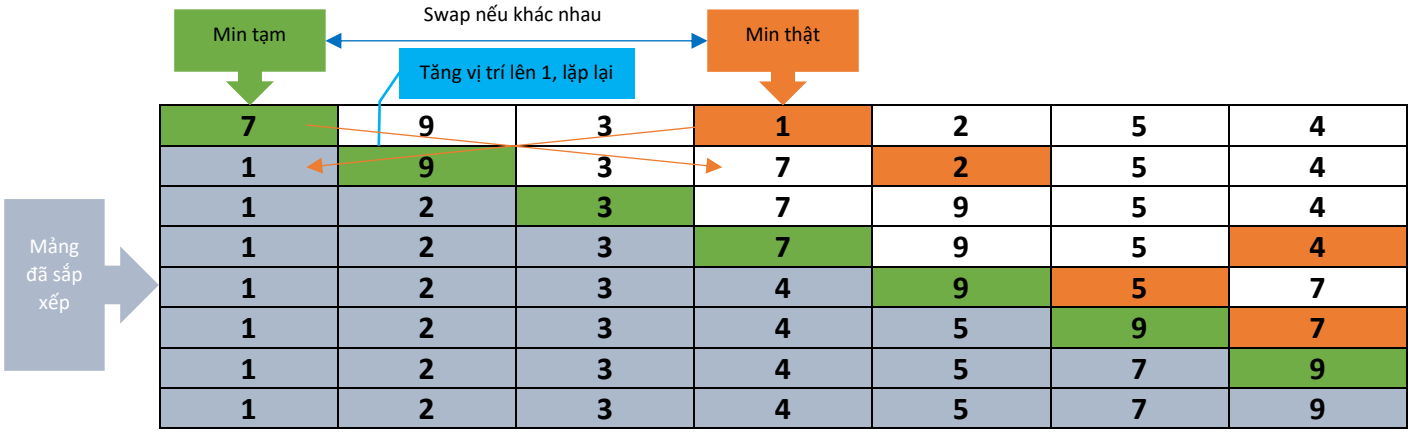
BÁO CÁO

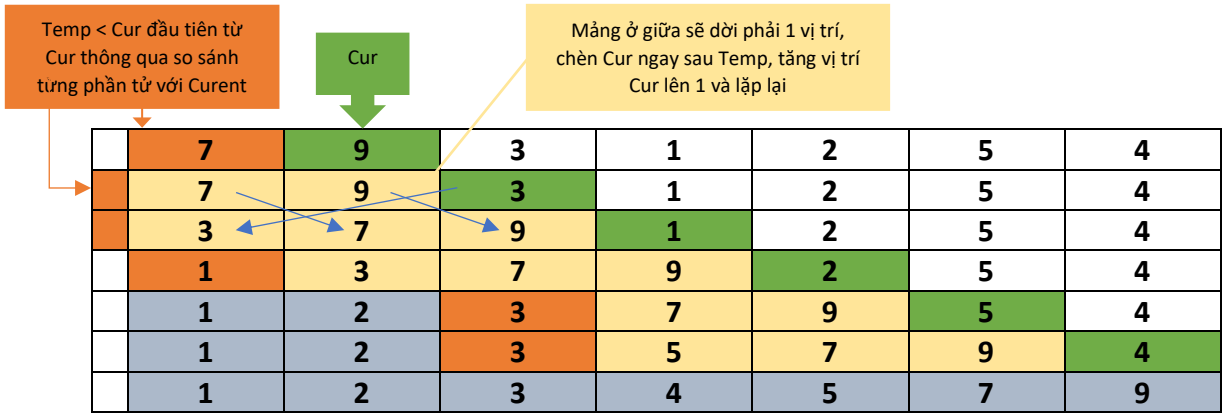
Do một số thuật toán khá dài, nếu trình bày rõ thì sẽ dài dòng và khó hiểu, do vậy em chỉ trình bày sơ bộ ý tưởng và ví dụ minh họa, còn chi tiết thì sẽ giải thích trong từng dòng code trong source.

1. **Selection Sort:** Quét đến hết mảng, tìm min rồi dồn về đầu



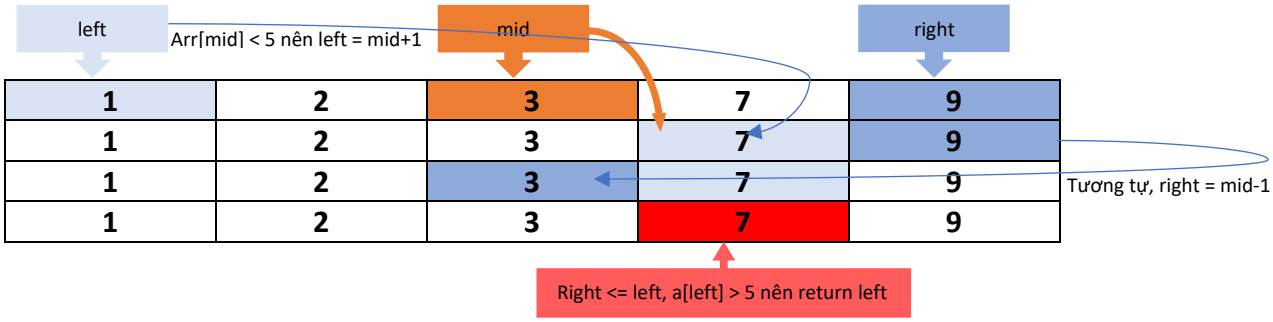
7	9	3	1	2	5	4
1	9	3	7	2	5	4
1	2	3	7	9	5	4
1	2	3	7	9	5	4
1	2	3	4	9	5	7
1	2	3	4	5	9	7
1	2	3	4	5	7	9
1	2	3	4	5	7	9

