Insertion Sort

Richard Hua

Semester 2, 2021

The problem of arrangements

We have n different types of object and we want to put r of them in a row. The ordering matters. We have an unlimited supply of each type of object. How many ways can we do this?
Answer: n^r.
Given a set X, an r-**permutation of** X is an arrangement of r of the elements of X in order, with no repetition of elements. We denote the number of r-permutations of a set with n elements by $P(n, r) = n(n-1)(n-2)\cdots(n-r+1) = \frac{n!}{(n-r)!}$.

Exercise

Three men and three women are going to occupy a row of six seats. In how many different orders can they be seated so that men occupy the two end seats?

Exercise

Exercise

Three men and three women are going to occupy a row of six seats. In how many different orders can they be seated so that men occupy the two end seats?

3×2 ways to seat two men on the end seats. The 4 remaining people can be seated in 4! ways.

Selections

We now consider counting problems in which the order we make our choices does not matter.

We choose an r-element subset of X. We denote the number of r-element subset of X by $\binom{n}{r}$.

If order matters, we have $\frac{n!}{(n-r)!}$ ways. But we have repeatedly counted the same set of elements $r!$ times. So $\binom{n}{r} = \frac{P(n,r)}{r!}$.

Exercise

An ordinary deck of 52 cards consists of four suits of 13 denominations each.

- *How many five-card poker hands are there?*
- *How many poker hands contain cards all of the same suit?*
- *How many poker hands contain three cards of one denomination and two cards of a second denomination?*

Selection with repetition

Suppose we have a bag containing a large number of marbles in each each of three colors: brown, yellow and blue.
In how many ways can we select 5 marbles from the bag?
We build a mapping from each choice to the set of sequences of five 0s and two 1s.
A selection if i brown marbles, j blue marbles and p yellow marbles corresponds to the sequence of i 0s followed by a 1 followed by j 0s followed by a 1 followed by p 0s (e.g. '1 brown, 0 blue, 4 yellow' is 0110000).
There are 7 symbols and two of them has to be 1s so $\binom{7}{2}$.

Selection with repetition

In general, the number of ways of selecting n objects from k types, with order unimportant and repetition allowed is $\binom{n+k-1}{k-1} = \binom{n+k-1}{n}$.

Exercise

A math lecturer is about to assign grades of A, B, C or D, where D is a failing grade, to his class of 100 *students. How many grade distributions are possible if:*

- *Any distribution is allowed.*
- *No more than* 80% *of the students may pass.*
- *No few than* 50% *of the students may pass.*

Inversions

Definition

Given a list $a = [a_0, a_1, \cdots, a_{n-1}]$, an **inversion** in a is an ordered pair of positions (i, j) such that $i < j$ but $a[i] > a[j]$

Example

Example

Using the natural order of integers, the list $[3, 2, 5]$ has only one inversion corresponding to the pair $(3, 2)$.
The list $[5, 2, 3]$ has two inversions, $(5, 2)$ and $(5, 3)$.

Exercise

Assuming all keys are different, what is the maximum number of inversions in a list of size n?

Inversions

Can be used as a measurement as to how ‘out of order’ a list is.
Performance of many sorting algorithm is dependent on the number of inversions the input list has.
A sorted list has no inversions so the process of sorting can be interpreted as the process of removing inversions (using data moves) from the input list.
Note that this is not always the case.

Question

Selection sort does not have a very good time complexity compared to the other sorting algorithms we will see in the course. But the number of data movement Selection sort does is very small. How do we explain this behaviour?

Inversions

Question

Selection sort does not have a very good time complexity compared to the other sorting algorithms we will see in the course. But the number of data movement Selection sort does is very small. How do we explain this behaviour?

Selection sort eliminates a large number of inversions per swap. Once we use swap to put the max element from the unsorted part of the list to the end, this item is effectively at the correct place it should be. i.e. we have eliminated all inversions associated with the max item. And so we only need at most $n-1$ data movement. Number of comparisons isn't very good, a lot of information about the keys are lost in-between iterations.

Insertion sort

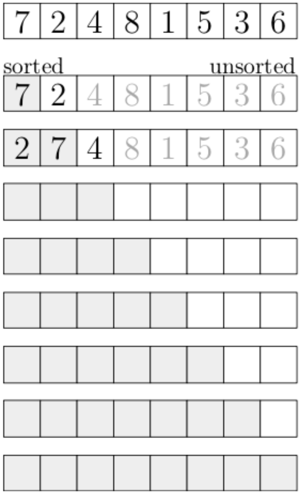
This is the sorting method used by card players to arrange cards in a hand.
Iterative algorithm and works as follows. The algorithm splits the input list of size n into a head (‘sorted’) and tail (‘unsorted’) sublist.

- The head sublist has initial size 1.
- Repeat the following step until the tail sublist is empty:
  - choose the first element x in the tail sublist;
  - find the last element y in the head sublist so that $y \leqslant x$;
  - insert x after y in the head sublist.

Pseudo-code

```
function INSERTIONSORT(list $a[0 \cdots n-1]$)
    for $i \leftarrow 1$ to $n-1$ do
        $j \leftarrow i-1$
        while $a[j] > a[j+1]$ and $j \geqslant 0$ do
            SWAP$(a, j, j+1)$
            $j \leftarrow j-1$
    return $a$
```

Example



Analysis

How many data movement (swap) do we need?
We only swap two adjacent elements,$a[j], a[j+1]$, if they are out of order, i.e. $(a[j], a[j+1])$ is an inversion. So we eliminate exactly 1 inversion per swap.
Total number of swaps is the same as the number of inversions in the input list.
What about comparisons?
At most $n-1$ plus the number of inversions.

Best case analysis

Exercise

What is the best case running time?

