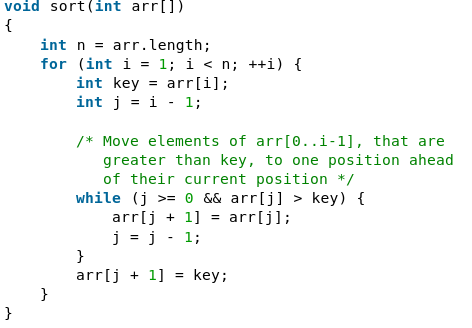
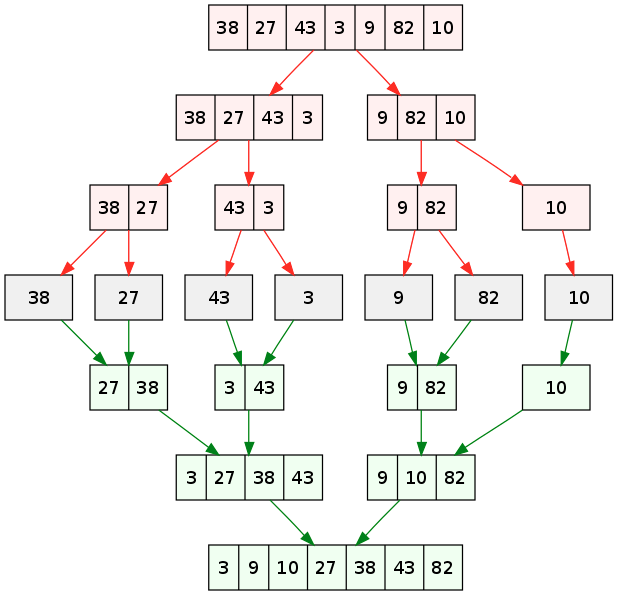
**Algorithmes de Tri**

1. **Insertion sorting**





1. **Merge Sort (moyen/difficile) - efficace - peu chronophage mais un peu spatiovore.**



// example of merge sort in Java

// merge function take two intervals

// one from start to mid

// second from mid+1, to end

// and merge them in sorted order

void merge(int Arr[], int start, int mid, int end) {

// create a temp array

int temp[] = new int[end - start + 1];

// crawlers for both intervals and for temp

int i = start, j = mid+1, k = 0;

// traverse both arrays and in each iteration add smaller of both elements in temp

while(i <= mid && j <= end) {

if(Arr[i] <= Arr[j]) {

temp[k] = Arr[i];

k += 1; i += 1;

}

else {

temp[k] = Arr[j];

k += 1; j += 1;

}

}

// add elements left in the first interval

while(i <= mid) {

temp[k] = Arr[i];

k += 1; i += 1;

}

// add elements left in the second interval

while(j <= end) {

temp[k] = Arr[j];

k += 1; j += 1;

}

// copy temp to original interval

for(i = start; i <= end; i += 1) {

Arr[i] = temp[i - start]

}

}

// Arr is an array of integer type

// start and end are the starting and ending index of current interval of Arr

void mergeSort(int Arr[], int start, int end) {

if(start < end) {

int mid = (start + end) / 2;

mergeSort(Arr, start, mid);

mergeSort(Arr, mid+1, end);