2. **Insertion Sort:** Quét từ phần tử thứ 2 đến hết, chèn ngược vào mảng trước nó theo thứ tự

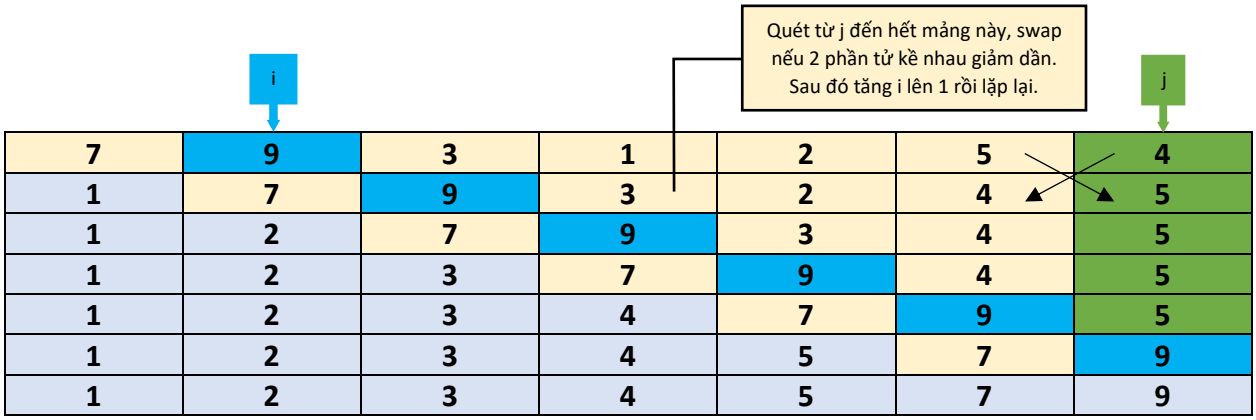


7	9	3	1	2	5	4
7	9	3	1	2	5	4
3	7	9	1	2	5	4
1	3	7	9	2	5	4
1	2	3	7	9	5	4
1	2	3	5	7	9	4
1	2	3	4	5	7	9

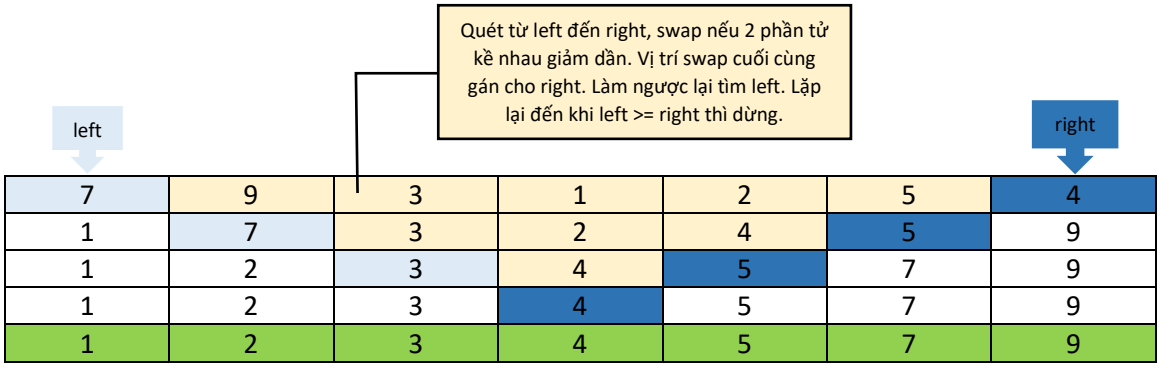
3. **Binary Insertion Sort:**
   - Ý tưởng tương tự Insertion Sort, tuy nhiên sẽ tìm trực tiếp vị trí ngay sau Temp thông qua Binary Search chứ không so sánh từng phần tử một.
   - Ví dụ Binary Search tìm vị trí ngay sau Temp < (Cur = 5) đầu tiên tính từ Cur trong mảng:



