Counting Sort, Radix Sort, Bucket Sort

CS 4231, Fall 2017 Mihalis Yannakakis

Counting Sort

- Restricted domain: D={1,…,k}
- Idea: Count how many input elements for each i in D

Example: Input A = [1, 2, 1, 3, 2, 5, 3, 2, 5]

Counts: C(1)=2, C(2)=3, C(3)=2, C(4)=0, C(5)=2

Output: B = [1, 1, 2, 2, 2, 3, 3, 5, 5]

$$\underbrace{\boxed{\;1\;1\;}}_{count(1)}\underbrace{\boxed{\;2\;\;2\;\;2\;}}_{count(2)}\boxed{\qquad\qquad\qquad\qquad}$$

Simple Counting Sort

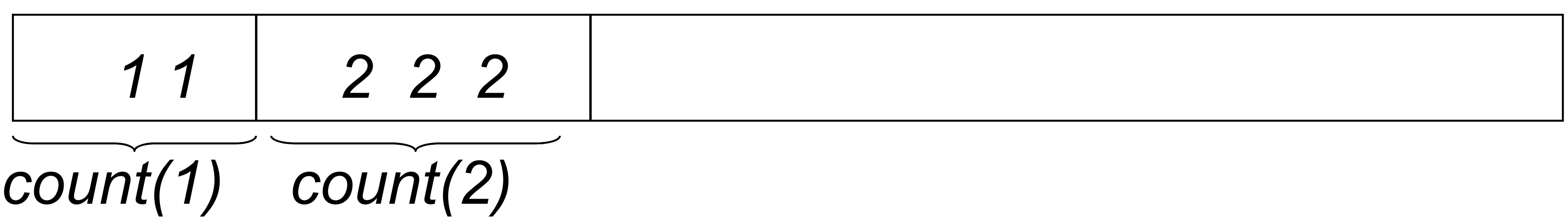
for i=1 **to** k **do** C[i]=0 (Initialize Count array)

for j=1 **to** n **do** C[A[j]]++ (Compute Counts)

m=1

for i=1 **to** k **do** (Place the elements in output array B)

  for j=1 **to** C[i] **do** { B[m]=i; m++}



- Complexity: O(n+k)

Sorting an Array of Records

Input: Array A of records, each with key in D={1,…,k}

Output: Array B of same records sorted by key

Bucket (Bin) Sort

Idea: A bucket (queue) Q[i] for each i =1,…,k

- Place each record A[j] in the appropriate bucket
- Empty in order the contents of the buckets in the output array B

Complexity: Θ(k+n|record|) if we copy whole records

Θ(k+n) if only indices of (pointers to) the records

Counting Sort of Array of Records

- Idea: Besides counts, compute prefix counts: how many elements have value $\leq$ i for each i=1,…,k

Example: Input A = [1, 2, 1, 3, 2, 5, 3, 2, 5]

Counts: C(1)=2, C(2)=3, C(3)=2, C(4)=0, C(5)=2

Prefix Counts: C(≤1)=2, C(≤2)=5, C(≤3)=7, C(≤5)=9

$\Rightarrow$ last 1 in position 2, last 2 in position 5, …

Output: B = [1, 1, 2, 2, 2, 3, 3, 5, 5]

Counting Sort

for i=1 **to** k **do** C[i]=0 (Initialize Count array)

for j=1 **to** n **do** C[A[j].key]++ (Compute Counts)

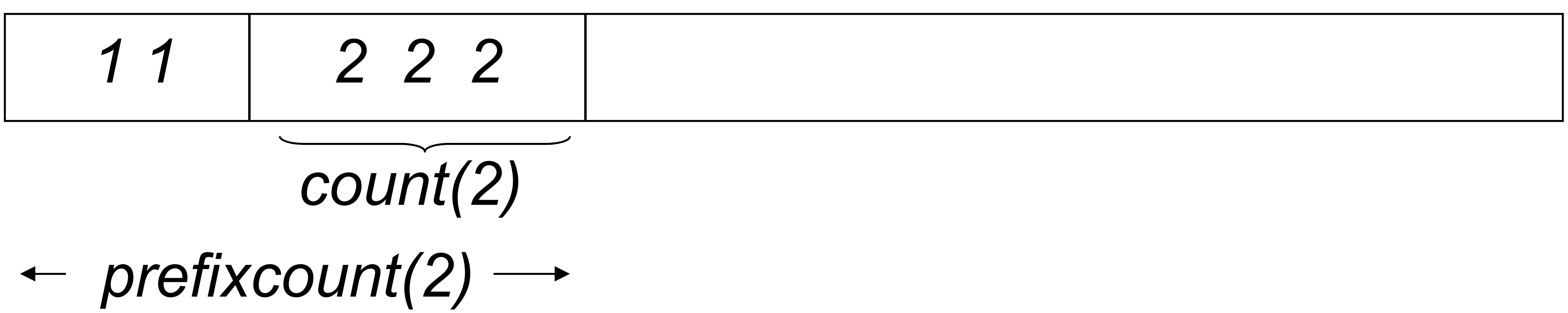
for i=2 **to** k **do** C[i]=C[i]+C[i-1]

(Compute prefix counts)

for j=n **downto** 1 **do**

{ B[C[A[j].key]]=A[j]; C[A[j].key]-- }

(Place the elements in output array B)