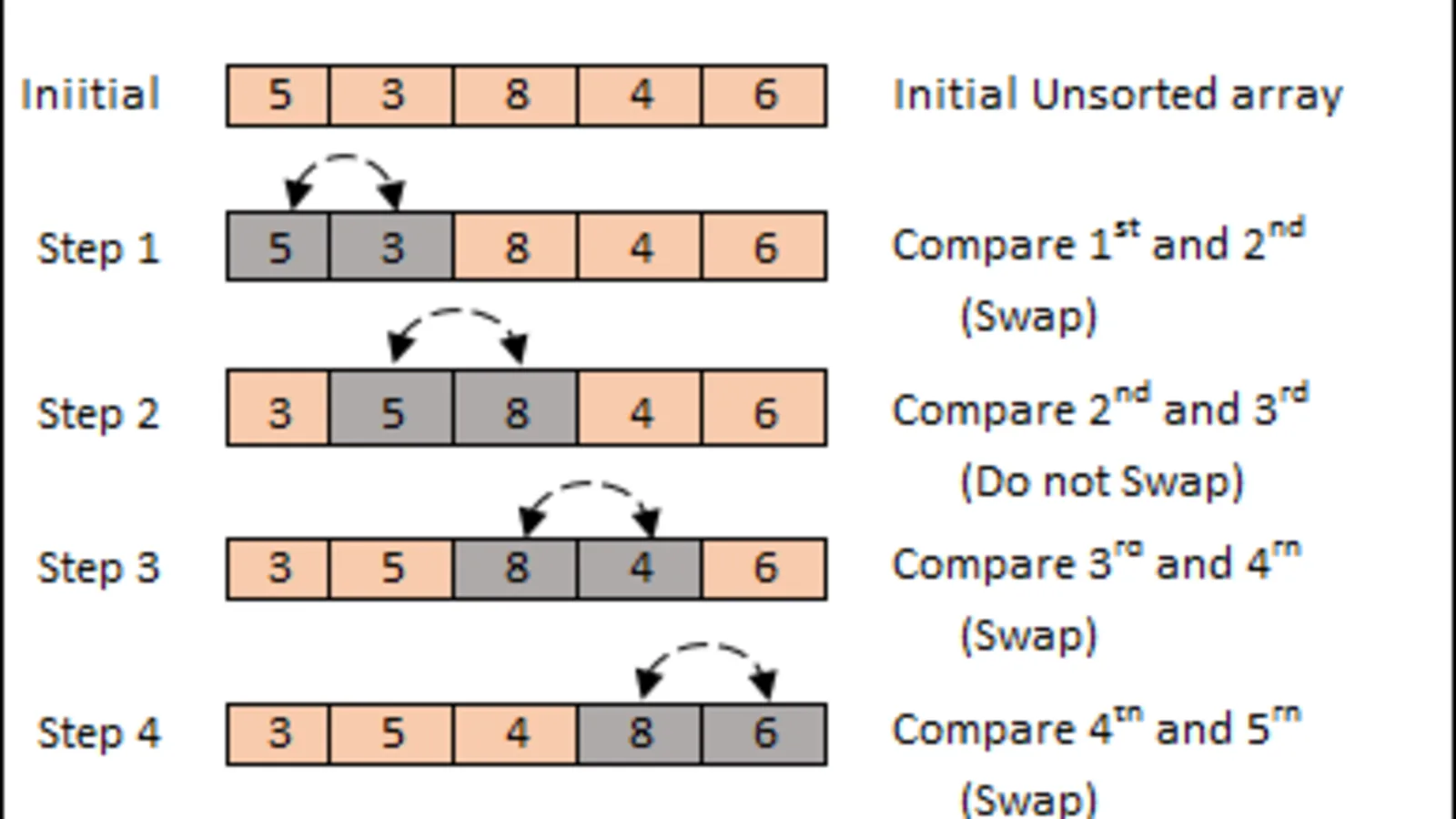
merge(Arr, start, mid, end);

}

}

Voir variante : TimSort.

1. **Bubble Sort ( très facile) - peu éfficace**

****

// Java program for implementation of Bubble Sort

class BubbleSort

{

void bubbleSort(int arr[])

{

int n = arr.length;

for (int i = 0; i < n-1; i++)

for (int j = 0; j < n-i-1; j++)

if (arr[j] > arr[j+1])

{

// swap arr[j+1] and arr[j]

int temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

/\* Prints the array \*/

void printArray(int arr[])

{

int n = arr.length;

for (int i=0; i<n; ++i)

System.out.print(arr[i] + " ");

System.out.println();

}

// Driver method to test above

public static void main(String args[])

{

BubbleSort ob = new BubbleSort();

int arr[] = {64, 34, 25, 12, 22, 11, 90};

ob.bubbleSort(arr);

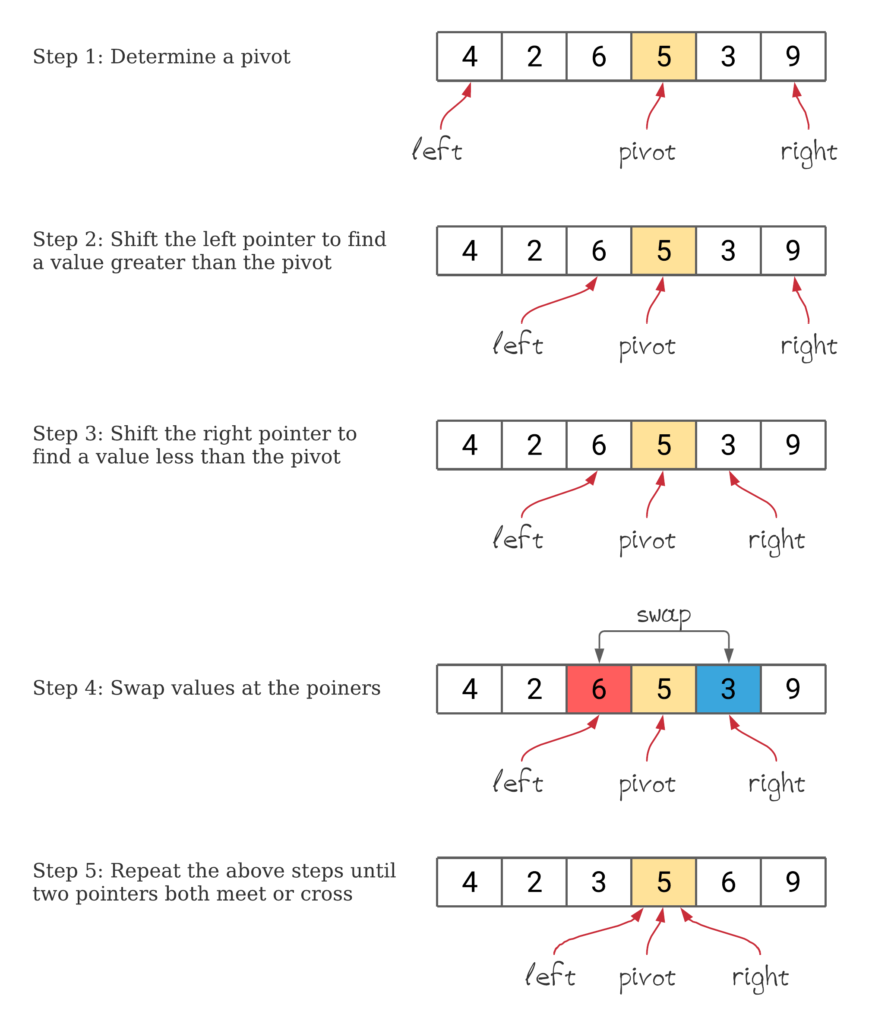
System.out.println("Sorted array");

