CSE 326: Data Structures
Lecture #16
Sorting Things Out

Bart Niswonger
Summer Quarter 2001

Unix Tutorial!!

- Tuesday, July 31st
  - 10:50am, Sieg 322

Printing worksheet

Shell
- different shell quotes : ' `
- scripting, #!
- alias
- variables / environment
- redirection, piping

Useful tools
- grep, egrep/grep -e
- sort
- cut
- file
- tr
- find, xargs
- diff, patch
- which, locate, whereis

Finding info
- Techniques
- Resources (ACM webpage, web, internal docs)

Process management

File management/permissions

Filesystem layout

Today's Outline

- Project
  - Rules of competition
- Sorting by comparison
  - Simple :
    - `SelectionSort`; `BubbleSort`; `InsertionSort`
  - Quick :
    - `QuickSort`
  - Good Worst Case :
    - `MergeSort`; `HeapSort`

Sorting: The Problem Space

General problem

Given a set of N *orderable* items, put them *in order*

Without (significant) loss of generality, assume:

- Items are integers
- Ordering is $\leq$

Most sorting problems map to the above in linear time.

Selection Sort

1. Find the smallest element, put it first
2. Find the next smallest element, put it second
3. Find the next smallest, put it next

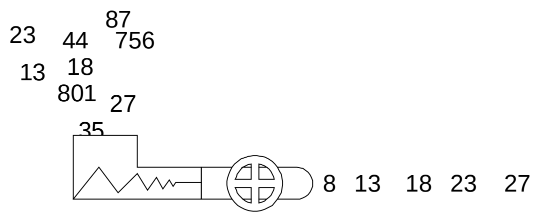
… etc.

Selection Sort

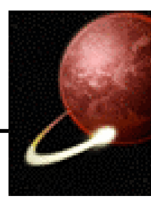
```
procedure SelectionSort (Array[1..N]

For i=1 to N-1
      Find the smallest entry in Array[i..N]
      Let j be the index of that entry
      Swap(Array[i],Array[j])
End For

While other people are coding QuickSort/MergeSort
      Twiddle thumbs
End While
```

HeapSort

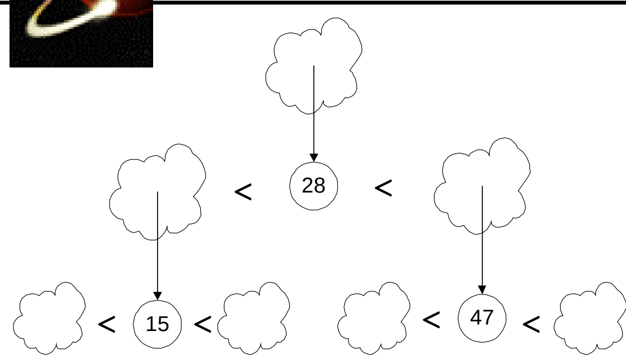
- Use a Priority Queue (Heap)



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



QuickSort



1. **Basic idea: Pick a *pivot.***
2. ***Partition* into less-than & greater-than pivot.**
3. **Sort each side *recursively*.**

QuickSort Partition

Pick pivot

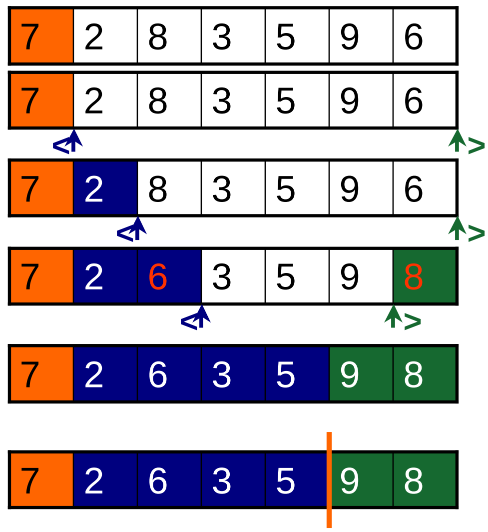
Partition with cursors

2 goes to less-than

6, 8 swap less/greater-than

3,5 less-than
9 greater-than

Partition done.
Recursively sort each side.



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)

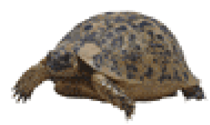
  $T(1) = b$

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

  where *i* is the number of elements smaller than the pivot

QuickSort : Worst Case

- What is the worst case?



