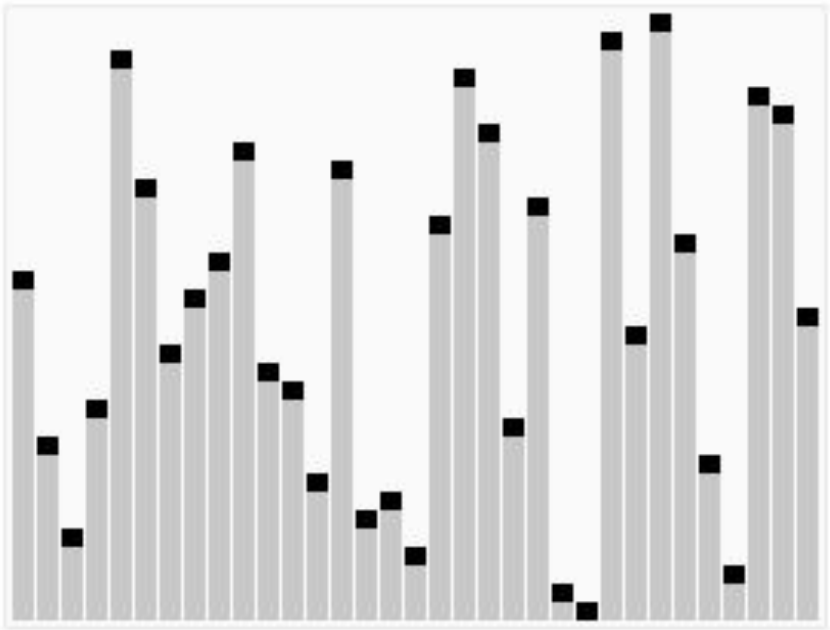
Sorting algorithm

dung.phamtrung@phenikaa-uni.edu.vn

Quick Sort

Quick Sort

Chia dãy số thành hai phần rồi sắp xếp hai phần đó

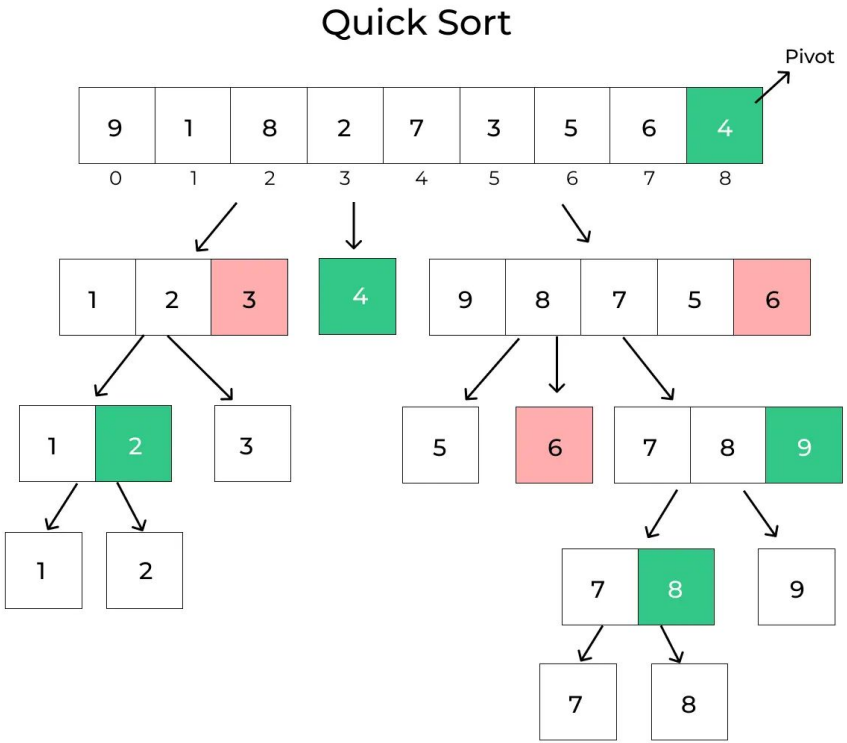


Quick Sort

Chọn một giá trị khóa

Chia dãy số thành hai phần: một là nhỏ hơn khóa, hai là lớn hơn hoặc bằng khóa

Sắp xếp hai dãy số con



Giã mã Quick Sort

```
QuickSort(Arr, L, H)
    if (L < H)
        k = Arr[ramdon(L,H)]
        i = L; j = H;
        Lặp lại
            while  (A[i] > k) i++
            while  (A[j] < k) j--
            if (i <= j)
                if (i < j) Swap(A[i],A[j])
                ++i; --j;
        khi (i < j)
        QuickSort( L, j)
        QuickSort( i, H)
```

