CSE 326: Data Structures
A Sort of Detour

Henry Kautz

Winter Quarter 2002

Sorting by Comparison

1. Simple: SelectionSort, BubbleSort
2. Good worst case: MergeSort, HeapSort
3. Good average case: QuickSort
4. Can we do better?

Selection Sort Idea

- Are first 2 elements sorted? If not, swap.
- Are the first 3 elements sorted? If not, move the 3rd element to the left by series of swaps.
- Are the first 4 elements sorted? If not, move the 4th element to the left by series of swaps.
  - *etc.*

Selection Sort

```
procedure SelectionSort (Array[1..N])

For (i=2 to N) {
      j = i;
      while ( j > 0 && Array[j] < Array[j-1] ){
            swap( Array[j], Array[j-1] )
            j --; }
}
```

Suppose Array is initially sorted?

Suppose Array is reverse sorted?

Selection Sort

```
procedure SelectionSort (Array[1..N])

For (i=2 to N) {
      j = i;
      while ( j > 0 && Array[j] < Array[j-1] ){
            swap( Array[j], Array[j-1] )
            j --; }
}
```

Suppose Array is initially sorted? $O(n)$

Suppose Array is reverse sorted? $O(n^2)$

Bubble Sort Idea

Slightly rearranged version of selection sort:

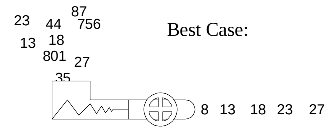
- Move smallest element in range 1,…,*n* to position 1 by a series of swaps
- Move smallest element in range 2,…,*n* to position 2 by a series of swaps
- Move smallest element in range 3,…,*n* to position 3 by a series of swaps
  - *etc.*

Why Selection (or Bubble) Sort is Slow

- *Inversion*: a pair (i,j) such that i<j but Array[i] > Array[j]
- Array of size N can have $\Theta(N^2)$ inversions
  - *average* number of inversions in a random set of elements is N(N-1)/4
- Selection/Bubble Sort only swaps adjacent elements
  - only removes 1 inversion!

7

HeapSort: sorting with a priority queue ADT (heap)



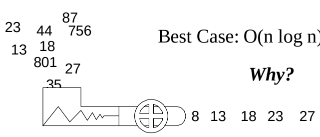
Worst Case:

Best Case:

Shove everything into a queue, take them out smallest to largest.

8

HeapSort: sorting with a priority queue ADT (heap)



Worst Case: O(n log n)

Best Case: O(n log n)

Why?

Shove everything into a queue, take them out smallest to largest.

9

MergeSort

```
MergeSort (Table [1..n])
Split Table in half
Recursively sort each half
Merge two halves together
```

Merging Cars by key [Aggressiveness of driver]. Most aggressive goes first.

Photo from http://www.nrma.com.au/inside-nrma/m-b-m/road-rage.html

```
Merge (T1[1..n],T2[1..n])
i1=1, i2=1
While i1<n, i2<n
      If T1[i1] < T2[i2]
            Next is T1[i1]
            i1++
      Else
            Next is T2[i2]
            i2++
      End If
End While
```

10

MergeSort Running Time

Any difference best / worse case?

$T(1) \le b$

$T(n) \le 2T(n/2) + cn$ for n>1

$T(n) \le 2T(n/2)+cn \quad \le \quad 2(2(T(n/4)+cn/2)+cn$

$= 4T(n/4) +cn +cn \quad \le \quad 4(2(T(n/8)+c(n/4))+cn+cn$

$= 8T(n/8)+cn+cn+cn$ — expand

$\le \; 2^kT(n/2^k)+kcn$ — inductive leap

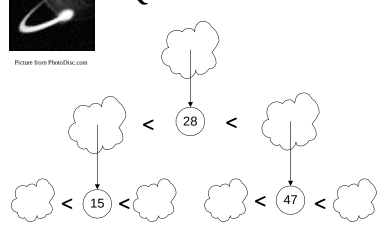
$\le \; nT(1) + cn \log n$ where $k = \log n$ — select value for k

$= O(n \log n)$ — simplify

11

Picture from PhotoDisc.com

QuickSort



Pick a "pivot". Divide into less-than & greater-than pivot. Sort each side recursively.

12

QuickSort Partition

Pick pivot:	7	2	8	3	5	9	6
Partition with cursors	7	2	8	3	5	9	6
2 goes to less-than	7	2	8	3	5	9	6

13

QuickSort Partition (cont’d)

6, 8 swap less/greater-than	7	2	6	3	5	9	8
3,5 less-than 9 greater-than	7	2	6	3	5	9	8
Partition done. Recursively sort each side.	7	2	6	3	5	9	8

14

Let’s go to the Races!

