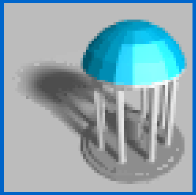
Introduction to Sorting Methods

- Weekly Reading:
  Lecture Notes & Chapter 7 (CLRS)

- Today's lecture
  - Basics of Sorting
  - Elementary Sorting Algorithms
    - Selection sort
    - Insertion sort
    - Shellsort

Sorting



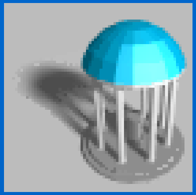
- Given n *records*, $R_1 \ldots R_n$, called a *file*. Each record R_i has a key K_i, and may also contain other (satellite) information. The keys can be objects drawn from an arbitrary set on which equality is defined. There must be an order relation defined on keys, which satisfy the following properties:
  - *Trichotomy*: For any two keys a and b, exactly one of $a < b$, $a = b$, or $a > b$ is true.
  - *Transitivity*: For any three keys a, b, and c, if $a < b$ and $b < c$, then $a < c$.

  The relation > is a *total ordering* (*linear ordering*) on keys.

Basic Definitions

- *Sorting*: determine a permutation $P = (p_1, \ldots , p_n)$ of n records that puts the keys in non-decreasing order $K_{p_1} \leq \ldots \leq K_{p_n}$.
- *Permutations*: a one-to-one function from {1, …, n} onto itself. There are $n!$ distinct permutation of n items.
- *Rank*: Given a collection of n keys, the *rank of key* is the number of keys that are $\prec$ than it. That is, rank(K_j) = $|\{K_i | K_i < K_j\}|$. If the keys are distinct, then the ranks of a key gives its position in the output file.

Terminology



- *Internal* (the file is stored in main memory and can be randomly accessed) vs. External (the file is stored in secondary memory & can be accessed sequentially only)
- *Comparison-based sort*: where it uses only the relation among keys, not any special property of the presentation of the keys themselves
- *Stable sort*: records with equal keys retain their original relative order; i.e. $i < j \; \& \; Kp_i = Kp_j \Rightarrow p_i < p_j$
- *Array-based* (consective keys are stored in consecutive memory locations) *vs. List-based sort* (may be stored in nonconsecutive locations in a linked manner)
- *In-place sort*: it needs a constant amount of extra space in addition to that needed to store keys

Elementary Sorting Methods

- Easier to understand the basic mechanisms of sorting
- Maybe more suitable for small files
- Good for well-structured files that are relatively easy to sort, such as those almost sorted
- Can be used to improve efficiency of more powerful methods

Sorting Categories

● Sorting by Insertion	insertion sort, shellsort
● Sorting by Exchange	bubble sort, quicksort
● Sorting by Selection	selection sort, heapsort
● Sorting by Merging	merge sort
● Sorting by Distribution	radix sort

Selection Sort

```
1. for i = n downto 2 do {
2.     max ← i
3.     for j = i - 1 downto 1 do{
4.         if A[max] < A[j] then
5.             max ← j
6.     }
7.     t ← A[max]
8.     A[max] ← A[i]
9.     A[i] ← t
10. }
```

Algorithm Analysis

- In-place sort
- Not stable
- The number of comparison is $\Theta(n^2)$ in the worst case, but it can be improved by a sequence of modifications, which leads to heapsort (see next lecture).