Complexity of Counting Sort

for i=1 **to** k **do** C[i]=0 — k

for j=1 **to** n **do** C[A[j].key]++ — n

for i=2 **to** k **do** C[i]=C[i]+C[i-1] — k

for j=n **downto** 1 **do**

{ B[C[A[j].key]]=A[j]; C[A[j].key]-- } — n|record|

Total: Θ(k+n|record|)

To avoid copying records, { B[C[A[j].key]]=j; C[A[j].key]-- }

Then output = array B of indices of the records in sorted order

Complexity Θ(k+n)

Bucket Sort and Counting Sort are stable

- Stable: Preserve the order among equal input elements

Input: A = [1, 2, 1, 3, 2, 5, 3, 2, 5]

Output: B = [1, 1, 2, 2, 2, 3, 3, 5, 5]

Radix Sorting

- Input elements: d-digit numbers, each digit takes k possible values
- More generally: d-vectors, where i-th component takes k_i values

  Example: Dates

Compare components (digits) of keys instead of the whole keys

Assumptions: 1. Can get i-th component of a key in cnst time

2. Can index (build arrays) on the components

Key Idea: Sort digit by digit, from least to most significant digit, using a stable sort

Radix Sort example

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

Correctness of Radix Sort

- After sorting on i-th least significant digit, elements are sorted according to suffix of last i digits.

  Proof: By induction.

  - If two numbers differ in i-th digit then ok.

  - If two numbers have equal i-th digits then stable digit sorting $\Rightarrow$ the ordering of last i-1 digits is maintained

Complexity of Radix Sorting

Time $\Theta(d(n+k))$

- If d=constant and k=O(n), then $\Theta(n)$

- Example application

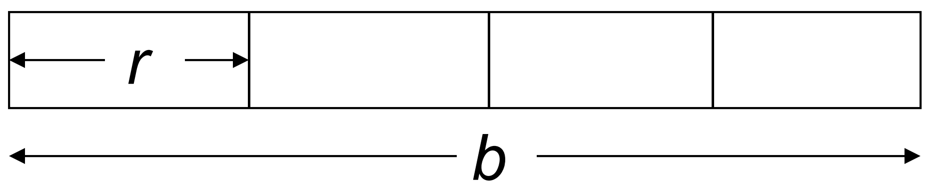
n integers in range 0 … $n^d - 1$, d constant

can be sorted in O(n) time

Write numbers $x < n^d$ in base n: $a_{d-1}a_{d-2}\cdots a_1 a_0$

$$x = a_{d-1}n^{d-1} + a_{d-2}n^{d-2} + \cdots + a_0$$
$$0 \le a_i \le n-1$$

- For n b-bit numbers, $n \le 2^b$
  can partition into blocks of r bits, $r \le b$
  $\Rightarrow$ Can sort in time $\Theta((b/r)(n+2^r))$



digits (passes) $d = b/r$, each digit base- 2^r

Optimal: $n \approx 2^r \Rightarrow r \approx \log n$

Time: $\Theta(bn/\log n)$

Numbers, Strings of varying length

- Strings: Lexicographic order

  alpha < alphabetical

- Fact: Can sort in time O(total length + k)

  (where each digit/letter takes k values)

  total length = sum of lengths of strings/numbers

(HW Exercise)

Bucket Sorting of random numbers

- Input: Real numbers drawn uniformly, independently from interval [0,1) (Generalizes to any interval)

- Algorithm: Partition interval [0,1) into n subintervals of length 1/n: [0,1/n), [1/n,2/n), …, [(n-1)/n, 1)
- One bucket B[i] for each subinterval [i/n, (i+1)/n), i=0,…,n-1
- Put every item *A*[*j*] in the corresponding bucket $B[\lfloor nA[j] \rfloor]$
- Sort the items in each bucket using e.g. insertion sort
- Concatenate the buckets

Analysis

- Let n_i = # items in bucket i (a random variable)
- Time = $\Theta(n) + \sum_i \Theta(n_i^2)$
- Expected time $T(n) = \Theta(n) + \sum_i \Theta(E[n_i^2])$
- All n_i have the same distributions $\Rightarrow$ same $E[n_i^2]$
- Claim: $E[n_i^2] = 2 - \frac{1}{n}$

$\Rightarrow \quad T(n) = \Theta(n)$

Analysis ctd.

- Proof of claim $E[n_i^2] = 2 - \frac{1}{n}$
- Let Z_j= indicator variable($A[j]$ falls in bucket i)

$$n_i = \sum_j Z_j \quad \Rightarrow \quad n_i^2 = (\sum_j Z_j)^2 = \sum_j Z_j^2 + 2\sum_{j \neq k} Z_j Z_k$$

$$E[n_i^2] = \sum_j E[Z_j^2] + 2\sum_{j \neq k} E[Z_j Z_k]$$

$$E[Z_j^2] = \Pr(Z_j^2 = 1) = \Pr(A[j] \text{ falls in } B[i]) = 1/n$$

$$E[Z_j Z_k] = \Pr(Z_j Z_k = 1) = \Pr(A[j], A[k] \text{ both fall in } B[i]) = 1/n^2$$

$$\Rightarrow \quad E[n_i^2] \; = \; 1 + 2\frac{n(n-1)}{2}\frac{1}{n^2} \; = \; 1 + \frac{n-1}{n} \; = \; 2 - \frac{1}{n}$$