If the input is sorted, only $n - 1$ comparisons and 0 swaps are needed so $\Theta(n)$.

Worst case analysis

Exercise

What is the worst case running time?

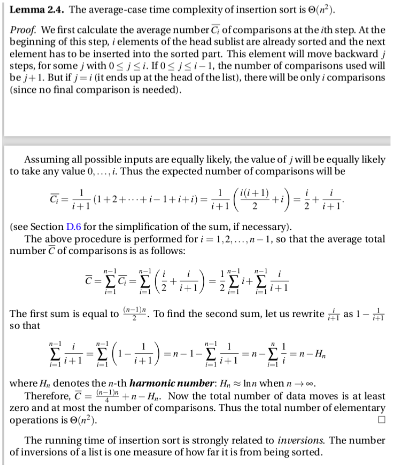
Both number of comparisons and swaps needed are in order of number of inversions in the list. Maximum number of inversions in a list of length n is $\Theta(n^2)$ (everything in reversed order).

Average case analysis

Question

What is the average case running time?

Textbook proof



Lemma 2.4. The average-case time complexity of insertion sort is $\Theta(n^2)$.

Proof. We first calculate the average number $\overline{C_i}$ of comparisons at the ith step. At the beginning of this step, i elements of the head sublist are already sorted and the next element has to be inserted into the sorted part. This element will move backward j steps, for some j with $0 \le j \le i$. If $0 \le j \le i-1$, the number of comparisons used will be $j+1$. But if $j = i$ (it ends up at the head of the list), there will be only i comparisons (since no final comparison is needed).

Assuming all possible inputs are equally likely, the value of j will be equally likely to take any value $0, \dots, i$. Thus the expected number of comparisons will be

$$\overline{C_i} = \frac{1}{i+1}(1+2+\cdots+i-1+i+i) = \frac{1}{i+1}\left(\frac{i(i+1)}{2}+i\right) = \frac{i}{2}+\frac{i}{i+1}.$$

(see Section D.6 for the simplification of the sum, if necessary).

The above procedure is performed for $i = 1, 2, \dots, n-1$, so that the average total number $\overline{C}$ of comparisons is as follows:

$$\overline{C} = \sum_{i=1}^{n-1} \overline{C_i} = \sum_{i=1}^{n-1}\left(\frac{i}{2}+\frac{i}{i+1}\right) = \frac{1}{2}\sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1}\frac{i}{i+1}$$

The first sum is equal to $\frac{(n-1)n}{2}$. To find the second sum, let us rewrite $\frac{i}{i+1}$ as $1-\frac{1}{i+1}$ so that

$$\sum_{i=1}^{n-1}\frac{i}{i+1} = \sum_{i=1}^{n-1}\left(1-\frac{1}{i+1}\right) = n-1-\sum_{i=1}^{n-1}\frac{1}{i+1} = n-\sum_{i=1}^{n}\frac{1}{i} = n-H_n$$

where H_n denotes the n-th ***harmonic number***: $H_n \approx \ln n$ when $n \to \infty$.

Therefore, $\overline{C} = \frac{(n-1)n}{4} + n - H_n$. Now the total number of data moves is at least zero and at most the number of comparisons. Thus the total number of elementary operations is $\Theta(n^2)$. □

The running time of insertion sort is strongly related to *inversions*. The number of inversions of a list is one measure of how far it is from being sorted.

Let's take a different approach

Question

Assuming all keys are different. What is the average number of inversions in a list of size n?

If all keys are different, we can generalize a list of size n an as a permutation of length n.
We denote the set of all permutations of length n by S_n. We denote the number of inversions in permutation s by $\mathrm{inv}(s)$. For a permutation s, let $\mathrm{rev}(s)$ denote its reversed permutation (e.g. $\mathrm{rev}([3,1,2]) = [2,1,3]$.
Observation: for $s \in S_n$, $\mathrm{inv}(s) + \mathrm{inv}(\mathrm{rev}(s)) = \binom{n}{2}$.

Continued

Question

Assuming all keys are different. What is the average number of inversions in a list of size n?

Average number of inversion can be calculated using the formula $\frac{\sum_{s\in S_n} \mathrm{inv}(s)}{n!} = \frac{\frac{1}{2}n!\binom{n}{2}}{n!} = \frac{\binom{n}{2}}{2} = \frac{n(n-1)/2}{2} = \frac{n(n-1)}{4} \in \Theta(n^2)$.

Conclusion: Insertion sort average-case performance is $\Theta(n^2)$.

Additional properties

Question

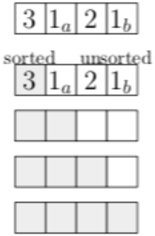
Is Insertion sort in-place?

Yes.

Question

Is Insertion sort stable?

Insertion sort is stable



Additional notes

Keep swapping two adjacent items seem very inefficient. Can we improve this?
Using a linked-list can reduce the number of data movement.

Question

How does it affect the overall performance?

Additional notes

Question

How does it affect the overall performance?

Sadly, there is no asymptotic improvement since the overall time complexity is dominated by the number of comparisons.

Additional notes

Any other ideas to improve Insertion sort?
Binary search to find insertion point in the array.
Swaps will take $\Theta(n^2)$ time. Not useful again.
Using bigger step size, gradually reducing the step size to 1.
Shell sort:

- Fix $h = n/2$. Run Insertion sort on the sublists formed by taking every hth element. Iterate with $h- = 1$.

Last iteration is just standard insertion sort (on an almost sorted list).
Properly chosen h can lead to $O(n(\log n)^2)$ time.
Simple algorithm but almost impossible to analyze.
Nothing is know about the average-case running time. Empirical results suggest it's something close to $O(n^{7/6})$.
Very easy to implement and works well enough in practice. A lot better than Insertion sort.