4. Bubble Sort:

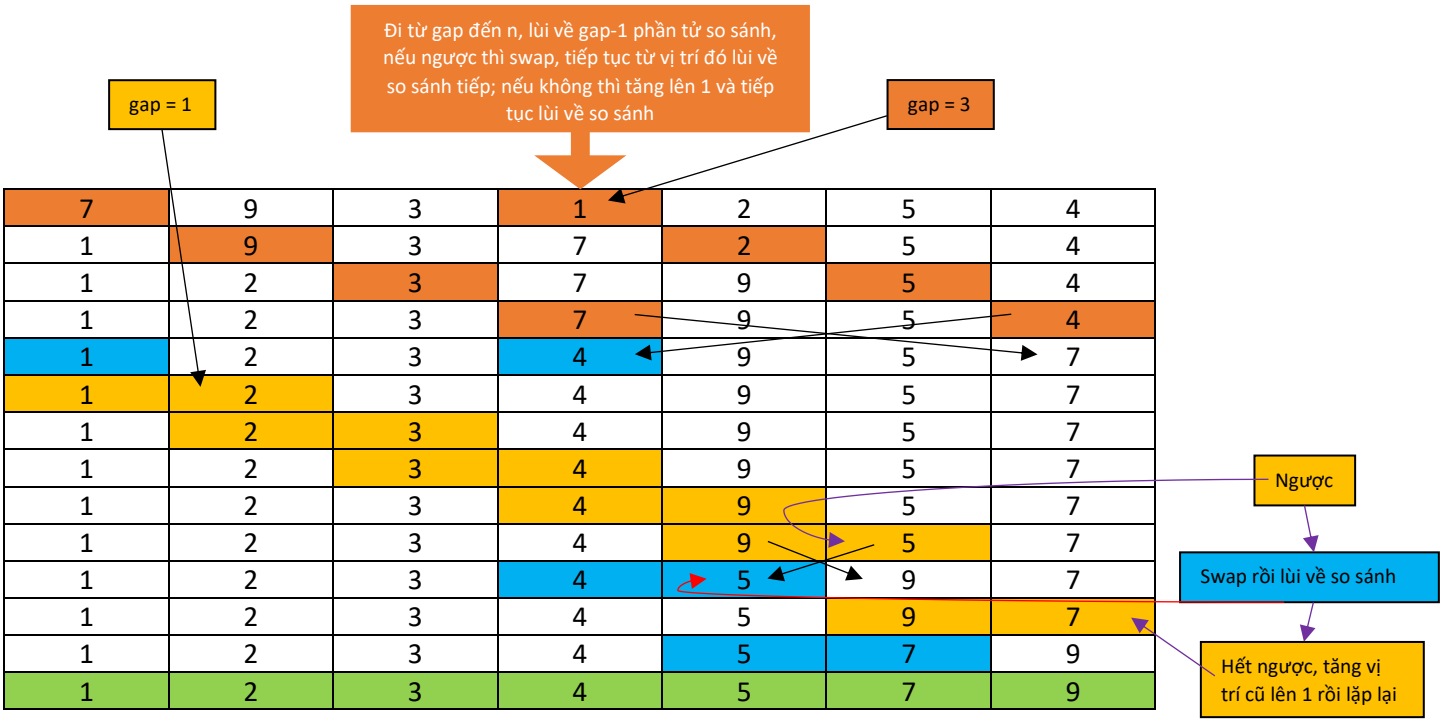


5. Shaker Sort: Ý tưởng tương tự Bubble Sort nhưng sẽ dùng 2 chốt ở 2 đầu để tối ưu tốc độ



6. Shell Sort:

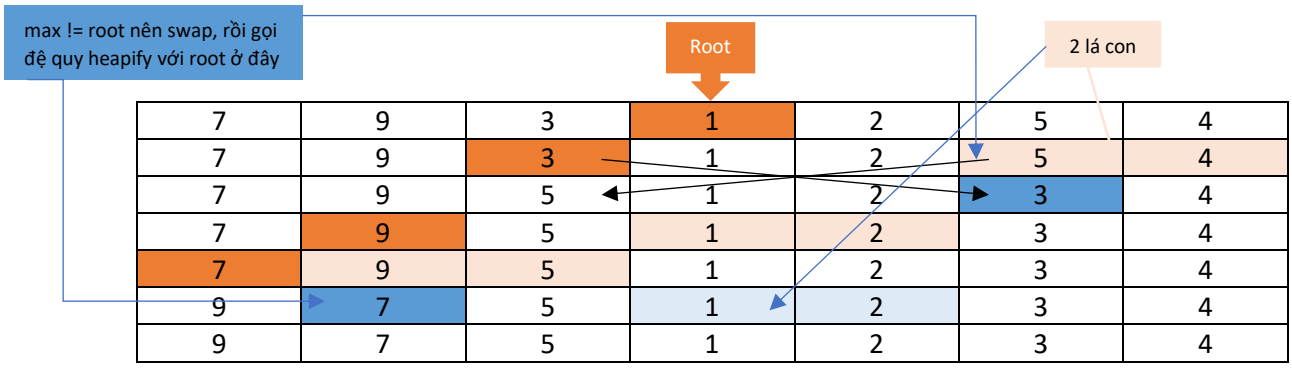
- Sắp xếp theo từng cặp cách nhau gap-1 phần tử theo Insertion Sort, sau đó giảm gap phân nửa và tiếp tục sắp xếp đến khi gap = 0 thì dừng thuật toán.
- N = 7 => gap = 7 / 2 = 3 => gap / 2 = 1 => gap / 2 = 0.



