More on Sorting

CSE 2011
Fall 2009

When to use which sorting algorithm?

- Large arrays: merge sort, quick sort.
- Small arrays: insertion sort, selection sort.
  - Recursion is expensive.
- Merge sort or quick sort in an average case?
  - Cost of comparing elements
  - Cost of moving/switching elements

Merge Sort or Quick Sort?

Merge sort
- Lowest number of comparisons among popular algorithms
- Lots of data movements/copying (merging)

Java
- Generic sort uses Comparator

$\Rightarrow$ comparison is expensive.
- Moving is cheap (uses "pointers" rather than copies of objects).

Quick sort
- More comparisons
- Fewer data movements

C++
- Copying large objects is expensive.
- Comparison is cheap (compiler does inline optimization).

Java
- Used for primitive types (inexpensive comparisons)

3

Lower Bound for Sorting

- Merge sort and heap sort (discussed later)
  - worst-case running time is O(N log N)
- Are there better algorithms? No.
- We need to prove that any sorting algorithm based on only comparisons takes Ω(N log N) comparisons in the worst case (worse-case input) to sort N elements.
- We will prove this after learning "Trees".

4

Linear Time Sorting (O(N))

CSE 2011
Fall 2009

5

Linear Time Sorting

- Can we do better (linear time algorithm) if the input has special structure (e.g., uniformly distributed, every number can be represented by d digits)? Yes.
- Counting sort, radix sort, bucket sort

6

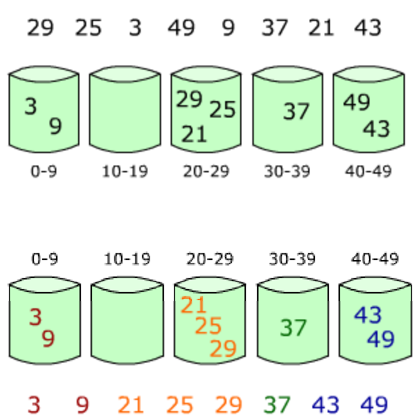
Bucket Sort

- Given an integer array A of size N,
- Assume that all elements in A have values < ***m***.
- Create an array B of size M. Each entry B[i] is considered a "bucket".
- For each element A[i], "throw" the element into bucket B[A[i]].
- Example: sort a list of students by GPA.
- Running time = ???
- What if ***m*** is large?

7

Bucket Sort (2)

- Each bucket contains more than one key values.
- After all inputs are thrown into the buckets, each bucket will be sorted (e.g., using insertion sort).
- Running time is still O(N).



8

Extendable Arrays

CSE 2011
Fall 2009

Extendable Array Implementation

When *push*() is called and an overflow occurs ($n = N$):

- Allocate a new array T of capacity $2N$
- Copy contents of the original array V into the first half of the new array T
- Set $V = T$
- Perform the insertion using new array V

- Note: when the number of elements in the list goes below a threshold (e.g., $N/4$), shrink the array by half the current size N of the array.

Time Analysis

- "*push*": inserting an element to be the last element of a list (or top of a stack)
- *add*(*e*) {

    Step 1: if overflow then extend the array;

    Step 2: "push" *e* to new array*;*

  }
- Proposition 1:

  Let *S* be a list implemented by means of an extendable array *V* as described before. The total time to perform a series of *n* "push" operations in *S*, starting from *S* being empty and *V* having size $N = 1$, is $O(n)$.

11

Time Analysis (2)

Step 2 takes O(n) (each "push" takes O(1))

Step 1:

- Allocate a new array *T* of capacity $2N$
- Copy $V[i]$ to $T[i]$ for i = 0, 1, …, $N–1$
- Set $V = T$

- If the array is extended *k* times, then $n = 2^k$
- The total number of copies is:

  $1 + 2 + 4 + 8 + \dots + 2^{k-1} = 2^k - 1 = n - 1 = O(n)$

- Step 1 + Step 2 = $O(n)$

12

Increment Strategies

- java.util.ArrayList and java.util.Vector use extendable arrays.
- *capacityIncrement* determines how the array grows:

  capacityIncrement = 0: array size doubles

  capacityIncrement = $c > 0$: array adds c new cells

- Proposition 2:

  If we create an initially empty java.util.Vector object with a fixed positive *capacityIncrement* value, then performing a series of n push operations on this vector takes $\Omega(n^2)$ time.
- $\Omega(n^2)$: takes at least time n^2

13

Increment Strategies (2)

Step 2 takes O(n) (each “push” takes O(1))

Step 1:

- Let a be the initial size of array V
- Let *capacityIncrement* = c
- If the array is extended k times then $n = a + ck$
- The total number of copies is:

  $(a) + (a+c) + (a+2c) + \ldots + (a+(k-1)c) =$

  $ak + c(1+2+\ldots+(k-1)) = ak + ck(k-1)/2 = \theta(k^2) = \theta(n^2)$
- We infer $\Omega(n^2)$ from $\theta(n^2)$

Which is the better increment strategy?

14

Next time …

- Lab test, Oct. 8, 17:30-19:00.
  - Be present in the lab (1004 or 1006) by 17:25.
- Reading week: Oct. 11 – 17.
  - “Succeed in Science” event, Oct. 15.
  - For more info, visit “science.yorku.ca/sis”.
- After the break: “Trees”.
- Midterm: Tuesday, Oct. 27.

15