CS180 - Sorting

Note Title 10/31/2011

Announcements

- Exam 1 is graded, look for mid-sem grades by Friday
- HW due Thursday
  - Next HW - due Tues after break
- Lab tomorrow

Exam 1 stats : Average 44.75
(Std Dev: 7.98)

55-60 : 2

53-54.5 : 2

51-52.5 : 4

49-50.5 : 1

47-48.5 : 4

45-46 : 4

40-42 : 5

30-39 : 4

under 30 : 2

Vectors versus lists

Q: What would operator [] look like in a list?

Takes an integer → returns value at that spot:
mylist[10]

In list, loop through → count up to index

Vectors versus lists (cont)

Running times:

	Vectors	Lists
operator []	O(1)	O(n)
find		
insert	O(n)	O(1)
erase/remove	O(n)	O(1)

Searching

What is linear search? $O(n)$

Go element by element + look for target.
(only makes sense if unordered)

Binary search? $O(\log n)$

Sorted list: look in middle,
rearse on left or right

$$B(n) = 1 + B\left(\frac{n}{2}\right)$$
$$= 1 + 1 + \cdots + B(1)$$

$\frac{n}{2^d} = 1$

$n = 2^d \Rightarrow d = \log_2 n$

Practical Considerations

Which is better?

Bin. Search needs O(1) - operator[] !

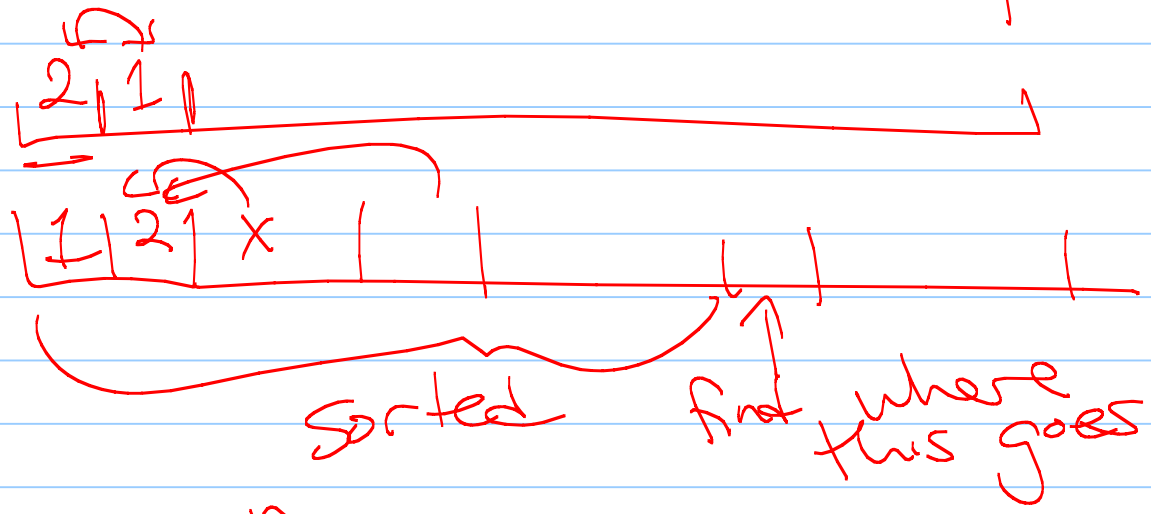
Sorting

Name some sorting algorithms.