ob.printArray(arr);

}

}

1. **Quick Sort**

****

// Java implementation of QuickSort

import java.io.\*;

class GFG{

// A utility function to swap two elements

static void swap(int[] arr, int i, int j)

{

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

/\* This function takes last element as pivot, places

the pivot element at its correct position in sorted

array, and places all smaller (smaller than pivot)

to left of pivot and all greater elements to right

of pivot \*/

static int partition(int[] arr, int low, int high)

{

// pivot

int pivot = arr[high];

// Index of smaller element and

// indicates the right position

// of pivot found so far

int i = (low - 1);

for(int j = low; j <= high - 1; j++)

{

// If current element is smaller

// than the pivot

if (arr[j] < pivot)

{

// Increment index of

// smaller element

i++;

swap(arr, i, j);

}

}

swap(arr, i + 1, high);

return (i + 1);

}

/\* The main function that implements QuickSort

arr[] --> Array to be sorted,

low --> Starting index,

high --> Ending index

\*/

static void quickSort(int[] arr, int low, int high)

{

if (low < high)

{

// pi is partitioning index, arr[p]

// is now at right place

int pi = partition(arr, low, high);

// Separately sort elements before

// partition and after partition

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

// Function to print an array

static void printArray(int[] arr, int size)

{

for(int i = 0; i < size; i++)

System.out.print(arr[i] + " ");

System.out.println();

}

// Driver Code

public static void main(String[] args)

{

int[] arr = { 10, 7, 8, 9, 1, 5 };

int n = arr.length;

quickSort(arr, 0, n - 1);

System.out.println("Sorted array: ");

printArray(arr, n);

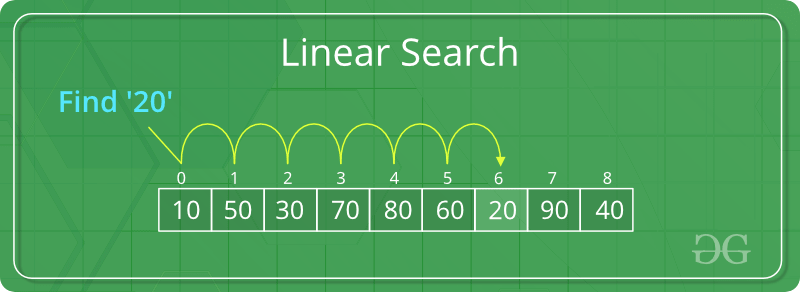
}

}

// This code is contributed by Ayush Choudhary

**Algorithme de Recherche :**

1. **Recherche linéaire (facile)**



// Java code for linearly searching x in arr[]. If x

// is present then return its location, otherwise

// return -1

class GFG

{

public static int search(int arr[], int x)

{

int n = arr.length;

for (int i = 0; i < n; i++)

{

if (arr[i] == x)

return i;

}

return -1;

}

// Driver code

public static void main(String args[])

{

int arr[] = { 2, 3, 4, 10, 40 };

int x = 10;

// Function call

int result = search(arr, x);

if (result == -1)