Optimizing QuickSort

- Choosing the Pivot
  - Randomly choose pivot
    - Good theoretically and practically, but call to random number generator can be expensive
  - Pick pivot cleverly
    - “Median-of-3” rule takes element at Median(first value, last value). Works well in practice.
- Cutoff
  - Use simpler sorting technique below a certain problem size
    - *Weiss* suggests using insertion sort, with a cutoff limit of 5-20

QuickSort : Best Case

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

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

< 2T(N/2) + cN

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

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

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

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

MergeSort

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

MergeSort (Collection [1..n])
1. Split Collection in half
2. Recursively sort each half
3. **merge** two *sorted* halves together

```
merge (C1[1..n], C2[1..n])
i1=1, i2=1
while i1<n and i2<n
        if C1[i1] < C2[i2]
                Next is C1[i1]
                i1++
        else
                Next is C2[i2]
                i2++
        end If
end while
```

MergeSort Analysis

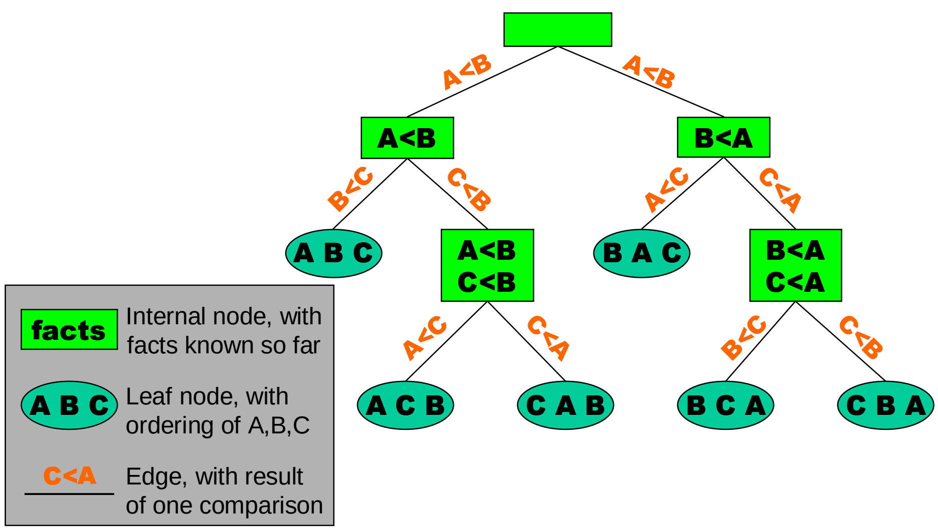
- Running Time
  - Worst case?
  - Best case?
  - Average case?
- Other considerations besides running time?

Is This The Best We Can Do?

- Sorting by Comparison
  - Only information available to us is the *set of N items* to be sorted
  - Only operation available to us is *pairwise comparison between 2 items*

What is the best running time we can possibly achieve?

Decision Tree Analysis



How deep is 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?

Lower Bound for log(n!)

Stirling's approximation: $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

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

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

Is This The Best We Can Do?

- Sorting by Comparison
  - Only information available to us is the *set of N items* to be sorted
  - Only operation available to us is *pairwise comparison between 2 items*

What happens if we relax these constraints?

BinSort (a.k.a. BucketSort)

Requires:

- Knowing the keys to be in {1, …, K}
- Having an array of size K

Works by:

Putting items into correct bin (cell) of array, based on key

BinSort example

K=5 list=(5,1,3,4,3,2,1,1,5,4,5)

Bins in array	
key = 1	1,1,1
key = 2	2
key = 3	3,3
key = 4	4,4
key = 5	5,5,5

Sorted list:
1,1,1,2,3,3,4,4,5,5,5

BinSort Pseudocode

```
procedure BinSort (List L,K)

LinkedList bins[1..K]
// Each element of array bins is linked list.
// Could also BinSort with array of arrays.

For Each number x in L
      bins[x].Append(x)
End For
For i = 1..K
      For Each number x in bins[i]
            Print x
      End For
End For
```

BinSort Running Time

- K is a constant
  - BinSort is linear time
- K is variable
  - Not simply linear time
- K is large (e.g. 2^{32})
  - Impractical

BinSort is “stable”

Definition: Stable Sorting Algorithm

Items in input with the same key end up in the same order as when they began.