15

Analyzing QuickSort

- Picking pivot: constant time
- Partitioning: linear time
- Recursion: time for sorting left partition (say of size i) + time for right (size N-i-1)

  T(1) = b

  T(N) = T(i) + T(N-i-1) + cN

  where i is the number of elements smaller than the pivot

16

QuickSort Worst case

Pivot is always smallest element.

T(N) = T(i) + T(N-i-1) + cN

T(N) = T(N-1) + cN

= T(N-2) + c(N-1) + cN

= T(N-k) + $c\sum_{i=0}^{k-1}(N-i)$

= O(N²)

17

Dealing with Slow QuickSorts

- Randomly choose pivot
  - Good theoretically and practically, but call to random number generator can be expensive
- Pick pivot cleverly
  - “Median-of-3” rule takes Median(first, middle, last element elements). Also works well.

18

QuickSort
Best Case

Pivot is always middle element.

T(N) = T(i) + T(N-i-1) + cN

T(N) = 2T(N/2 - 1) + cN

$< 2T(N/2) + cN$

$< 4T(N/4) + c(2N/2 + N)$

$< 8T(N/8) + cN(1+1+1)$

$< kT(N/k) + cN\log(k) = O(N\log N)$

19

QuickSort
Average Case

- Assume all size partitions equally likely, with probability 1/N

$T(N) = T(i) + T(N-i-1) + cN$

average value of T(i) or T(N-i-1) is $(1/N)\sum_{j=0}^{N-1} T(j)$

$T(N) = \left((2/N)\sum_{j=0}^{N-1} T(j)\right) + cN$

$= O(N\log N)$

details: Weiss pg 278-279

20

Could We Do Better?*

- For any possible correct Sorting by Comparison algorithm…
  - What is lowest best case time?
  - What is lowest worst case time?

* (no. sorry.)

21

Best case time

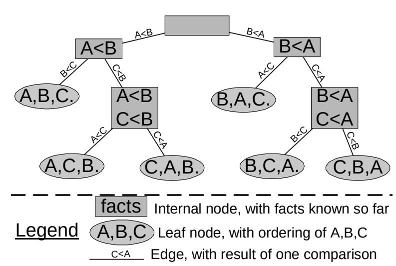
22

Worst case time

- How many comparisons does it take before we can be sure of the order?
- This is the minimum # of comparisons that any algorithm could do.

23

Decision tree to sort list A,B,C



24

Max depth of the decision tree

- How many permutations are there of N numbers?
- How many leaves does the tree have?
- What's the shallowest tree with a given number of leaves?
- What is therefore the worst running time (number of comparisons) by the best possible sorting algorithm?

25

Max depth of the decision tree

- How many permutations are there of N numbers?
  N!
- How many leaves does the tree have?
  N!
- What's the shallowest tree with a given number of leaves?
  log(N!)
- What is therefore the worst running time (number of comparisons) by the best possible sorting algorithm?
  log(N!)

26

Stirling's approximation

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

$$\log(n!) \approx \log\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right)$$

$$= \log(\sqrt{2\pi n}) + \log\left(\left(\frac{n}{e}\right)^n\right) = \Omega(n \log n)$$

27

Not enough RAM – External Sorting

- *E.g.:* Sort 10 billion numbers with 1 MB of RAM.
- Databases need to be very good at this

28

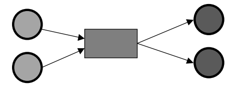
MergeSort Good for Something!

- Basis for most external sorting routines
- Can sort any number of records using a tiny amount of main memory
  - in extreme case, only need to keep 2 records in memory at any one time!

29

External MergeSort

- Split input into two tapes
- Each group of 1 records is sorted by definition, so merge groups of 1 to groups of 2, again split between two tapes
- Merge groups of 2 into groups of 4
- Repeat until data entirely sorted



log N passes

30

Better External MergeSort

- Suppose main memory can hold M records.
- Initially read in groups of M records and sort them (*e.g.* with QuickSort).
- Number of passes reduced to log(N/M)

31

Summary

- Sorting algorithms that only compare adjacent elements are $\Theta(N^2)$ worst case – but may be $\Theta(N)$ best case
- HeapSort and MergeSort - $\Theta(N \log N)$ both best and worst case
- QuickSort $\Theta(N^2)$ worst case but $\Theta(N \log N)$ best and average case
- Any comparison-based sorting algorithm is $\Omega(N \log N)$ worst case
- External sorting: MergeSort with $\Theta(\log N/M)$ passes

32