- Bubble sort
- Quick sort
- ✗ Insertion sort
- Merge sort

Insertion Sort

maintain sorted sublist & insert next value in place



Time: $\sum_{i=1}^{n} O(i) = O(n^2)$

A be a list

for (i = 1 to n)

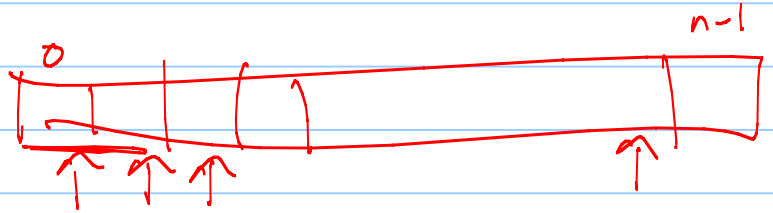
{ Find where A[i] goes in Sublist A[0..i-1] } binary or linear

Say j is result of ↑

insert A[i] in position j

O(i) operation each time, in either list or vector

Bubble Sort



for $j = n-2$ down to 1
  for $i = 0$ to ~~$n-2$~~ j
    compare $A[i]$ & $A[i-1]$
    swap if out of order

$O(n^2)$

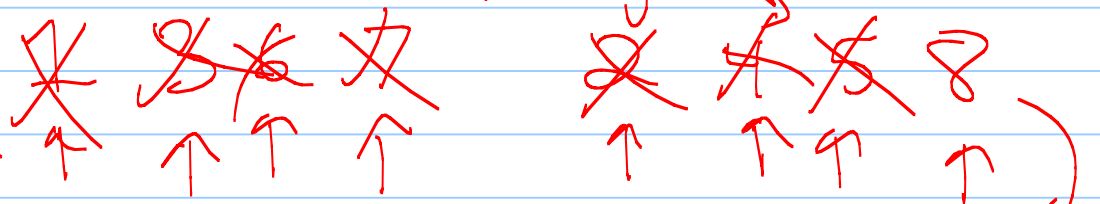
Merge Sort

Base case

7 3 1 6 | 2 5 4 8

recursively sort left & right halves

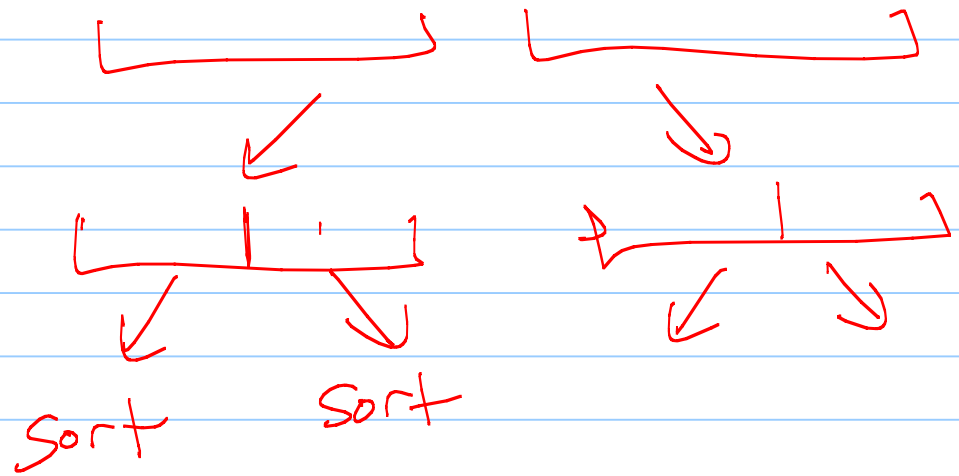
Merge:



