Tutorial 2

Sorting Algorithms: Probabilistic Analysis

Outline

- Bubble Sort
- Quick Sort
- Counting Sort
- Comparison of sorting algorithms

1. Bubble Sort

- Theorem: The average number of comparisons done by bubble sort is $\theta(n^2)$
  - Proof by inversion
  - Facts
    - Each comparison remove at least one inversion
    - All the inversions must be eliminated during the process of sorting

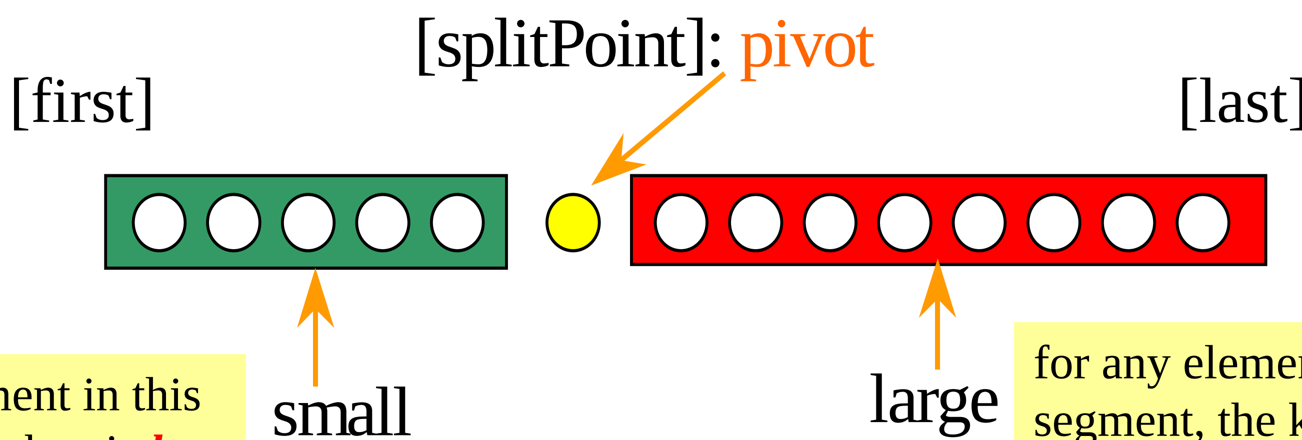
- (1) $\Omega(n^2)$
  - Assume n numbers, 1,2,…,n
  - For k ($1 \leqslant k \leqslant n$), consider the expected number of inversions of k
    - K=n, expected inversions $\geqslant \frac{1}{n}[(n-1)+(n-2)+...+1+0] = \frac{n(n-1)}{2n}$
    - K=n-1, expected inversions $\geqslant \frac{1}{n}[(n-2)+...+1+0+0] = \frac{(n-1)(n-2)}{2n}$
    - K=i, expected inversions $\geqslant \frac{i(i-1)}{2n}$
  - The expected number of inversions $\geqslant \sum_{i=1}^{n} \frac{i(i-1)}{2n}$
  - Which is $\Omega(n^2)$
- (2) $O(n^2)$ – easy to proof
- According to (1)(2), the average number of comparisons is $\theta(n^2)$.

- A more simple solution
- Computing the average number of inversions in inputs of size n (n>1):
  - For any i, j, ($1 \leq j \leq i \leq n$), either (x_i,x_j) or (x_j ,x_i) is an inversion, thus the probability of inversion is 1/2.
  - The number of transpose pairs (x_i,x_j) on n distinct integers is $n(n-1)/2$.
  - So, the average number of inversions in all possible inputs is $\boldsymbol{n(n-1)/4}$.
- **The average behavior of any sorting algorithm that remove at most one inversion per key comparison must in $\Omega(n^2)$**

2. Quick Sort

Quicksort: the Strategy

- Dividing the array to be sorted into two parts: “small” and “large”, which will be sorted recursively.



for any element in this segment, the key is ***less than pivot***.

for any element in this segment, the key is ***not less than pivot***.