Quick Sort

Độ phức tạp

- Trường hợp tốt nhất: O(nlogn)
- Trường hợp xấu nhất: $O(n^2)$

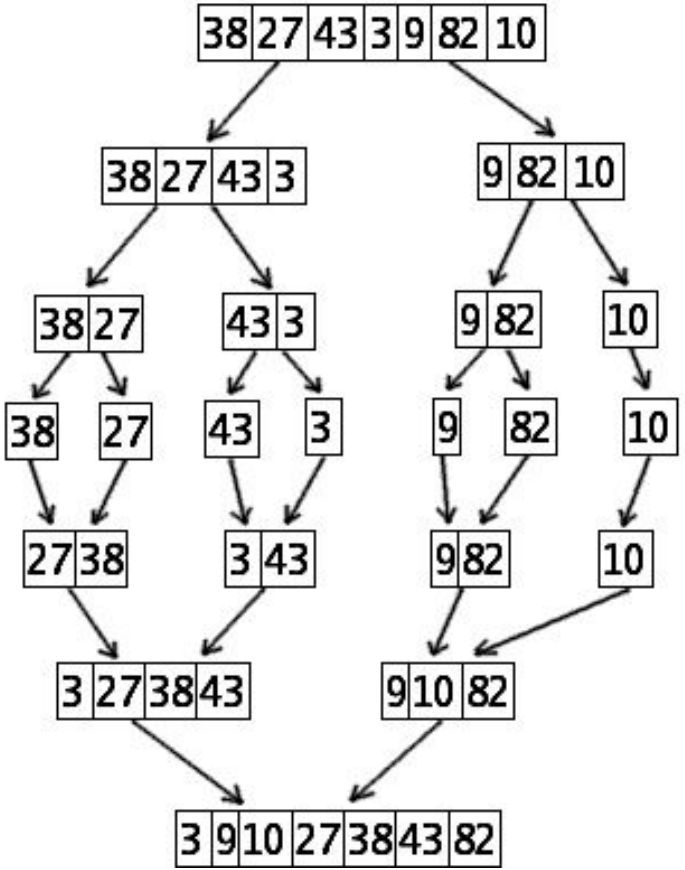
Không có tính ổn định

Là thuật toán có vận tốc trung bình là nhanh nhất

Merge Sort

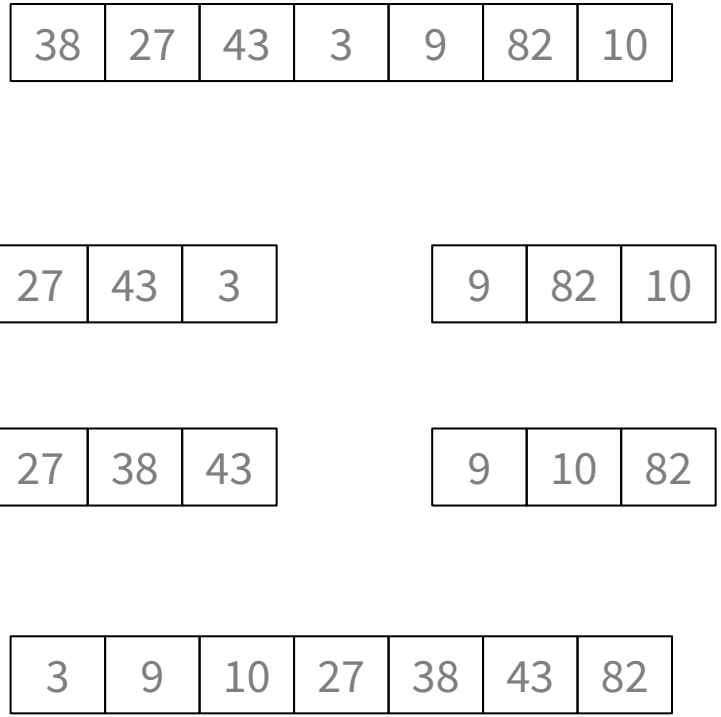
Merge Sort

Sắp xếp hai dãy đã được sắp xếp thành một dãy số được sắp xếp



Merge Sort

- Tách dãy số ra làm hai phần
- Sắp xếp hai dãy số đó
- Trộn hai dãy số lại thành dãy số sắp xếp



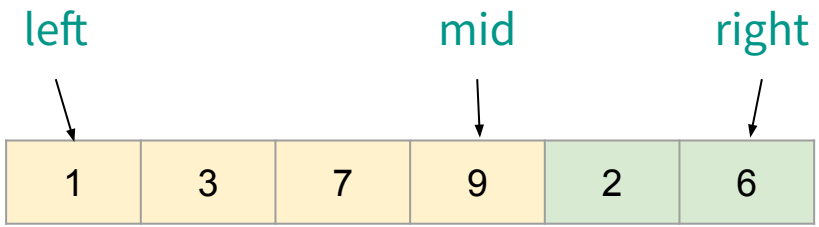
Megre

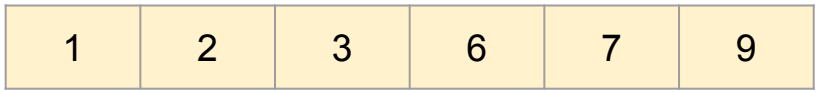
Cho hai dãy số đã được sắp xếp

- Dãy số A gồm n phần tử
- Dãy số B gồm m phần tử

Hãy trộn hai dãy số trên thành một dãy số C cũng được sắp xếp có n + m phần tử

A	B		C
1, 3, 7, 9	2, 6	1 < 2	1
3, 7, 9	2, 6	3 > 2	1, 2
3, 7, 9	6	3 < 6	1, 2, 3
7, 9	6	7 > 6	1, 2, 3, 6
7, 9			1, 2, 3, 6
			1, 2, 3, 6, 7, 9





Megre

```
Megre( Arr, left, mid, right)
        i = left;
        j = mid + 1;
        v = left
        while (i <= mid && j <= right)
                if (Arr[i] < Arr[j])
                        result[v] = Arr[i]
                        i++
                else
                        result[v] = Arr[j]
                        j++
                v++
        while (i <= mid)
                result[v] = Arr[i]
                i++
                v++
        while (j <= right)
                result[v] = Arr[j]
                j++
                v++
        return result
```

Merge Sort

```
MergeSort(Arr, left, right)
    if (left < right)
        mid = (left + right) / 2
        MergeSort(Arr, left, mid)
        MergeSort(Arr, mid + 1, right)
        Merge(Arr, left, mid, right)
```

Merge Sort

Độ phức tạp O(nlogn)