- BinSort is stable
  - Important if keys have associated values
  - Critical for RadixSort

Mr. Radix

Herman Hollerith

Born February 29, 1860 - Died November 17, 1929

Art of Compiling Statistics; Apparatus for Compiling Statistics

Herman Hollerith invented and developed a punch-card tabulation machine system that revolutionized statistical computation.

Born in Buffalo, New York, the son of German immigrants, Hollerith enrolled in the City College of New York at age 15 and graduated from the Columbia School of Mines with distinction at the age of 19.

His first job was with the U.S. Census effort of 1880. Hollerith successively taught mechanical engineering at the Massachusetts Institute of Technology and worked for the U.S. Patent Office. Hollerith began working on the tabulating system during his days at MIT, filing for the first patent in 1884. He developed a hand-fed 'press' that sensed the holes in punched cards; a wire would pass through the holes into a cup of mercury beneath the card closing the electrical circuit. This process triggered mechanical counters and sorter bins and tabulated the appropriate data.

Hollerith's system-including punch, tabulator, and sorter-allowed the official 1890 population count to be tallied in six months, and in another two years all the census data was completed and defined; the cost was $5 million below the forecasts and saved more than two years' time. His later machines mechanized the card-feeding process, added numbers, and sorted cards, in addition to merely counting data.

In 1896 Hollerith founded the Tabulating Machine Company, forerunner of Computer Tabulating Recording Company (CTR). He served as a consulting engineer with CTR until retiring in 1921.

In 1924 CTR changed its name to IBM- the International Business Machines Corporation.

Source: National Institute of Standards and Technology (NIST) Virtual Museum - http://museum.nist.gov/panels/conveyor/hollerithbio.htm

RadixSort

- Radix = “The base of a number system” (Webster’s dictionary)
  - *alternate terminology: radix is number of bits needed to represent 0 to base-1; can say “base 8” or “radix 3”*
- Idea: BinSort on each digit, bottom up.

RadixSort – magic! It works.

- Input list:
  126, 328, 636, 341, 416, 131, 328
- BinSort on lower digit:
  341, 131, 126, 636, 416, 328, 328
- BinSort result on next-higher digit:
  416, 126, 328, 328, 131, 636, 341
- BinSort that result on highest digit:
  126, 131, 328, 328, 341, 416, 636

Not magic. It provably works.

- Keys
  - K-digit numbers
  - base B
- Claim: after i^{th} BinSort, least significant i digits are sorted.
  - e.g. B=10, i=3, keys are 1776 and 8234.
    8<u>234</u> comes before 1<u>776</u> for last 3 digits.

RadixSort
Proof by Induction

- Base case:
  - i=0. 0 digits are sorted (that wasn't hard!)
- Induction step
  - assume for i, prove for $i+1$.
  - consider two numbers: X, Y. Say X_i is i^{th} digit of X (from the right)
    - $X_{i+1} < Y_{i+1}$ then $i+1^{th}$ BinSort will put them in order
    - $X_{i+1} > Y_{i+1}$, same thing
    - $X_{i+1} = Y_{i+1}$, order depends on last i digits. Induction hypothesis says already sorted for these digits. (Careful about ensuring that your BinSort preserves order aka "stable"…)

What types can you RadixSort?

- Any type T that can be BinSorted
- Any type T that can be broken into parts A and B, such that:
  - You can reconstruct T from A and B
  - A can be RadixSorted
  - B can be RadixSorted
  - A is always more significant than B, in ordering

Example:

- 1-digit numbers can be BinSorted
- 2 to 5-digit numbers can be BinSorted without using too much memory
- 6-digit numbers, broken up into A=first 3 digits, B=last 3 digits.
  - A and B can reconstruct original 6-digits
  - A and B each RadixSortable as above
  - A more significant than B

RadixSorting Strings

- 1 Character can be BinSorted
- Break strings into characters
- Need to know length of biggest string (or calculate this on the fly).
- Null-pad shorter strings
- Running time:
  - N is number of strings
  - L is length of longest string
  - RadixSort takes O(N*L)

To Do

- Finish Project III (due Wednesday!)
- Finish reading chapter 7

Coming Up

- **More Algorithms!**
- Sorting
- Project III due (Wednesday)

- Unix Tutorial (Tuesday, tomorrow!)