7	9	3	1	2	5	4
1	9	3	7	2	5	4
1	2	3	7	9	5	4
1	2	3	7	9	5	4
1	2	3	4	9	5	7
1	2	3	4	9	5	7
1	2	3	4	9	5	7
1	2	3	4	9	5	7
1	2	3	4	9	5	7
1	2	3	4	9	5	7
1	2	3	4	5	9	7
1	2	3	4	5	9	7
1	2	3	4	5	7	9
1	2	3	4	5	7	9

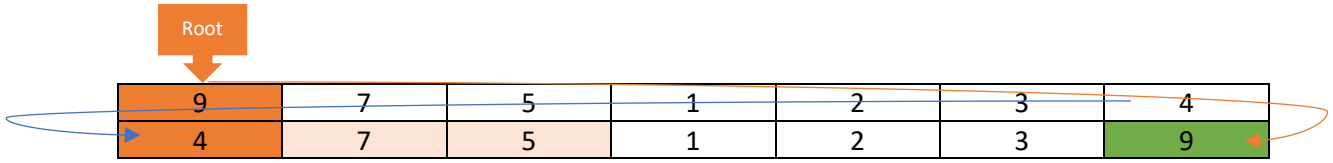
7. Heap Sort:

- Xây dựng Heap:
  - N = 7 => (N-1)/2 =3
  - Đi từ phần tử 3 đến đầu để xây dựng Max Heap.



7	9	3	1	2	5	4
7	9	3	1	2	5	4
7	9	5	1	2	3	4
7	9	5	1	2	3	4
7	9	5	1	2	3	4
9	7	5	1	2	3	4
9	7	5	1	2	3	4

- Tách từng lá lớn nhất ra khỏi Heap bằng cách swap max (phần tử đầu) với phần tử cuối, sau đó gọi heapify với root = 0, lặp lại đến khi lá cuối cùng được tách ra:

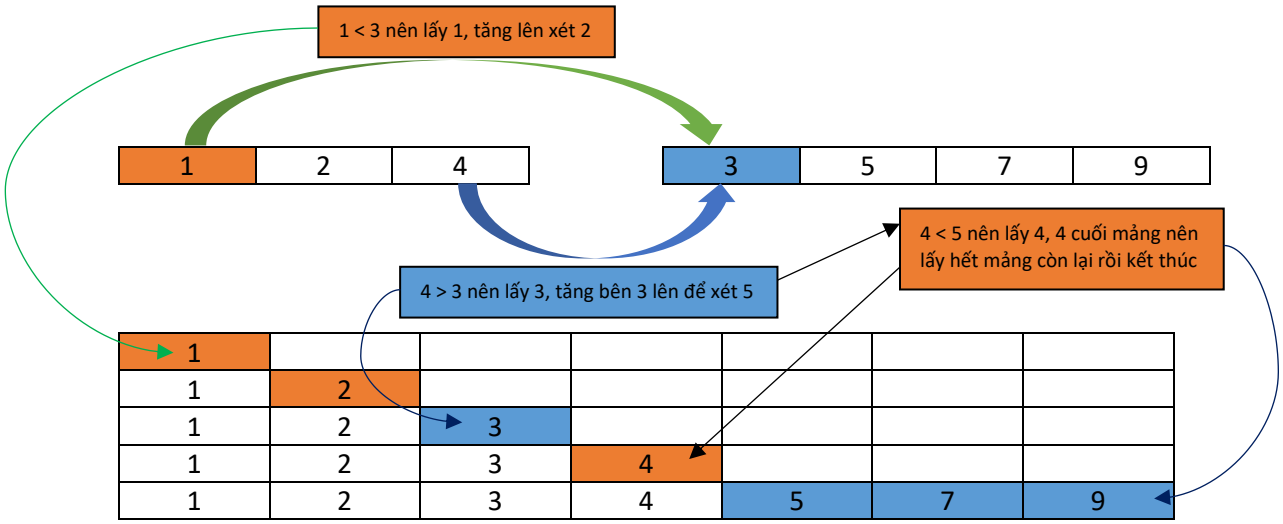


9	7	5	1	2	3	4
4	7	5	1	2	3	9

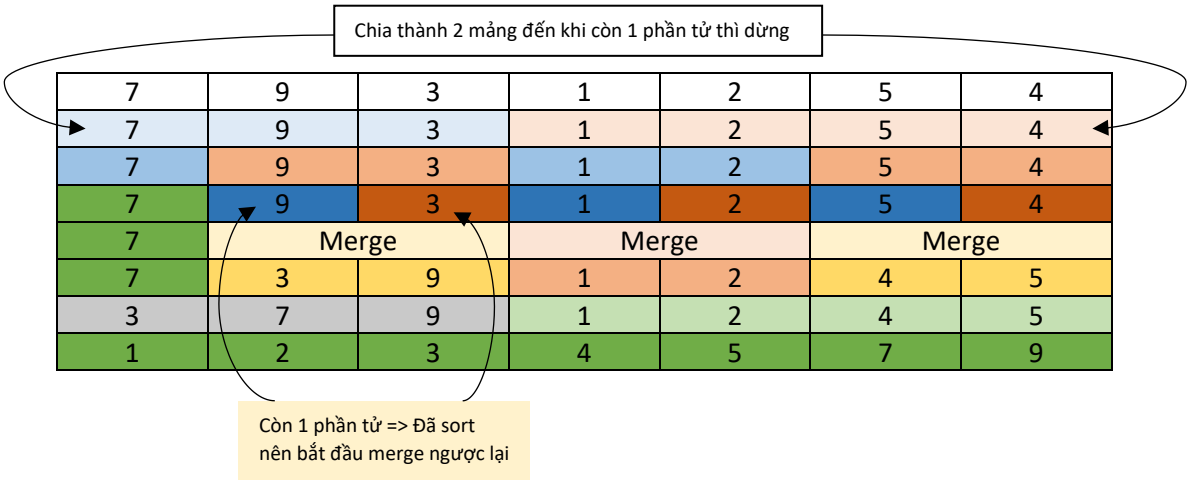
7	4	5	1	2	3	9
3	4	5	1	2	7	9
5	4	3	1	2	7	9
2	4	3	1	5	7	9
4	2	3	1	5	7	9
1	2	3	4	5	7	9
3	2	1	4	5	7	9
1	2	3	4	5	7	9
2	1	3	4	5	7	9
1	2	3	4	5	7	9
1	2	3	4	5	7	9