Insertion Sort

```
1.  for j = 2 to n do {
2.        key ← A[j]
3.        i ← j - 1
4.        while i > 0 and key < A[i] {
5.              A[i+1] ← A[i]
6.               i ← i - 1
7.         }
8.         A[i+1] ← key
9.   }
```

Algorithm Analysis

- In-place sort
- Stable
- If A is sorted: $\Theta(n)$ comparisons
- If A is reversed sorted: $\Theta(n^2)$ comparisons
- If A is randomly sorted: $\Theta(n^2)$ comparisons

Worst Case Analysis

- The *maximum* number of comparison while inserting A[*i*] is (*i-1*). So, the number of comparison is

$$
\begin{aligned}
C_{wc}(n) &\le \textstyle\sum_{i = 2 \text{ to } n} (i - 1) \\
&\le \textstyle\sum_{j = 1 \text{ to } n-1} j \\
&= n(n-1)/2 \\
&= \Theta(n^2)
\end{aligned}
$$

Average Case Analysis

- Consider when we try to insert the key A[i]. There are i places where it can end up after insertion. Assume all possibilities are equally likely with probability of $1/i$. Then, the average number of comparisons to insert A[i] is

$$\sum_{j = 1 \text{ to } i-1} [\, 1/i * j \,] + 1/i * (i - 1) = (i+1)/2 - 1/i$$

- Summing over insertion of all keys, we get

$$C_{avg}(n) = \sum_{i = 2 \text{ to } n} [(i+1)/2 - 1/i]$$

$$= n^2/4 + 3n/4 - 1 - \ln n = \Theta(n^2)$$

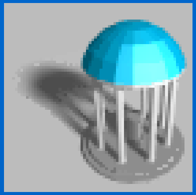
- Therefore, $T_{avg}(n) = \Theta(n^2)$

Analysis of Inversions in Permutation

- Worst Case: $n(n-1)/2$ inversions
- Average Case:
  - Consider each permutation Π and its transpose permutation Π^T. Given any Π, Π^T is unique and $\Pi \neq \Pi^T$
  - Consider the pair (i, j) with $i < j$, there are $n(n-1)/2$ pairs.
  - (i, j) is an inversion of Π if and only if $(n-j, n-i)$ is not an inversion of Π^T. This implies that the pair (Π, Π^T) together have $n(n-1)/2$ inversions.

  $\Rightarrow$ The average number of inversions is $n(n-1)/4$.

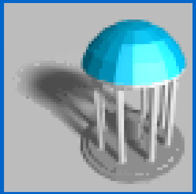
Theorem



- **Any algorithm that sorts by comparison of keys and removes at most one inversion after each comparison must do at least $n(n-1)/2$ comparisons in the worst case and at least $n(n-1)/4$ comparisons on the average.**

$\Rightarrow$ If we want to do better than $\Theta(n^2)$, we have to remove more than a constant number of inversions with each comparison.

Insertion Sort to Shellsort



- Shellsort is a simple extension of insertion sort. It gains speed by allowing exchanges with elements that are far apart.
- The idea is that rearrangement of the file by taking every *h*th element (starting anywhere) yield a sorted file. Such a file is "*h*-sorted". A "*h*-sorted" file is *h* independent sorted files, interleaved together.
- By *h*-sorting for some large values of "increment" *h*, we can move records far apart and thus make it easier for *h*-sort for smaller values of *h*. Using such a procedure for any sequence of values of *h* which ends in 1 will produce a sorted file.

Shellsort

- A family of algorithms, characterized by the sequence $\{h_k\}$ of increments that are used in sorting.
- By interleaving, we can fix multiple inversions with each comparisons, so that later passes see files that are “nearly sorted”. This implies that either there are many keys not too far from their final position, or only a small number of keys are far off.

Shellsort

```
1.  h ← 1
2.  While h ≤ n {
3.      h ← 3h + 1
4.   }
5.  repeat
6.      h ← h/3
7.      for i = h to n do {
8.          key ← A[i]
9.           j ← i
10.         while key < A[j - h] {
11.             A[j] ← A[j - h]
12.              j ← j - h
13.              if j < h then break
14.          }
15.         A[j] ← key
16.     }
17. until h ≤ 1
```

Algorithm Analysis

- In-place sort
- Not stable
- The exact behavior of the algorithm depends on the sequence of increments -- difficult & complex to analyze the algorithm.
- For $h_k = 2^k - 1$, $T(n) = \Theta(n^{3/2})$