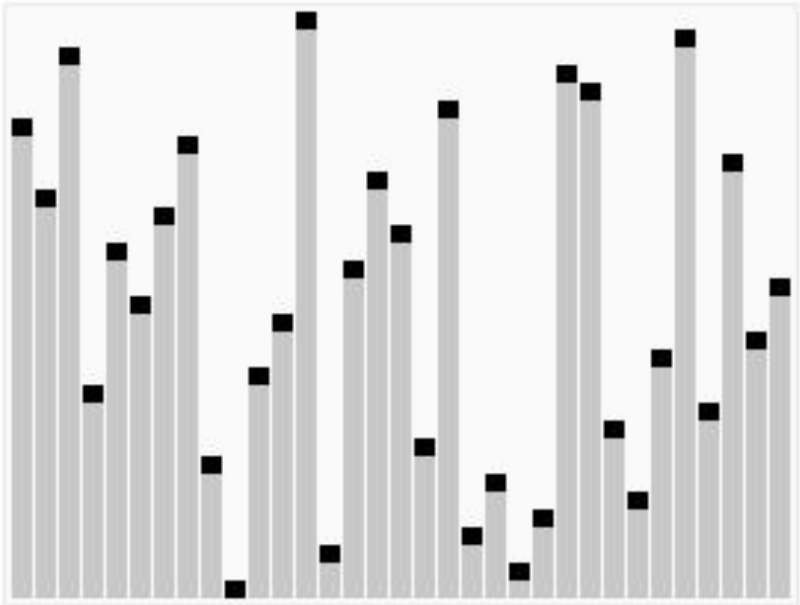
Có tính ổn định

Heap Sort

Heap Sort

Dựa trên cấu trúc Heap để tìm giá trị lớn nhất

Nó là một cải tiến của Selection Sort



Selection Sort

```
SelectionSort(Arr, n)
    for (i = n - 1; i > 0; i--)
        index_max = Arr[i]
        for (j = i - 1; j >= 0; j--)
            if (Arr[index_max] < Arr[j])
                index_max = j
        swap(Arr[index_max], Arr[i])
```

Tìm vị số lớn nhất
từ vị trí 0 đến vị trí i

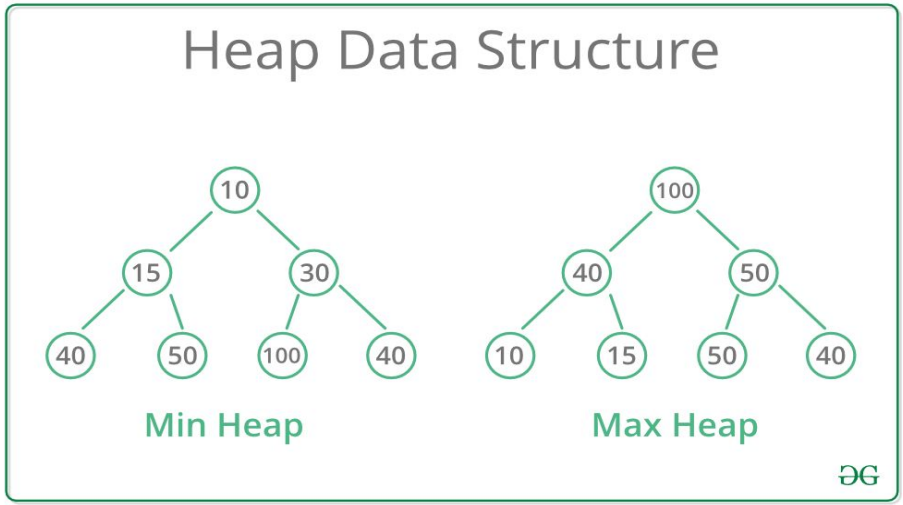
Cấu trúc Heap

Một mảng A có thể coi là một câu nhị phân với công thức

$$iLeftChild(i) = 2 \cdot i + 1$$
$$iRightChild(i) = 2 \cdot i + 2$$
$$iParent(i) = floor((i - 1) / 2)$$