8. Merge Sort:

- Dựa trên ý tưởng merge 2 mảng đã sắp xếp, do vậy, đầu tiên sẽ ví dụ cho hàm MergeArray:

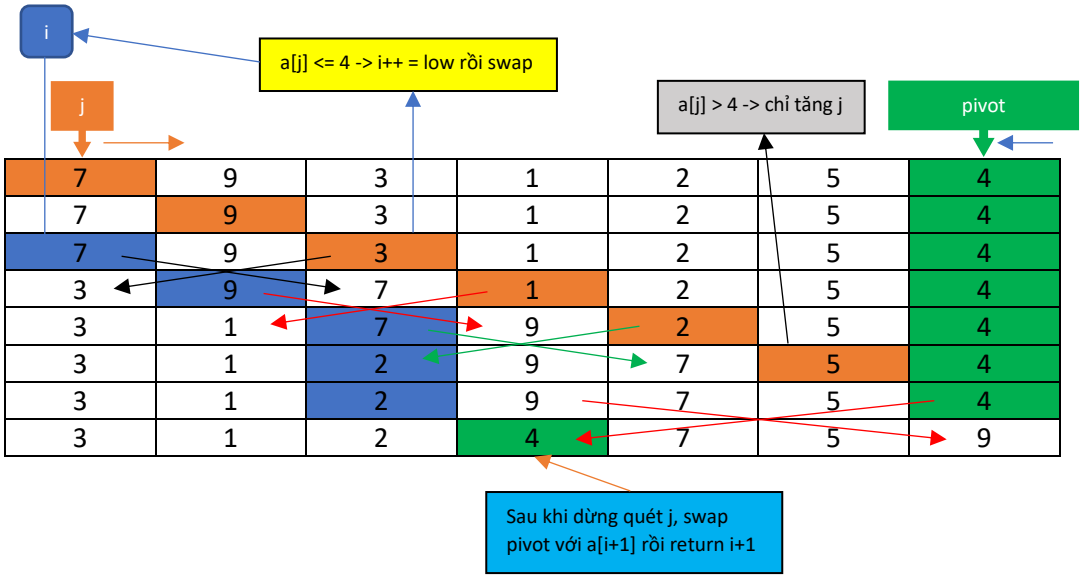


- Sau khi đã có hàm MergeArray, ta chia mảng ban đầu thành phân nữa, tiếp tục chia đến khi mảng con còn 1 phần tử thì dừng, rồi gộp ngược lại:



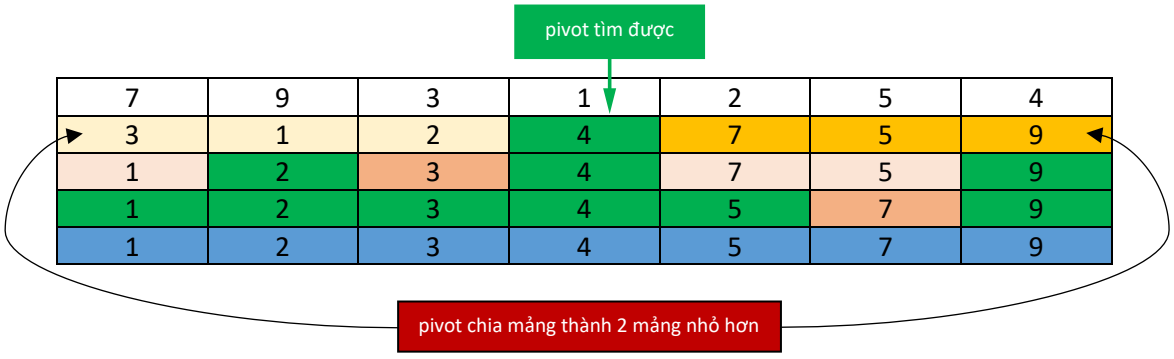
9. Quick Sort:

- Dựa trên ý tưởng tìm partition, do vậy, ta sẽ đi qua ví dụ về hàm Partition trước:
  - Đầu tiên, j = low, i = low – 1 và pivot bằng a[high] như hình.
  - Trong quá trình quét j từ low đến high-1, nếu a[j] <= pivot thì tăng i rồi swap a[j] và a[i].