To be sorted recursively

QuickSort: the algorithm

- Input: Array *E* and indexes *first*, and *last*, such that elements *E*[*i*] are defined for *first*≤*i*≤*last*.
- Output: *E*[*first*],…,*E*[*last*] is a sorted rearrangement of the same elements.
- The procedure:

```
void quickSort(Element[ ] E, int first, int last)
    if (first<last)
        Element pivotElement=E[first];
        Key pivot=pivotElement.key;
        int splitPoint=partition(E, pivot, first, last);
        E[splitPoint]=pivotElement;
        quickSort(E, first, splitPoint-1);
        quickSort(E, splitPoint+1, last);
    return
```

The splitting point is chosen arbitrarily, as the first element in the array segment here.

How to select pivot?

- Fixed strategy
- Random strategy

The fixed strategy (divide and conquer)

- Input: A, n
- Divide
  - Select a item from a fixed position
  - Partition: q, n-q-1
- Conquer:
  - QuickSort(A,first,q-1)
  - QuickSort(A,q+1,last)
- Combine
- $T(n)=T(q)+T(n-q-1)+\Theta(n)$

The fixed strategy

- Worst case
  - Partition: 0, n-1
  - Input is sorted or inverse sorted
  - $T(n)= T(0)+T(n-1)+\Theta(n)=T(n-1)+\Theta(n)$
  - $T(n) \in \Theta(n^2)$
- Best Case
  - Partition: n/2, n/2
  - $T(n)= 2T(n/2)+\Theta(n)$
  - $T(n) \in \Theta(n\lg n)$

- Average case? Θ(nlgn)
- What about partition: n/5, 4n/5?

  $T(n)= T(n/5)+T(4n/5)+\Theta(n)$

  Use the recursion tree for analysis, we have $T(n)\in\Theta(n\lg n)$
- What about partition: 0.001n, 0.999n?
  Θ(nlgn)
- What about partition: n/k, ((k-1)n)/k? Θ(nlgn)

Randomized Strategy

- Randomized QuickSort
- Input: Array *E* and indexes *first*, and *last*, such that elements *E*[*i*] are defined for *first*≤*i*≤*last*.
- Output: *E*[*first*],…,*E*[*last*] is a sorted rearrangement of the same elements.
- The procedure:

```
void Random_quickSort(Element[ ] E, int first, int last)
    if (first<last)
        Element pivotElement=E[Random(first,last)];
        Key pivot=pivotElement.key;
        int splitPoint=partition(E, pivot, first, last);
        E[splitPoint]=pivotElement;
        Random_quickSort(E, first, splitPoint-1);
        Random_quickSort(E, splitPoint+1, last);
    return
```

What is the expected number of comparisons?

- Input A, n
- Elements of A can be denoted as
  $z_1, z_2, \ldots, z_n$ ($z_1 < z_2 < \ldots < z_n$) (assume distinct)
- Let random variables

$$X_{ij} = \begin{cases} 1 \text{ if } z_i \text{ is compared to } z_j \\ 0 \text{ else} \end{cases}$$

- The total comparison $X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$
- The expectation of X?

$$E[X] = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} [X_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$$

$\Pr\{z_i \text{ is compared to } z_j\} = ?$

- When two items are compared?
  - For example A={1,3,2,5,4,6}, if pivot=4, then A is separated into L={1,2,3}, R={5,6}
    - 4 is compared to all elements in L and R
    - Elements in L is never compared to elements in R
- Depends on the chosen of pivot
  - Consider $Z_{ij}=\{z_i, z_{i+1}, \ldots, z_j\}$
  - If z_x ($i<x<j$) is chosen as a pivot first, then z_i and z_j will not be compared
  - So, z_i is compared to z_j if and only if z_i (or z_j) is chosen as a pivot before any other item in Z_{ij}

