School of Computer Science

COMP SCI 1103/2103 Algorithm Design & Data Structure

Distribution Sort

Sorting Algorithms

- Selection Sort
  - Complexity
    - Worst and average-case $O(n^2)$
- Insertion Sort (stable)
  - Complexity
    - Worst and average-case $O(n^2)$
- Bubble Sort (stable)
  - Complexity
    - Worst and average-case $O(n^2)$
- Quicksort
  - Complexity
    - Worst-case $O(n^2)$
    - Average-case $O(n \log n)$
- Merge Sort (stable)
  - Complexity
    - Worst and average-case $O(n \log n)$

Comparison Sort

- Most of what we've been looking at have been comparison sorts
- The fundamental engine of the sort is comparing two values.
- Comparison sorts include:
  - quicksort
  - merge sort
  - bubble sort
  - insertion sort
  - selection sort
  - heapsort (we will cover it at the end of the semester)
- How good can the comparison sort be? $\Omega(n \log n)$

Distribution sort

- We also have another kind of sort: distribution sort!
  - Bucket sort
- Basic outline of bucket sort:
  - Set up some empty buckets
  - Go over your original list and scatter the objects into the buckets.
  - Sort each bucket
  - Visit the buckets in turn and gather the results

Bucket sort

- Map the items to buckets (a hash function)
  - The buckets make up a contiguous set - you can think of each bucket as holding a subset of the possible values.
  - Everything that you're sorting will go into one (and only one) of the buckets.
  - Your scatter operation is going to use some sort of fast calculation to work out which bucket things go into.
  - Each item takes O(1), whole scattering O(n)
- The gather operation is a simple one:
  - Walk the buckets, in order
  - Extract the sorted list from each bucket
  - Join them together in sequence

Special Case – Counting Sort

- Counting sort
  - Bucket size 1-m buckets
  - O(m+n)
    - m being the number of possibilities for the values
    - n size of the array to be sorted
  - Steps
    - Count the number of each value, i, in the *input* array and store it in an array of size m called *count:* count[i] stores the number of times 'i' appears in the *input* array. (histogram)
    - Update count to hold a cumulative sum: count[i+1] = count[i+1] + count[i]
    - Go through *input* array from end. Look at the value and get count[value] this is the index where this value should be placed in the *result* array. Subtract one from count[value]
  - We have to start at end in 3rd step if we want sort to be stable

Example Counting Sort

- Input: 2 5 6 6 2 3 4 10 3 6 7 8 (12 elements)
- Buckets 1, …, 10
- Count={0, 2, 2, 1, 1, 3, 1, 1, 0, 1}
- Cumulative sum={0,2,4,5,6,9,10,11,11,12}
- Sorted sequence: place 8 at position 11, 7 at position 10, 6 at position 9, ….

1	2	3	4	5	6	7	8	9	10	11	12
2	2	3	3	4	5	6	6	6	7	8	10

Bucket sort- Scattering and sorting

- The choice of the number of buckets and the way that we scatter makes a big difference.
  - Works best if items are evenly distributed across the buckets
- Also it is important what sort of sorting algorithm is used for sorting the elements inside each bucket.
- If we use one bucket, and insertion sort, what happens?
- For *n* values, uniformly distributed over a range, what happens if we use k buckets and then insertion sort?
  - Scatter the items: $O(n)$
  - Sort buckets: $k * O((n/k)^2) = O(n(n/k))$
    - $= O(n)$ if $k=cn$
  - Gather items: $O(n+k)$

Bucket Sort

- Distribution of data matters!
  - Take one bucket for a wide range that does not have many items.
  - Assign smaller ranges to buckets where the items are concentrated
- Worst-case performance for Bucket sort is $O(n^2)$
  - If number of buckets is not unreasonably huge!
- Average case complexity $O(n+k)$
  - which is $O(n)$ if $k=O(n)$

Summary on sorting

- Comparison sort:
  - Selection sort
  - Insertion sort
  - Bubble sort
  - Quicksort
  - Mergesort
- Distribution sort:
  - Bucket sort
    - Counting sort
- Comparison sorts have a lower bound of $\Omega(n \log n)$.
- Distribution sorts do better if we can come up with a proper way of scattering that isn't high complexity
- There are other sorting algorithms which you may see in later courses.

THE UNIVERSITY of ADELAIDE