7	9	3	1	2	5	4
7	9	3	1	2	5	4
7	9	3	1	2	5	4
3	9	7	1	2	5	4
3	1	7	9	2	5	4
3	1	2	9	7	5	4
3	1	2	9	7	5	4
3	1	2	4	7	5	9

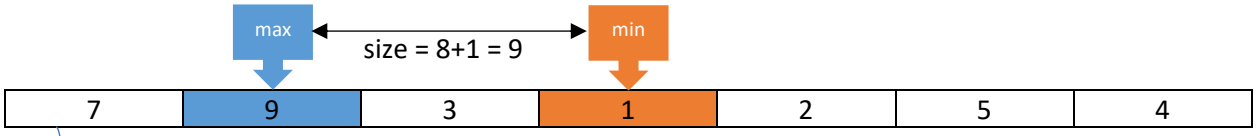
- Sau khi tìm được partition thì mọi chuyện trở nên dễ dàng hơn, vì nửa bên trái sẽ nhỏ hơn partition, còn nữa bên phải sẽ lớn hơn, do vậy khi mảng có số phần tử nhỏ hơn 2, sau khi tìm partition xong, mảng đó đã được sort.
- Trong hàm Quicksort, partition sẽ chia mảng làm 2 nửa, bên nào nhỏ hơn ta sẽ chạy bên đó, sau đó thay đổi left hoặc right để chạy bên mảng còn lại, lặp lại đến khi left >= right thì dừng:



7	9	3	1	2	5	4
3	1	2	4	7	5	9
1	2	3	4	7	5	9
1	2	3	4	5	7	9
1	2	3	4	5	7	9

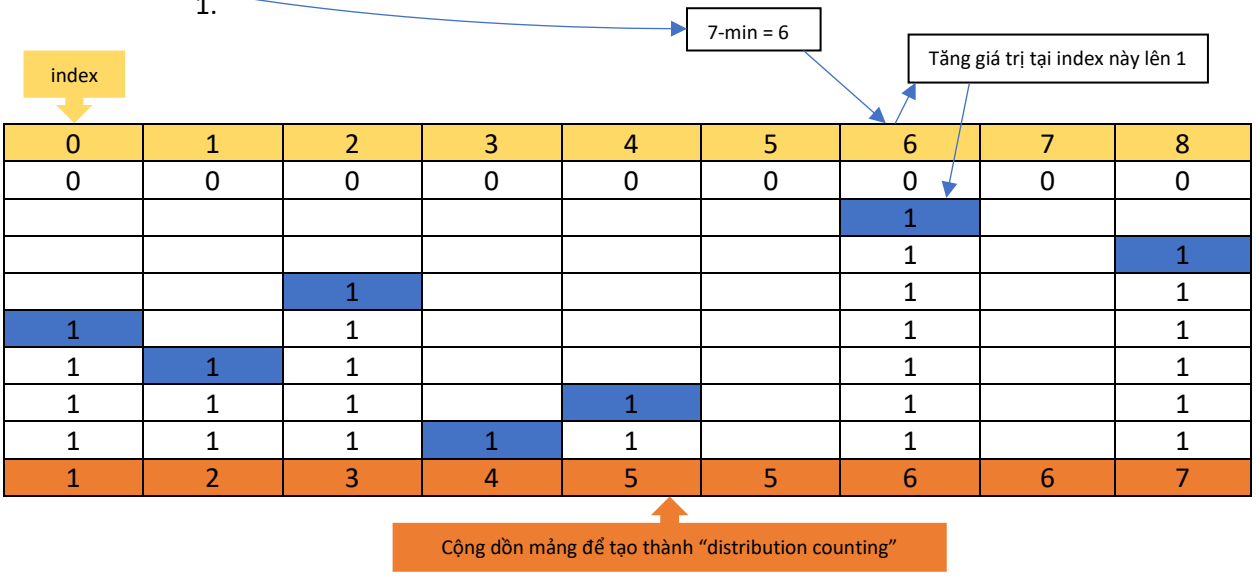
10. Counting Sort:

- Tìm min và max để xác định size của count_array. (size = max-min+1)



7	9	3	1	2	5	4

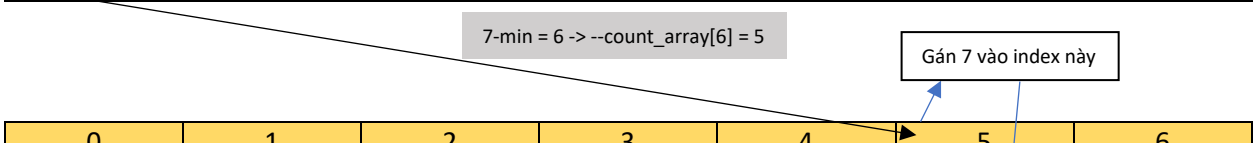
- Sau khi đã tìm được size, ta tạo mảng count_array với size đó và gán tất cả phần tử bằng 0.
- Quét hết mảng ban đầu, tăng giá trị phần tử của mảng count_array có index = a[i]-min lên 1.



0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0
						1		
						1		1
		1				1		1
1		1				1		1
1	1	1				1		1
1	1	1		1		1		1
1	1	1	1	1		1		1
1	2	3	4	5	5	6	6	7

- Sau khi có "distribution counting", tạo một mảng tạm là output_array để chứa các phần tử đã sort bằng cách ngược lại.