new: 1 2 3 4 5 6 7 8

1 comparison + increment
↳ 1 more element in sorted list

Merge subroutine: $O(n)$ time



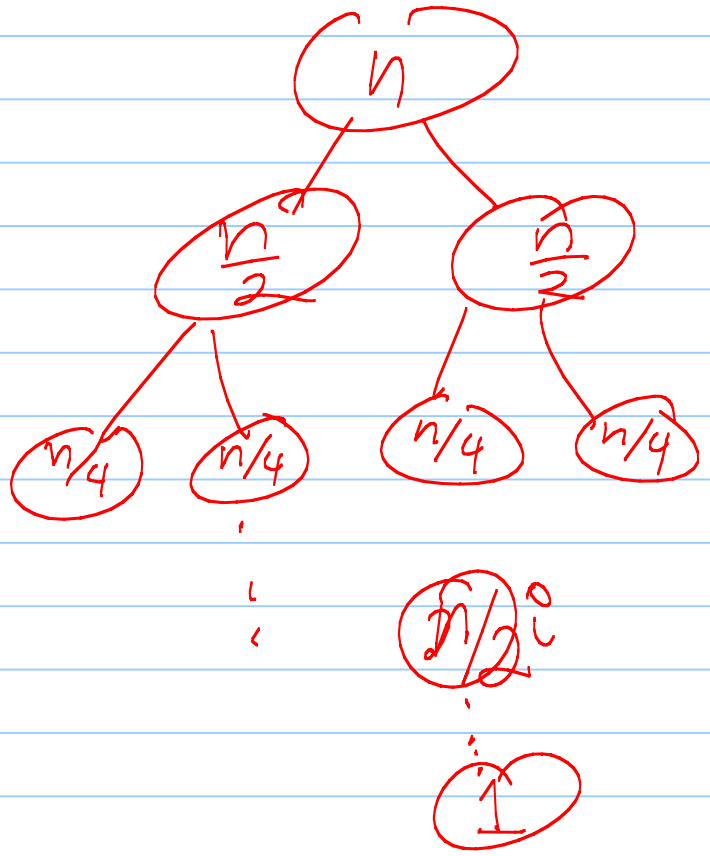
$$M(n) = 2M\left(\frac{n}{2}\right) + O(n)$$

$$M(1) = O(1)$$

$$M(n) = O(n \log n)$$

$$M(n) = 2M\left(\frac{n}{2}\right) + O(n)$$

seen



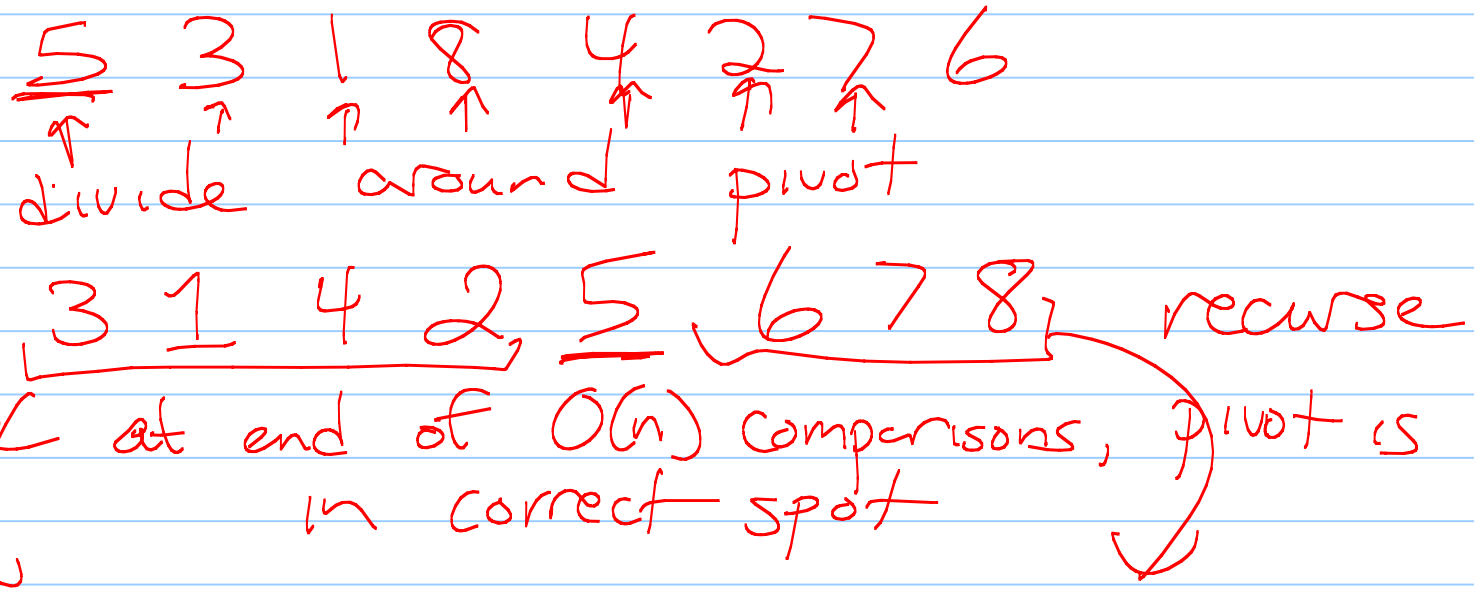
$$M(n) = \sum_{i=0}^{\log n} 2^i \cdot \frac{n}{2^i}$$