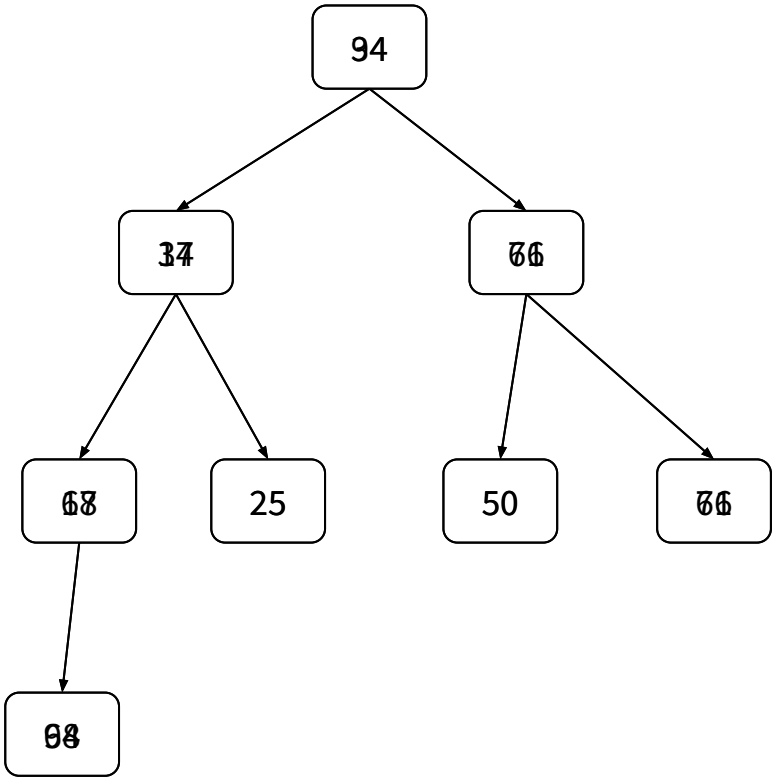
Mảng A có cấu trúc Heap cực đại nếu

$$a[i] >= a[2 * i + 1]$$
$$a[i] >= a[2 * i + 2]$$



Tạo cấu trúc Max Heap

```
MakeMaxHeap(Arr, max_index)
    par_index = (max_index + 1) / 2
    for (i = par_index; i >= 0; i--)
        if (Arr[2*i + 1] > Arr[i])
            swap(Arr[2*i + 1], Arr[i])
        if (Arr[2*i + 2] > Arr[i])
            swap(Arr[2*i + 2], Arr[i])
```



Heap Sort

```
SelectionSort(Arr, n)
    for (i = n - 1; i > 0; i--)
        index_max = Arr[i]
        for (j = i - 1; j >= 0; j--)
            if (Arr[index_max] < Arr[j])
                index_max = j
        swap(Arr[index_max], Arr[i])
```

```
HeapSort(Arr, n)
    for (i = n - 1; i > 0; i--)
        MakeHeapSort(Arr, i)
        swap(Arr[0], Arr[i])
```

Heap Sort

Độ phức tạp O(nlogn)

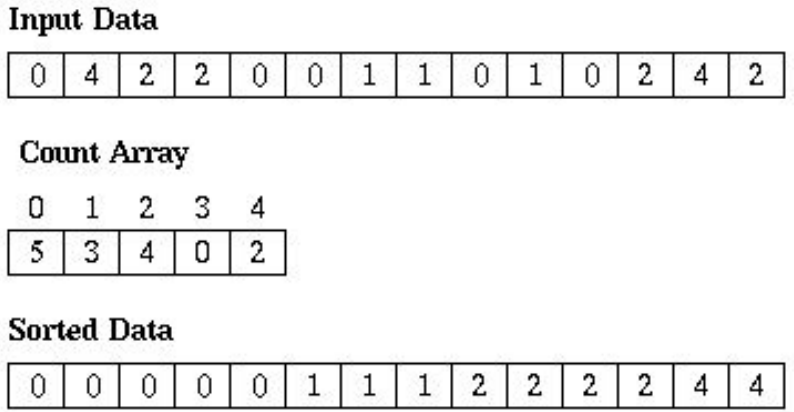
Không có tính ổn định

Couting Sort

Couting Sort

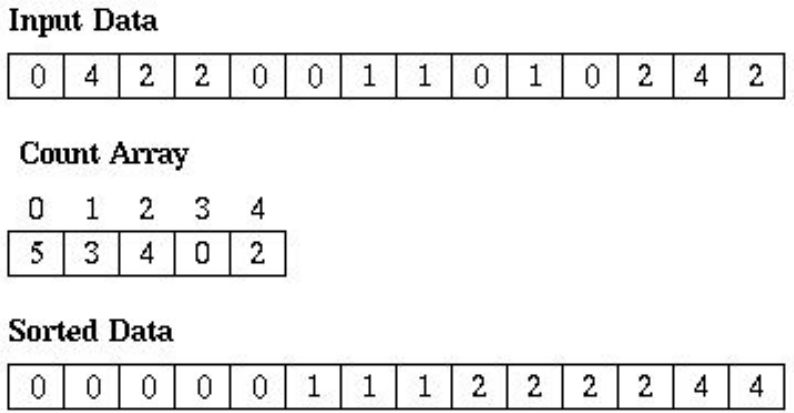
Đếm số lần xuất hiện của mỗi phần tử

Là thuật toán sắp xếp không so sánh



Couting Sort

- Đếm số lần xuất hiện của các phần tử trong danh sách
- In lại danh sách theo số đã đếm được



