G205 Fundamentals of Computer Engineering

CLASS 7, Wed. Sept. 29 2004

Stefano Basagni

Fall 2004

M-W, 1:30pm-3:10pm

Sorting in Linear Time

- We cannot go faster than $\Omega(n)$
- Must be a non-comparison sorting
- Works when assumptions on the number to be sorted are made
  - Counting sort → numbers in $\{0,1,\ldots,k\}$
  - Radix sort → numbers with a constant number of digits
  - Bucket sort → numbers drawn from a uniform distribution

Counting Sort, 1

- Numbers are integers in {0,1,…,k}
- INPUT: A[1…n], A[j]$\in${0,1,…,k} for all j=1,2,…,n. Array A and values n and k are given as parameters
- OUTPUT: B[1…n], sorted. B is assumed to be already allocated and is given as a parameter
- Auxiliary storage: C[0…k]

Counting Sort, 2

```
Counting-Sort(A,B,n,k)
  for i=0 to k do C[i] = 0
  for j=1 to n do C[A[ j ]]=C[A[j]]+1
  for i=1 to k do C[i]=C[i]+C[i-1]
  for j=n downto 1 do
    B[C[A[j]]]=A[j]
    C[A[j]]=C[A[j]]-1
```

Counting Sort, Example

- INPUT: A = $2_1, 5_1, 3_1, 0_1, 2_2, 3_2, 0_2, 3_3$
- OUTPUT: B = $0_1, 0_2, 2_1, 2_2, 3_1, 3_2, 3_3, 5_1$
- Counting-Sort is STABLE: keys with same value appear in same order in output as they did in input (because of how the last loop works)
- Analysis: $\Theta(n+k)$, which is $\Theta(n)$ if k is in O(n)

Radix Sort

- Key idea: Sort least significant digit of each number first
- To sort d digits:

```
Radix-Sort(A,d)
  for i = 1 to d do
    use a stable sorting to sort array A on
    digit i
```

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

Radix Sort: Correctness

- Induction on number of passes (i in pseudo-code)
- Assume digits 1, 2,…, i-1 are sorted
- Show that a stable sort on digit i leaves digits 1, 2, …, i sorted
  - If two digits in position i are different ordering by position i is correct (other digits are irrelevant)
  - If the digits are the same, numbers are already in the right order (ind. hyp.)

Radix Sort, Analysis

- Use Counting Sort as stable sorting
- $\Theta(n+k)$ per pass
- d passes
- $\Theta(d(n+k))$ total
- If k is in O(n) the $T_{RS}(n)$ is in $\Theta(dn)$
- When d is $\Theta(1)$ Radix Sort is linear time

How to break a number into digits

- n b-bits numbers
- Break into r-bits digits, have d=ceil(b/r)
- Use Counting Sort $k = 2^r - 1$
- $T_{RS}(n)$ is in $\Theta((b/r)(n+2^r))$
- Exercise: Choose r and compare Radix Sort and Merge-Sort

Searching

- The Selection Problem
  - INPUT: A set A of n (distinct) numbers and a number i, $0 \leq i \leq n$
  - OUTPUT: The element i in A that is larger than exactly i-1 other elements of A
- The element i is called the i-th order statistics of A
- The first order statistics is the minimum (i=1)
- The n-th is the maximum (i=n)
- Solvable in O(n log n)

Minimum or Maximum

```
Minimum(A,n)
  min = A[1]
  for i = 2 to n do
    if min > A[i] then min = A[i]
  return min
```

- n-1 comparisons, $T_M(n) \in O(n)$
- n-1 comparisons are necessary (tournament) $\rightarrow T_M(n) \in \Omega(n)$
- Minimum is OPTIMAL

Minimum AND Maximum, 1

```
Min-Max(A,n,min,max)
if n mod 2 = 0
  then  max=MAX(A[1],A[2]) // one comparison
        min=MIN(A[1],A[2])  // one comparison
        k=3
  else  max=min=A[1]
        k=2
  for i = k to n-1 step 2 do    // floor(n/2) iter
```

Minimum AND Maximum, 2

```
if A[i]>A[i+1]                                  // 1 co
  then
    if max<A[i] then max=A[i];                  // 1 co
    if min>A[i+1] then min=A[i+1];              // 1 co
  else
    if max<A[i+1] then max=A[i+1];              // 1 co
    if min>A[i] then min=A[i];                  // 1 co
```

Min-Max Analysis

- n odd: 3*ceil(n/2) comparisons
- n even: 3((n-2)/2)+1=(3n/2)-2
- At most 3*ceil(n/2) < 2n-2 comparisons
- Both are asymptotically in $\Theta(n)$

Searching for a Given Element

- Unsorted arrays, worst-case $\Theta(n)$
- Sorted arrays, binary search
- Input: A sorted array A, a value v and a range [low…high] in A to search for v
- Output: i such that v=A[i] or NIL is v is not found in A between low and high
- Initial call: A, v, 1, n

Iterative Binary Search

```
ITERATIVE-BINARY-SEARCH(A, v, low, high)
  while low ≤ high do
    mid=(low+high)/2
    if v = A[mid] then return mid
    if v > A[mid]
      then low=mid+1
      else high=mid-1
return NIL
```

Recursive Binary Search

```
REC-BSEARCH(A, v, low, high)
  if low > high then return NIL
  mid=(low+high)/2
  if v = A[mid] then return mid
  if v > A[mid]
    then return REC-BSEARCH(A,v,mid+1,high)
    else return REC-BSEARCH(A,v,low,mid-1)
```

Binary Search Analysis

- Based on the comparison on v with A's middle element the search continues halved
- The recurrence for the procedures is:
  - $T(n)=\Theta(1)$ $\quad$ $n=1$
  - $T(n)=T(n/2)+\Theta(1)$ $\quad$ $n>1$
- Solution: $T(n)$ in $\Theta(\log n)$

Assignments

- Textbook, pages 165—173, 183—185
- Updated information on the class web page:

  www.ece.neu.edu/courses/eceg205/2004fa