i - 1
    TANT QUE j  0 ET array[j] > current FAIRE
        array[j+1] ← array[j]
        j ← j - 1
    FIN TANT QUE
    array[j+1] ← current
FIN POUR
```

Code en C :

```c
#include <stdio.h>

void insertion_sort(int array[], int n) {
    for (int i = 1; i < n; i++) {
        int current = array[i];
        int j = i - 1;
        while (j >= 0 && array[j] > current) {
```

```
            array[j + 1] = array[j];
            j--;
        }
        array[j + 1] = current;
    }
}
```

Code en OCaml :

```
let insertion_sort arr =
  let n = Array.length arr in
  for i = 1 to n - 1 do
    let current = arr.(i) in
    let j = ref (i - 1) in
    while !j >= 0 && arr.(!j) > current do
      arr.(!j + 1) <- arr.(!j);
      decr j
    done;
    arr.(!j + 1) <- current
  done
```

Analyse de la complexité :

- **Comparaisons :** Dans le pire cas (tableau inversé), environ $n(n-1)/2$.
- **Échanges :** Identique au nombre de comparaisons dans le pire cas.
- **Complexité temporelle :** $O(n^2)$ dans le pire cas, mais $O(n)$ dans le meilleur cas (tableau déjà trié).
- **Complexité spatiale :** $O(1)$.

3. Tri à bulles (*Bubble Sort*)

Pseudo-code :

```
POUR i DE 0 À n-1 FAIRE
    POUR j DE 0 À n-i-2 FAIRE
        SI array[j] > array[j+1] ALORS
            ÉCHANGER array[j] ET array[j+1]
        FIN SI
    FIN POUR
FIN POUR
```

Code en C :

```
#include <stdio.h>

void bubble_sort(int array[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (array[j] > array[j + 1]) {
                int temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }
}
```

Code en OCaml :

```
let bubble_sort arr =
  let n = Array.length arr in
  for i = 0 to n - 2 do
    for j = 0 to n - i - 2 do
      if arr.(j) > arr.(j + 1) then
        let temp = arr.(j) in
        arr.(j) <- arr.(j + 1);
        arr.(j + 1) <- temp
    done
  done
```

Analyse de la complexité :

— **Comparaisons :** $n(n-1)/2$, car chaque paire est comparée.
— **Échanges :** Identique au nombre de comparaisons dans le pire cas.
— **Complexité temporelle :** $O(n^2)$ dans le pire cas.
— **Complexité spatiale :** $O(1)$.

Signature : E.SABAHI MP2I Informatique