Couting Sort

```
CoutingSort(input, k)
    for (i = 0; i < length(input); i++)
        count[input[i]]++
    index = 0
    for (i = 0; i <= k; i++)
        for (j = 0; j < cout[i]; j++)
            output[index] = i
            index++
    return output
```

Input Data

0	4	2	2	0	0	1	1	0	1	0	2	4	2

Count Array

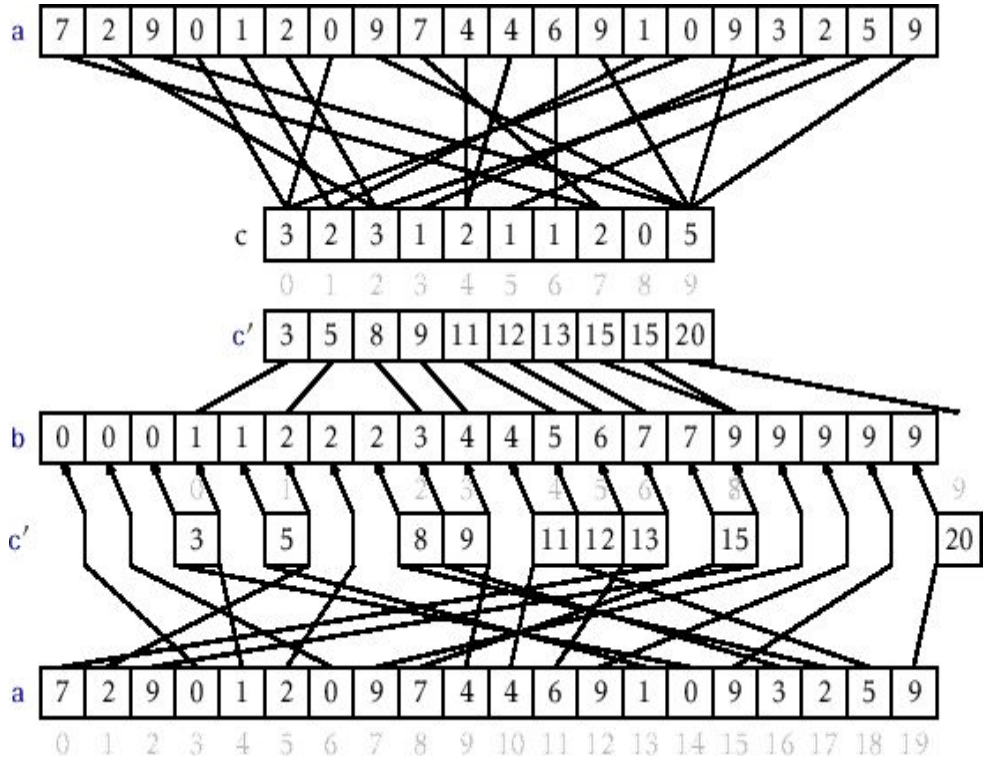
0	1	2	3	4
5	3	4	0	2

Sorted Data

0	0	0	0	0	1	1	1	2	2	2	2	4	4

Couting Sort

```
CountingSort(input, k)
    for (i = 0; i < length(input); i++)
        j = input[i]
        count[j]++
    for (i = 1; i <= k; i++)
        count[i] = count[i] + count[i - 1]
    for (i = length(input) - 1; i >= 0; i--)
        j = input[i]
        count[j] --
        output[count[j]] = j
    return output
```



Couting Sort

Độ phức tạp là O(n+k), với k là khoảng từ min đến max giá trị

Không có độ ổn định

Phù hợp với dữ liệu có k nhỏ