Introduction to Sorting

Richard Hua

Semester 2, 2021

Sorting problem

- Sorting is the problem of rearranging input data according to a particular linear order.
- Common ordering examples: dictionary (lexicographic) order on strings, integers.
- Some problems are easier to solve on sorted data. e.g. binary search, finding frequencies of different items.
- The problem of sorting is to rearrange an input list of **keys**, which can be compared using a **total order**i, $\leqslant$, into an output list such that if i and j are keys and i precedes j in the output list, then $i \leqslant j$.

Attributes of sorting algorithms

Definition

A sorting algorithm is called **comparison-based** if the only way to gain information about the total order is by comparing a pair of elements at a time via the order $\leqslant$.

What about non-comparison-based sorting?

Definition

A sorting algorithm is called **in-place** if it uses only a fixed additional amount of working space, independent of the input size.

Obviously quite useful, although this is often hard to tell at first glance.

Attributes of sorting algorithms

Definition

A sorting algorithm is called **stable** if whenever two objects have the same key in the input, they appear in the same order in the output.

How is this useful?
Consider a list of hospital patients and say each patient has a name and an age and say the list is currently sorted by their names.
E.g.
$A = [(\text{Abby}, 24), (\text{Angel}, 21), \cdots, (\text{Bill}, 24), \cdots (\text{Peter}, 24) \cdots]$.
Now we sort the list A by patients' age.
Unstable result $A = [\cdots, (\text{Peter}, 24), (\text{Abby}, 24), (\text{Bill}, 24), \cdots]$.
Stable result $A = [\cdots, (\text{Abby}, 24), (\text{Bill}, 24), (\text{Peter}, 24), \cdots]$.

Additional assumptions

- We do not know any information about the keys themselves (e.g. all keys are small integers) only their order relation.
- Two elementary operations: **comparison** (of two items) and a **move**.
- Number of comparisons needed is generally the dominant term.
- Actual time needed by a data move depends on the data structure of the list (e.g. array vs linked-list).

Basic operations

For a comparison, we choose keys x and y and answer the question is '$x < y$?' (or maybe we ask which of $x < y$, $x = y$, $y < x$ is true).

Note that there are two types of data moves:

- **Swap**: $\Theta(1)$ with arrays. What about linked-lists?
- **Insert**: Not $\Theta(1)$ with arrays.

In practice ...

There is often many more factors to consider in practice:

- How many items have to be sorted.
- Are the items only related by the order relation, or do they have other restrictions (e.g. are they all small integers in range from 1 to 200).
- To what extent they are pre-sorted?
- Can they be placed into an internal (fast) computer memory or must they be sorted in external (slow) memory, such as on disk (**external sorting**.).

Conclusion: There is no 'best' sorting algorithm, it is important to understand the strengths and weaknesses of each.

Selection sort

Find the maximum element of the unsorted part of the list by sequential search, and move it to the end of the sorted part. Iterate.

function SELECTIONSORT(list $a[0 \cdots n-1]$)
    for $i \leftarrow n-1$ to 0 **do**
        $k \leftarrow$ FINDMAX($a[0 \cdots i]$)
        if $k \neq i$ **then**
            SWAP(a,i,k)

Time complexity

function FINDMAX(array $a[0, \cdots, n-1]$)
    $k \leftarrow 0$
    for $j \leftarrow 1$ to $n-1$ **do**
        if $a[k] < a[j]$ **then**
            $k = j$
    return k

FINDMAX($a[i \cdots n-1]$) does $n-1-i$ comparisons.
The total number of key comparisons is $\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2} \in \Theta(n^2)$.
The number of swaps is at most $n-1$.
The time complexity of SELECTIONSORT is $\Theta(n^2)$ in all cases.

Best/Worst case

Question

Does SELECTIONSORT have best/worst case?

Best/Worst case

Question

Does SELECTIONSORT have best/worst case?

Answer: Yes, they have the same asymptotic time complexity as the average case but the exact number of swap operation is different.
Does it make any difference in practice?
What if data movement is very computationally expensive? e.g. very large data items to move around.

Stable/in-place

Question

Is SELECTIONSORT in-place?

Yes. We don't need much extra memory space (independent of input size).

Question

Is SELECTIONSORT stable?

Stable

Exercise

Execute SELECTIONSORT on the following input.

3	2	1_a	1_b