VE281

Data Structures and Algorithms

Recitation Class

Oct. 8 2018

VE281 TA Group

Asymptotic Algorithm Analysis

- Best, Worst, Average Case
  - Best case: least number of steps required, corresponding to the ideal input
  - Worst case: most number of steps required, corresponding to the most difficult input.
  - Average case: average number of steps required, given any input.

Specific input. Not specific size!

Consider Large Input Size

Asymptotic Algorithm Analysis

- Big-Oh, Big Omega, Theta
  - Big-Oh:
    - $T(n) = O(f(n))$ if only if it exists two positive constants c and n_0 such that $T(n) \leq cf(n)$ for all $n > n_0$.
    - If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty$, then $f(n)$ is $O(g(n))$.
  - Big Omega: $T(n) \geq cg(n)$
  - Theta: $c_1 g(n) \leq T(n) \leq c_2 g(n)$

$$n! \gg 2^n \gg n^{c+\varepsilon} \gg n^3 \gg n^{2+\varepsilon} \gg n^2 \gg n \log n \gg n^{1+\varepsilon} \gg n$$

$$\gg \sqrt{n} \gg \log n \gg \frac{\log n}{\log \log n} \gg \log \log n \gg \alpha(n) \gg 1$$

Asymptotic Algorithm Analysis

- Analyzing Time Complexity of Programs

```
sum = 0;
for(i = 1; i <= n; i *= 2)
    for(j = 1; j <= i; j++)
        sum++;
```

Asymptotic Algorithm Analysis

- Analyzing Time Complexity of Programs

```
sum = 0;
for(i = 1; i <= n; i *= 2)
    for(j = 1; j <= i; j++)
        sum++;
```

- The number of times that the statements `sum++ / j<=i / j++` occur is

$$1 + 2 + 4 + 8 + \cdots 2^{\log n} \approx 2n - 1$$

- The time complexity is $\Theta(n)$.

Sorting

- Sorting Basics
  - In place: requires $O(1)$ additional memory
  - Stability: whether the algorithm maintains the relative order of records with equal keys
  - Comparison sort:
    - Insertion sort $O(N^2)$, In place, Stable, Best case:$O(N)$
    - Selection sort $O(N^2)$, In place, Not stable
    - Bubble sort $O(N^2)$, In place, Stable
    - Merge sort $O(N \log N)$, Not in place, Stable
    - Quick sort $O(N \log N)$, Weakly in place, Not stable
  - Non-comparison sort
    - counting sort, bucket sort, radix sort

Sorting

- Master Method: $\boldsymbol{T(n) \le aT\left(\frac{n}{b}\right) + O(n^d)}$

$$\boldsymbol{T(n) = \begin{cases} O\left(n^d \log n\right) & if\ a = b^d \\ O\left(n^d\right) & if\ a < b^d \\ O\left(n^{\log_b a}\right) & if\ a > b^d \end{cases}}$$

$a = 2, b = 2, d = 1$

- Merge Sort (divide-and-conquer)

```
void mergesort(int *a, int left, int
  right) {
    if (left >= right) return;
    int mid = (left+right)/2;
    mergesort(a, left, mid);
    mergesort(a, mid+1, right);
    merge(a, left, mid, right);
}
```

Sorting

- Quick Sort
  - Choose an array element as **pivot**.
  - Put all elements < pivot to the left of pivot.
  - Put all elements ≥ pivot to the right of pivot.
  - Move pivot to its correct place in the array.
  - Sort left and right subarrays recursively
  (not including pivot).
- Pivot: Randomly pick
- Partition: In-place/Array
- Average running time: $O(n \log n)$
- Worst running time: $O(N^2)$

Sorting

- Counting Sort $O(N + k)$

1. Allocate an array **`C[k+1]`**.
2. Scan array A. For `i`**`=1`** `to` **`N`**, increment **`C[A[i]]`**.
3. For **`i=1`** to **`k`**, **`C[i]=C[i-1]+C[i]`**

   `C[i]` now contains number of items less than or equal to **`i`**.
4. For **`i=N`** down to **`1`**, put **`A[i]`** in new position **`C[A[i]]`** and decrement **`C[A[i]]`**.

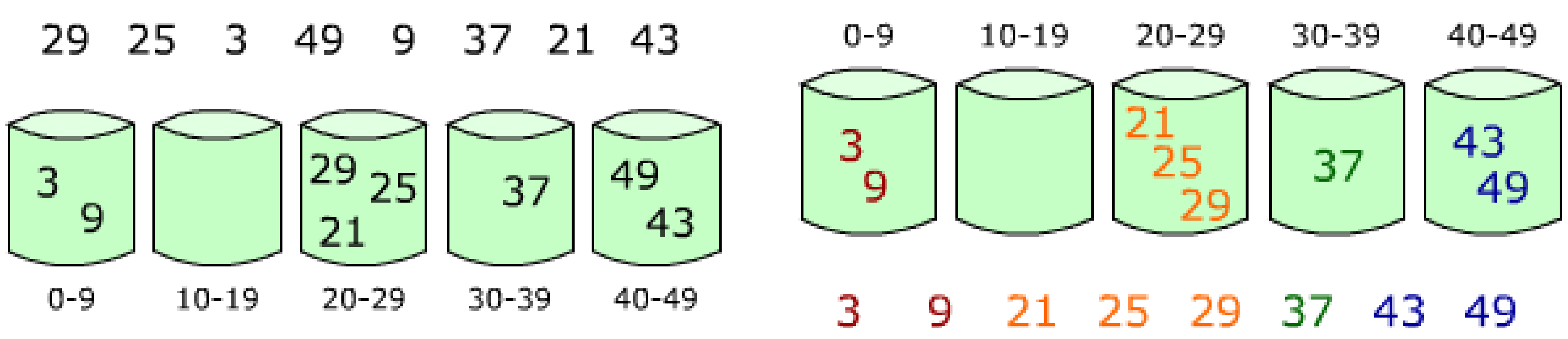
`A[i]` is put in correct position

since **`C[A[i]]`** items less than or equal to **`A[i]`**

Sorting

- Bucket Sort $O(N)$

1. Set up an array of initially empty "buckets".
2. Scatter: Go over the original array, putting each object in its bucket.
3. Sort each non-empty bucket <u>by a comparison sort</u>.
4. Gather: Visit the buckets in order and put all elements back into the original array.



Sorting

- Radix Sort $O(kN)$.

Given an array of integers, from the least significant bit (LSB) to the most significant bit (MSB), repeatedly do **stable** bucket sort according to the current bit

Sort 815, 906, 127, 913, 098, 632, 278.

0	1	2	3	4	5	6	7	8	9
		632	913		815	906	127	098 278	

0	1	2	3	4	5	6	7	8	9
906	913 815	127	632				278		098

0	1	2	3	4	5	6	7	8	9
098	127	278				632		815	906 913