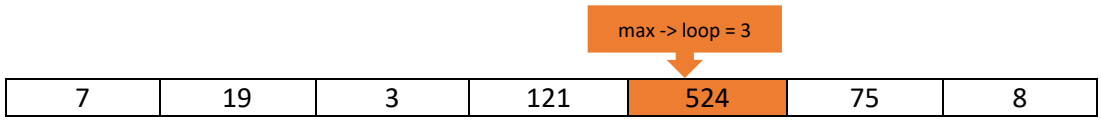
7	9	3	1	2	5	4



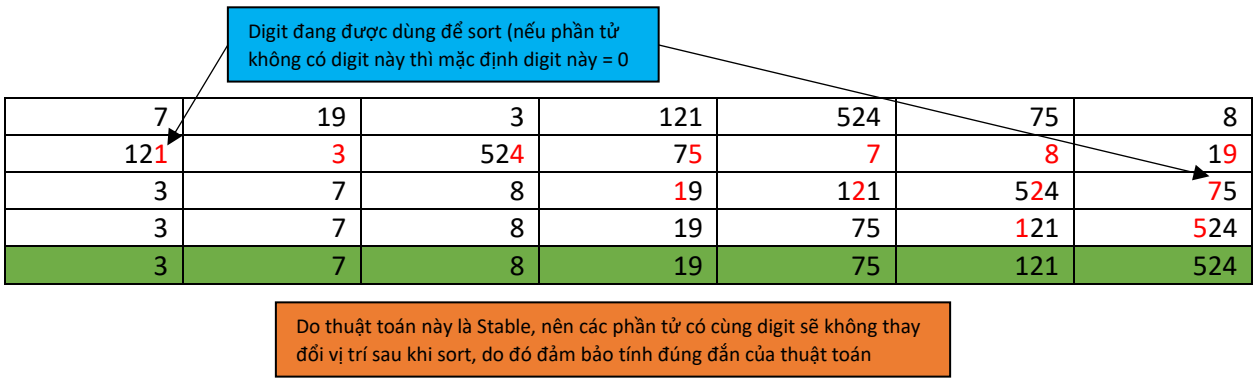
0	1	2	3	4	5	6
					7	
					7	9
		3			7	9
1		3			7	9
1	2	3			7	9
1	2	3		5	7	9
1	2	3	4	5	7	9
1	2	3	4	5	7	9

11. Counting Radix Sort:

- Đầu tiên, tìm loop (số digit lớn nhất trong mảng) thông qua max.



- Sau đó sắp xếp mảng theo từng digit bắt đầu từ LSD theo phương pháp Counting Sort đã trình bày ở trên:

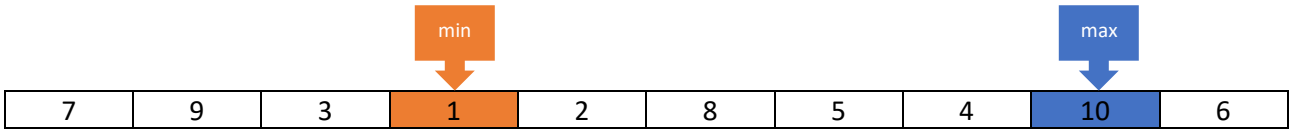


7	19	3	121	524	75	8
121	3	524	75	7	8	19
3	7	8	19	121	524	75
3	7	8	19	75	121	524
3	7	8	19	75	121	524

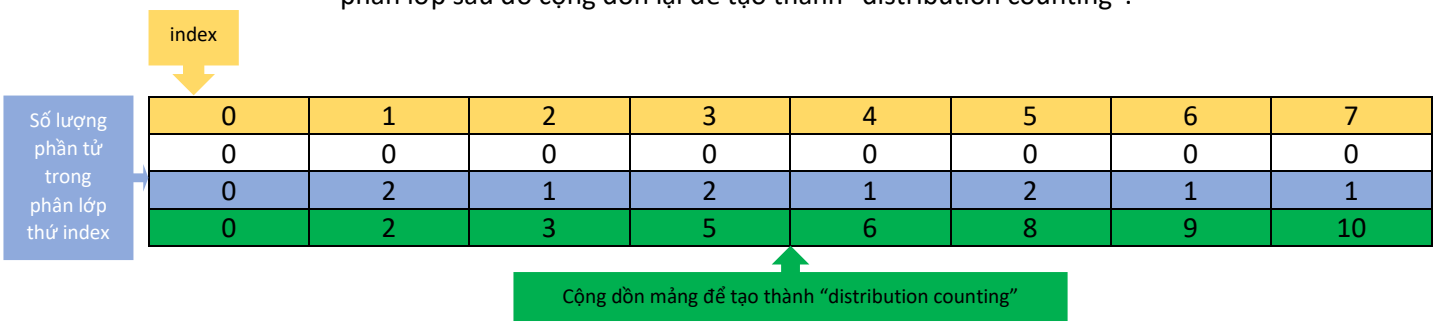
Do thuật toán này là Stable, nên các phần tử có cùng digit sẽ không thay đổi vị trí sau khi sort, do đó đảm bảo tính đúng đắn của thuật toán

12. Flash Sort:

- Đầu tiên tìm số phân lớp thông qua công thức: m = (int)((0.2 * n) + 2) = 7.
- Sau đó tìm min và max của mảng:



- Áp dụng công thức: K(A(i)) = 1 + INT((m-1)(A(i)-Amin)/(Amax-Amin)) để xác định xem phần tử thuộc phân lớp nào.
- Đặt: c = (m - 1.0) / (max - min) = 2/3. Suy ra: K(A(i)) = 1 + INT(2(A(i)-Amin)/3).
- Tương tự như Counting Sort, dùng mảng L size m+1 để chứa số lượng các số trong mỗi phân lớp sau đó cộng dồn lại để tạo thành "distribution counting".