System.out.print(

"Element is not present in array");

else

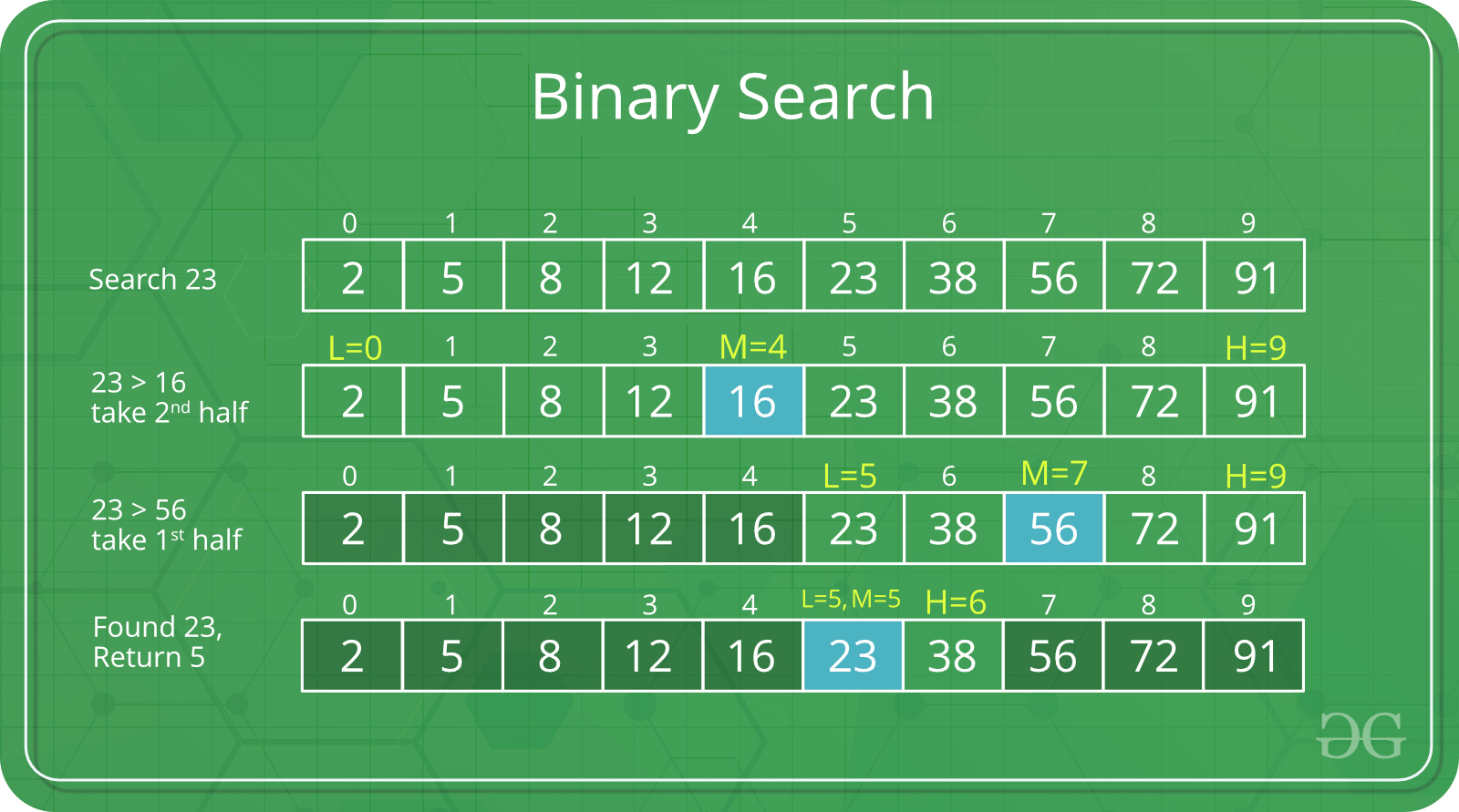
System.out.print("Element is present at index "

+ result);

}

}

1. **Binaire (facile mais impose une liste ordonnée)**

****

// Java implementation of recursive Binary Search

class BinarySearch {

// Returns index of x if it is present in arr[l..

// r], else return -1

int binarySearch(int arr[], int l, int r, int x)

{

if (r >= l) {

int mid = l + (r - l) / 2;

// If the element is present at the

// middle itself

if (arr[mid] == x)

return mid;

// If element is smaller than mid, then

// it can only be present in left subarray

if (arr[mid] > x)

return binarySearch(arr, l, mid - 1, x);

// Else the element can only be present

// in right subarray

return binarySearch(arr, mid + 1, r, x);

}

// We reach here when element is not present

// in array

return -1;

}

// Driver method to test above

public static void main(String args[])

{

BinarySearch ob = new BinarySearch();

int arr[] = { 2, 3, 4, 10, 40 };

int n = arr.length;

int x = 10;

int result = ob.binarySearch(arr, 0, n - 1, x);

if (result == -1)

System.out.println("Element not present");

else

System.out.println("Element found at index " + result);

}

}

/\* This code is contributed by Rajat Mishra \*/