

## Sorting, Part II

CSE 130: Introduction to C Programming  
Spring 2005

1

## Mergesort

- Mergesort is a recursive sorting algorithm
- It works by recursively mergesorting halves of the array
- Mergesort is more time-efficient than bubblesort, etc.
  - It runs in  $n \log n$  time

2

## Mergesort Algorithm

- If the list size is  $> 1$ :
  1. Divide list in half
  2. Mergesort first half
  3. Mergesort second half
  4. Merge sorted halves together

3

## Merging Two Lists

- Set a variable to point to the next unused element of each list
- While both lists have unused elements:
  - Add the smaller unused element to the merge list, and increment that list's index
- When one list is empty, add the second list's elements to the merge list

4

## Merging Example

- List 1: 1, 3, 4      List 2: 2, 5, 6  
Merge list: <empty>
- List 1: 1, 3, 4      List 2: 2, 5, 6  
Merge list: 1
- List 1: 1, 3, 4      List 2: 2, 5, 6  
Merge list: 1, 2
- List 1: 1, 3, 4      List 2: 2, 5, 6  
Merge list: 1, 2, 3...

5

## Mergesort Code

```
void mergesort (int list[], int pos1, int pos2)
{
    if (pos2 > pos1)
    {
        int mid = (pos2 + pos1)/2;
        mergesort(list, pos1, mid);
        mergesort(list, mid+1, pos2);
        merge(list, pos1, mid, mid+1, pos2);
    }
}
```

6

## Radix Sort

- Method: Form groups (based on digits in the same place), then combine those groups
  - i.e., all items with 3 in the tens place
- This requires  $d$  iterations, where  $d$  is the number of digits in the largest element
- Worst-case running time:  $O(dn)$

7

## Pseudocode

1. for ( $j = d$  down to 1)
  1. Initialize 10 groups to empty
  2. for ( $i = 0$  through  $N-1$ )
    1. Place  $A[i]$  at the end of group  $K$
    2. Increment  $K$ th counter
  3. Replace  $A$  with group 0 + group 1+ etc.

8

## An Illustration

- 0123, 2154, 0222, 0004, 0283, 1560, 1061, 2150
- 1560, 2150, 1061, 0222, 0123, 0283, 2154, 0004
- 0004, 0222, 0123, 2150, 2154, 1560, 1061, 0283
- 0004, 1061, 0123, 2150, 2154, 0222, 0283, 1560
- 0004, 0123, 0222, 0283, 1061, 1560, 2150, 2154

9

## Counting (Bucket) Sort

- Like radix sort, counting sort doesn't require any direct comparisons
- Counting sort counts the number of times each element occurs, then generates a sorted list with the proper number of each element
- Counting sort destroys the original list
- Counting sort runs in linear time

10

## Buckets

- Counting sort requires a set of *buckets*
  - Normally, we use an array for this
- Each bucket corresponds to a particular value in the data range
  - i.e., 100 buckets for data in the range 1-100
- Counting sort starts by counting the number of values that belong in each bucket

11

## Filling Buckets

- Unsorted data: 3, 2, 4, 1, 5, 1 (range: 1-5)
- Bucket 1: 2
- Bucket 2: 1
- Bucket 3: 1
- Bucket 4: 1
- Bucket 5: 1

12

## Creating the New List

- Each bucket has a *number* and a *value*
  - *value* is the # of times *number* occurs in the data set
- For each bucket:
  - for 1 through *value*:
    - add *number* to the sorted data set

13

## Creating the New List

- Bucket 1: 2
- Bucket 2: 1
- Bucket 3: 1
- Bucket 4: 1
- Bucket 5: 1
- Resulting data set: 1, 1, 2, 3, 4, 5

14