- Kế tiếp, tiến hành swap a[max] và a[0]:

10	9	3	1	2	8	5	4	7	6

- Sau đó, tiến hành sắp xếp các phần tử vào đúng phân lớp của nó:
  - j = 0, numMoves = 0, k = 7

index

mảng L	0	1	2	3	4	5	6	7
	0	2	3	5	6	8	9	10
	0	2	3	5	6	8	9	9
	0	2	3	5	5	8	9	9
	0	2	3	5	5	7	9	9
	0	2	3	4	5	7	9	9
	0	1	3	4	5	7	9	9
	0	1	3	4	5	7	8	9
	0	1	3	4	5	6	8	9
	0	1	3	3	5	6	8	9
	0	0	3	3	5	6	8	9
	0	0	2	3	5	6	8	9
	0	0	1					

j<L[k]

K=7 -> location = --L[7]=9

Swap evicted với phần tử này, ++numMoves rồi lặp lại

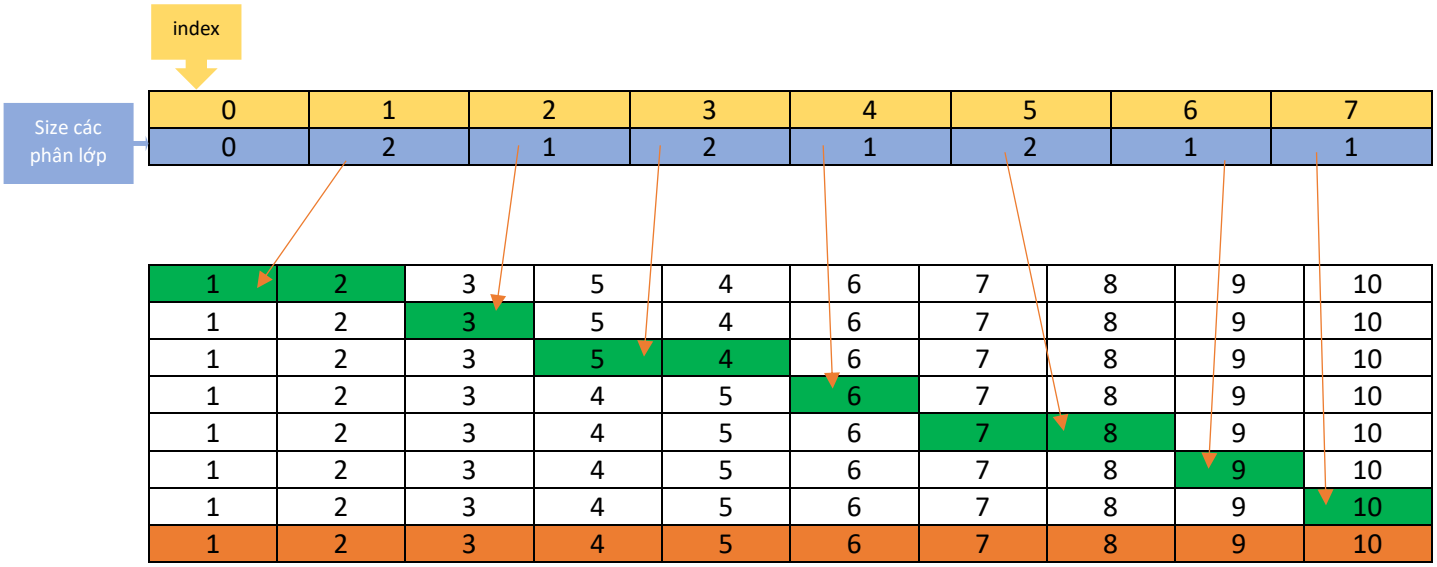
index

evicted	0	1	2	3	4	5	6	7	8	9
	10	9	3	1	2	8	5	4	7	6
	6	9	3	1	2	8	5	4	7	10
	8	9	3	1	2	6	5	4	7	10
	4	9	3	1	2	6	5	8	7	10
	2	9	3	1	4	6	5	8	7	10
	9	2	3	1	4	6	5	8	7	10
	7	2	3	1	4	6	5	8	9	10
	5	2	3	1	4	6	7	8	9	10
	1	2	3	5	4	6	7	8	9	10
	1	2	3	5	4	6	7	8	9	10

j=L[K]=0-> ++j = 1, K = 1 thì j=L[K]=0 -> ++j = 2, K = 2 thì j < L[K] = 3 nên tiếp tục gán evicted = A[2] rồi lặp lại như trên

j=L[K]=2, numMoves = 10 nên ngừng quét, đã sắp xếp xong

- Tiếp theo, tìm threshold theo công thức: threshold = (int)(1.25 * ((n / m) + 1)) = 3.
- Việc còn lại khá đơn giản, sau khi tìm size các phân lớp nhờ vào trừ ngược lại mảng L, nếu các phân lớp có size < threshold thì tiến hành sắp xếp bằng Insertion Sort, nếu không thì gọi đệ quy Flash Sort cho phân lớp đó để chia nhỏ phân lớp đó ra giúp khi chạy Insertion Sort sẽ nhanh hơn.


index
Size các phân lớp
0 1 2 3 4 5 6 7
0 2 1 2 1 2 1 1
1 2 3 5 4 6 7 8 9 10
1 2 3 5 4 6 7 8 9 10
1 2 3 5 4 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10