**Algorithm presentation**

1.Insertion Sort  
**a.**Idea

The list is divided into two sublists, sorted and unsorted, by

an imaginary wall.

• Take the first element of the unsorted region and places it

into its correct position in the sorted region

• After each placement, the size of the sorted region grows by 1 and

the size of the unsorted region shrinks by 1.

• A list of n elements requires n − 1 passes to completely

rearrange the data.

**b**.Operate

• Step 1. Set the increment variable i = 2

• Step 2. Find the correct position pos in a[1. . i − 1] to insert

a[i], i.e., where a pos − 1 ≤ a[i] ≤ a[pos]

• Set x = a[i]

• Move forward a[pos. . i − 1] one element

• Set a pos = x

• Increase i by 1 and go to Step 3

• Step 3. Check whether the end of the array is reached by

comparing i with n

• If i ≤ n then go to Step 2

• Otherwise, stop the algorithm

**C**. complexity of the algorithm

|  |  |  |
| --- | --- | --- |
| Best case | Worst case | Average case |
| O(n) | O(n2) | O(n2) |

**d.** improved case : Binary insertion sort :

Binary Insertion Sort uses binary search to find the right place to insert the selected item at each iteration. In the case of normal insertion, the sort takes O(i) (at the i-thiteration). We can reduce it to O(logi) using binary search

**e.** Draw và comment Comment:

**f.** Programming note

**g.** Ref

reference : <https://nguyenvanhieu.vn/thuat-toan-sap-xep-chen/>

## 2.Heap Sort

**a.**Idea

• A max heap is a sequence of n elements, h1, h2, . . , hn such

that hi ≥ h2i and hi ≥ h2i+1 for all i = 1,2, . . ,

• The sequence of elements h+1, . . , hn is a natural heap.

• The element h1 of a heap is the largest value.

• We also have min heap with opposite characteristics.

Ảnh có chứa đồng hồ

Mô tả được tạo tự động

**b**.Operate

• Step 1. Heap construction. Construct a heap for the array

• Step 2. Maximum deletion. Apply maximum key deletion

n − 1 times to the remaining heap

• Swap the first element and the last element of the heap

• Decrease heapSize by 1, heapSize = n − 1;

• While heapSize > 1

• Rebuild the heap at the first position, a[0. . heapSize]

• Swap the first element and the last element of the heap

• Decrease heapSize by 1, heapSize = heapSize − 1;

**C**. complexity of the algorithm

|  |  |  |
| --- | --- | --- |
| Best case | Worst case | Average case |
| O(n log2 n) | O(n log2 n) | O(n log2 n) |

It is not recommended for small numbers of elements.

**d.** improved case : This sort not have improved case.

**e.** Draw và comment Comment:

**f.** Programming note

**g.** Ref

reference : Data Structures and Algorithms

SORTING ALGORITHMS

(Part II)

Nguyễn Ngọc Thảo

Page : 6 and 11

## 3.Counting Sort

**a**.Idea

• For each element, its final position in the sorted array is

indicated by the total number of smaller elements.

• The sorted order is determined based on counting.

• This works in a situation that elements to be sorted belong to a

known small set of values: ∀i ∈ 1, n : ai ∈ N ∧ ai ∈ [l. u]

**b**.Operate

• Step 1. Compute the frequency of each of those values and store them in array

• Step 2. Distribution counting: add up sums of frequencies

• Step 3. The distribution values decide the proper positions for the last occurrences of their elements in the sorted array.

The array should be visited from right to left.

**C**. complexity of the algorithm

|  |  |  |
| --- | --- | --- |
| Best case | Worst case | Average case |
| O(n) | O(n\*n) | O(n+k). |

**d.** improved case : An analysis of counting sort

• Better efficiency yet high space complexity

• Two consecutive passes are made through the input array.

• In practice, this algorithm should be used when u ≤ n

• It would be disaster if u ≫ n

• Counting radix sort: an alternative implementation of radix

sort that avoids using bins

**e.** Draw và comment Comment:

**f.** Programming note

**g.** Ref

reference : Data Structures and Algorithms

SORTING ALGORITHMS

(Part II)

Nguyễn Ngọc Thảo

Page : 17