level i

2^i nodes

Quick Sort

Pick pivot element



Quicksort:

Worst case, choose an awful pivot (1 or 8) every time:

$O(n^2)$ worst

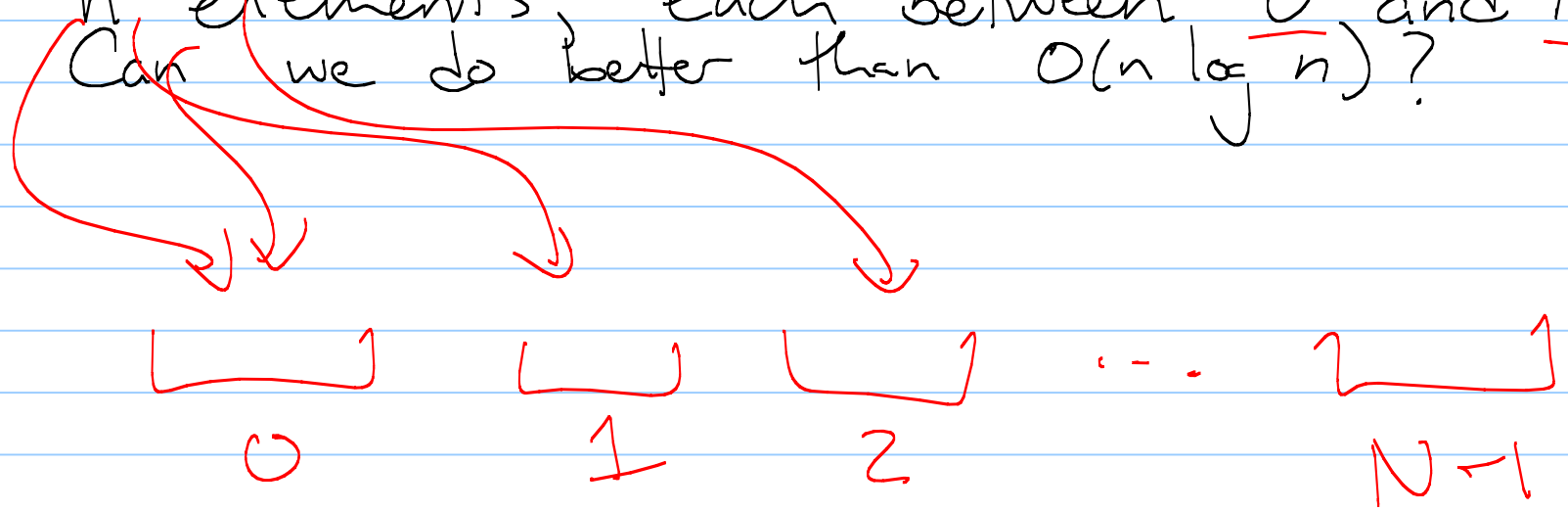
Expected running time:

(with "high probability")

$O(n \log n)$

Bucket Sort

n elements, each between 0 and N-1
Can we do better than $O(n \log n)$?



