Algorithm Lab

Week 1: Sort

Description:

In computer science, sorting algorithm is an algorithm that puts elements of a list in a certain order (From Wikipedia). Thus, we can define sorting algorithms as followed:

Instance: a list of elements $A = (a_1, a_2, \dots, a_n)$ *and a comparator* $C(v_0, v_1) \to \{True, False\}$

Result: a list $B = (b_1, b_2, \dots, b_n)$ *that* $\{a_1, a_2, \dots, a_n\} = \{b_1, b_2, \dots, b_n\}$ *and for all* $1 < i \leq n$, $C(b_{i-1}, b_i)$ *is* $True$.

If we want to sorting a list of integers by increasing order, our comparator, we should use $\leq$ as our comparator. Please design an algorithm to sorting integers in increasing order.

Algorithm Design

Insertion sort will divide data to 2 partitions, sorted and unsorted.

1 Let first element as sorted part and the others as unsorted part.

2 Insert an unsorted element v_i to sorted part and keep them in certain order.

- 2.1 Find minimum j that $C(v_j, v_i) = False$.
- 2.2 Shifting all v_k that $k \geq j$.
- 2.3 Insert v_i to current position.

3 Repeat step 2 until unsorted part contains no elements.

Implementation

Language: C

```
int* sort (int* A, int n)
{
        // index i divide A[] to 2 partitions
        // A[0…i-1] are sorted, A[i…n-1] are unsorted
        for (int i = 1; i < n; ++i)
        {
                int j = i, val = A[j];
                while (j > 0 && !(A[j-1] <= val))
                {
                        A[j] = A[j-1];
                        --j;
                }
                A[j] = val;
        }
        return A;
}
```

Analysis

Space complexity

Assume that instance is an n elements array.

- Instance: n values and 1 index (A, n)
- Divider: 1 index (i)
- Ordering maintain: 1 index (j) and 1 value(val)

Totally needs $n+1$ values and 3 indices, so the space complexity is $O(n)$

Time complexity

Best case: already sorted no. of comparisons $n-1$

$O(n)$

worst case: Reverse order no. of comparisons

$n(n-1)/2 \sim n(n-1) \sim n^2 - n$

$O(n^2)$