As probability of each item is chosen in Z_{ij} is equal

$$\Pr\{z_i \text{ is compared to } z_j\} = \Pr\{z_i \text{ is first pivot chosen from } Z_{ij}\}$$

$$+\Pr\{z_j \text{ is first pivot chosen from } Z_{ij}\} = \frac{2}{j-i+1}$$

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \Pr\{z_i \text{ is compared to } z_j\}$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1}$$

$$< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k}$$

$$\approx 2\sum_{i=1}^{n-1} (\ln(n) + \gamma) \quad \text{(Harmonic Series, book P22, equation 1.11)}$$

$$= O(n\lg n)$$

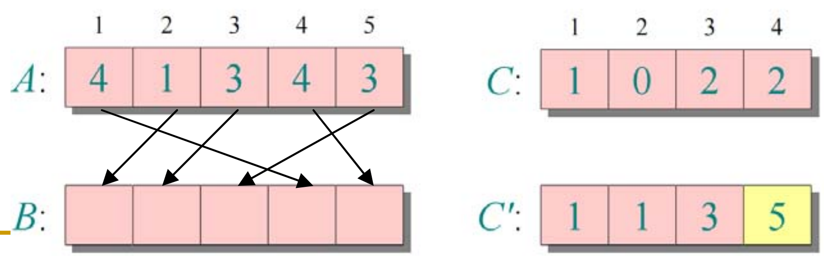
So the expected running time of randomized QuickSort is O(nlgn)

3. Counting Sort

Algorithm Description [CLRS P168]

- ***Input***: $A[1\,..\,n]$, where $A[j] \in \{1, 2, \ldots, k\}$.
- ***Output***: $B[1\,..\,n]$, sorted.
- ***Auxiliary storage***: $C[1\,..\,k]$.

for $i \leftarrow 1$ **to** k
    do $C[i] \leftarrow 0$
for $j \leftarrow 1$ **to** n
    do $C[A[j]] \leftarrow C[A[j]] + 1$ $\quad \triangleright\ C[i] = |\{\text{key} = i\}|$
for $i \leftarrow 2$ **to** k
    do $C[i] \leftarrow C[i] + C[i-1]$ $\quad \triangleright\ C[i] = |\{\text{key} \le i\}|$
for $j \leftarrow n$ **downto** 1
    do $B[C[A[j]]] \leftarrow A[j]$
        $C[A[j]] \leftarrow C[A[j]] - 1$



Analysis on Counting Sort

$\Theta(k)$ $\left\{ \begin{array}{l} \textbf{for } i \leftarrow 1 \textbf{ to } k \\ \quad \textbf{do } C[i] \leftarrow 0 \end{array} \right.$

$\Theta(n)$ $\left\{ \begin{array}{l} \textbf{for } j \leftarrow 1 \textbf{ to } n \\ \quad \textbf{do } C[A[j]] \leftarrow C[A[j]] + 1 \end{array} \right.$

$\Theta(k)$ $\left\{ \begin{array}{l} \textbf{for } i \leftarrow 2 \textbf{ to } k \\ \quad \textbf{do } C[i] \leftarrow C[i] + C[i-1] \end{array} \right.$

$\Theta(n)$ $\left\{ \begin{array}{l} \textbf{for } j \leftarrow n \textbf{ downto } 1 \\ \quad \textbf{do } B[C[A[j]]] \leftarrow \mathrm{A}[j] \\ \quad\quad C[A[j]] \leftarrow C[A[j]] - 1 \end{array} \right.$

$\Theta(n + k)$

- Running time: Θ (n+k)
- Extra Space: k
- K maybe large, for integers in 32-bit machine, k=2^{32-1}
- It is efficient when k= θ (n)

Comparison of sorting algorithms

Algorithm	Best Case	Worst Case	Average	Space usage	Stable (Ex.P216 4.51)
Insertion Sort	n	$n^2/2$	$\Theta(n^2)$	In place	Yes
Quick Sort	nlgn	$n^2/2$	Θ(nlgn)	lgn	No
Mergesort	nlgn/2	nlgn	Θ(nlgn)	n	Yes
Heapsort	2nlgn / nlgn(Accel.)	2nlgn / nlgn(Accel.)	Θ(nlgn)	In place	No
Selection Sort	$n^2/2$	$n^2/2$	$\Theta(n^2)$	In place	No
Bubble Sort	n	$n^2/2$	$\Theta(n^2)$	In place	Yes
Shell Sort	_	_	_	In place	No
Count Sort	n+k	n+k	Θ(n+k)	k	Yes