$O(n+N)$-time — just allocate N buckets + loop through list

Spps 10 things:

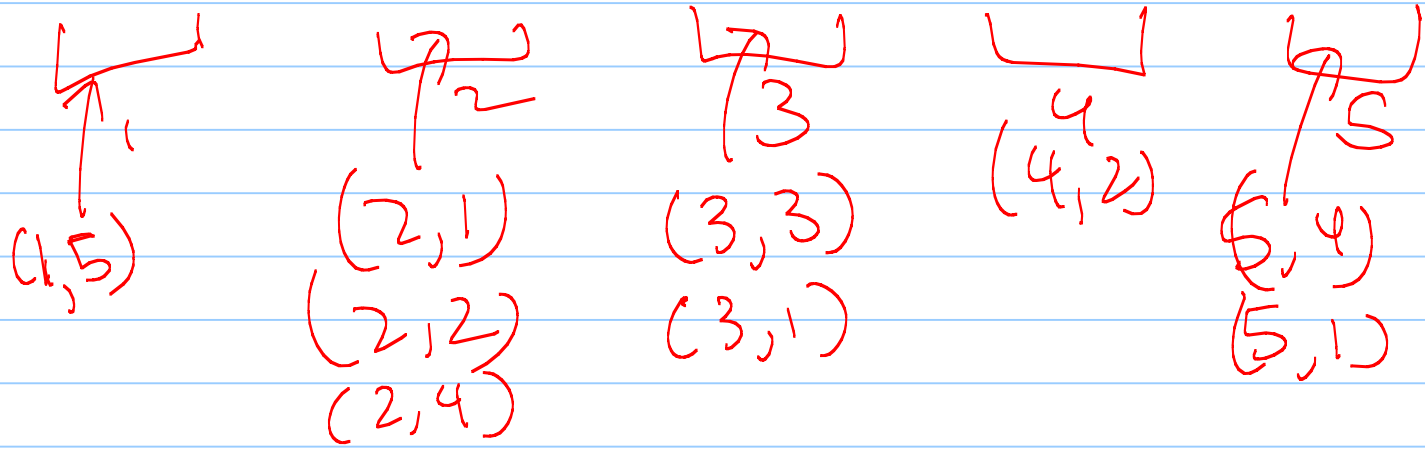
1 10,000 1,000,000, 7, 6,
2, −50, . . .

need over 1,000,000
buckets to sort 10 things

Radix Sort : for multiple-key sorting

Ex: (1,5), (2,1), (4,2), (3,3), (5,4), (3,1), (2,2), (5,1), (2,4)

Sort lexicographically: (use repeated bucket sorts)



Practicalities

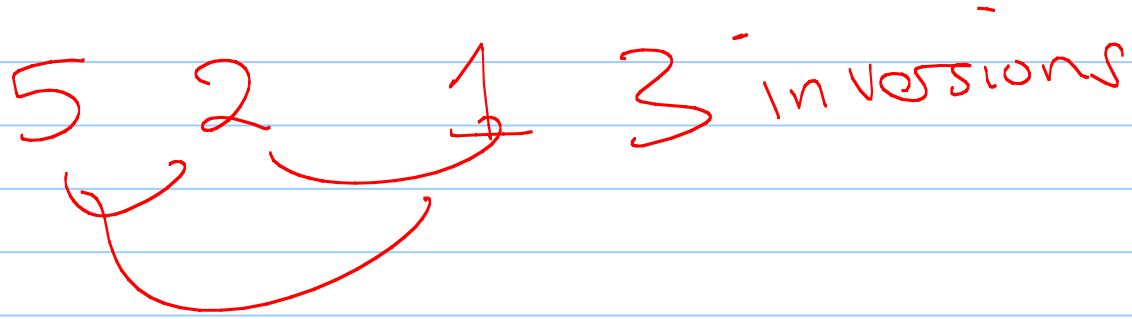
Experimentally, quicksort runs faster than merge on small inputs.

Why?

No need for allocating new array

More practicalities

- If implemented well, the running time of insertion sort is $O(m+n)$, where m = # of inversions (or out of order elements)



Conclusion: It depends!

- If the range of values is small, bucket sort (or radix sort) are faster. (characters! (strings))
- Quicksort despite worst case $O(n^2)$ is actually the one usually implemented.
- Also- can depend on linked vs. array based structure.