Math 182 Lecture 5

Chapter 3 Sorting Algorithms

What is sorting?

E.g. Alphatezing a list of names

Arranging cards in a hand

Sorting a bunch of numbers from lowest to highest

Goal: Write algorithms to do this on a computer

a_i's numbers ($\in \mathbb{N}$)

Input: We are given a sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.
Output: We want to output a permutation (reoreding) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

in this context the a_i's are called keys

§8.1 Insertion Sort

Idea: this is how a human might sort a deck of cards.

E.g. $A = \langle 3, 2, 1 \rangle$

INSERTION-SORT(A)

```
→ 1  for j = 2 to A.length
  2      key = A[j]
  3      // Insert A[j] into the sorted sequence A[1..j-1].
  4      i = j - 1
  5      while i > 0 and A[i] > key
  6          A[i+1] = A[i]
  7          i = i - 1
  8      A[i+1] = key
```

processes next key which needs to be inserted in growing list

finds correct location to insert key, moves everything to right of that spot one spot to the right

key gets inserted into correct spot.

Sps $A = \langle 3, 4, 2, 1 \rangle$

when $j = 2$ $\quad \langle \boxed{3}, 4, 2, 1 \rangle$

key to process

$\langle 3, 4, 2, 1 \rangle$

already in correct order, do nothing

when $j = 3$ $\quad \langle \boxed{3, 4,} 2, 1 \rangle$

key to process

$\langle 3, 4, 4, 1 \rangle$ key = 2 | while loop

$\langle 3, 3, 4, 1 \rangle$ key = 2

$\langle 2, 3, 4, 1 \rangle$ (line 8)

when $j = 4$ $\quad \langle \boxed{2, 3, 4,} 1 \rangle$ key = 1

key to process

$\langle 2, 3, 4, 4 \rangle$ | while loop

$\langle 2, 3, 3, 4 \rangle$

$\langle 2, 2, 3, 4 \rangle$

$i = 0$ $\quad \langle 1, 2, 3, 4 \rangle$ line 8

↑ sorted version of original A.

$j = 5$ $\quad \langle \boxed{1, 2, 3, 4} \rangle$

Correctness

Theorem 3.1.1. *At the end of* INSERTION-SORT(A), *the array A consists of the original elements from A, but in sorted order.*

Proof: (Loop Invariant) At the end of each time line 1 runs, the subarray $A[1..j-1]$ consists of the same elements originally in $A[1..j-1]$, but in sorted order.

(Initialization) $j=2$ $A[1..j-1] = A[1]$ a single element. This sorted and is the original element in that subarray $A[1]$.

(Maintenance) Sps line 1 has just run and value of j is $j=j_0$ where $2 \le j_0 \le A.length$ and we know LI is true $A[1..j_0-1]$ sorted $key = A[j_0]$. While loop (lines 5–8) inserts key in correct place so now $A[1..j_0]$ sorted and consists of original elements in $A[1..j_0]$. (See notes for details)

(Termination) The last time line 1 is run, LI is true and $j = A.length+1$ LI says $A[1..A.length]$ is sorted and $A[1..A.length]$ consists of elements originally in these spots. But $A = A[1..A.length]$ is entire array so done. ∎

$j = 2, \ldots, n+1$

$(n+1) - 2 + 1$

Running time

$n := A.length$

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into...
4      i = j - 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i - 1
8      A[i + 1] = key
```

cost:	times:
c_1	n
c_2	$n-1$
0	
c_3	$n-1$
c_4	$\sum_{j=2}^{n} t_j$ ←
c_5	$\sum_{j=2}^{n} (t_j - 1)$
c_6	$\sum_{j=2}^{n} (t_j - 1)$
c_7	$n-1$

If $2 \le j \le n$

what are the possibilities for number of times line 5 will run?

$t_j :=$ # times line 5 runs when for value of j.

$\rightarrow 1 \le t_j \le j$

$i = j-1, \ldots, 0$

$0, \ldots, j-1 \quad (j-1) - 0 + 1 = j$

Running Time:

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7(n-1)$$

Running time not clear bcs t_j's depend on how sorted A originally is.

Best case running time: If A already sorted then $t_j = 1$ for every j so

$\sum_{j=2}^{n} 1 = n-1$ $\sum_{j=2}^{n} (1-1) = 0$ so

$T(n) = an + b$ in this case so

Best-case running time is $T(n) = \Theta(n)$

Worst-case If A is in reverse order then $t_j = j$ for every j so

$\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$ $\sum_{j=2}^{n} (j-1) = \frac{n(n+1)}{2} - 1 - (n-1)$

$\Rightarrow$ quadratic.

$T(n) = an^2 + bn + c$ so worst-case running time is $\Theta(n^2)$.

<5,4,3,2,1>
<4,5,3,2,1>
<3,4,5,2,1>
<2,3,4,5,1>
<1,2,3,4,5>

Theorem 3.1.2. *The running time of* INSERTION-SORT *can be characterized as follows:*

(1) The best-case running time is $\Theta(n)$.
(2) The worst-case running time is $\Theta(n^2)$.
(3) The overall running time is $O(n^2)$ and $\Omega(n)$.

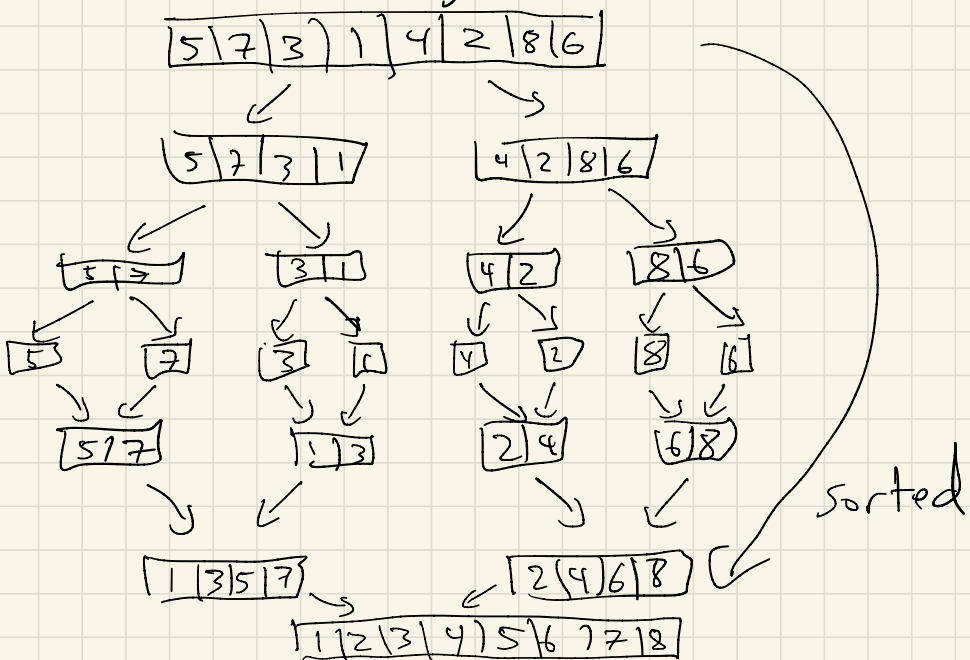
Remarks:

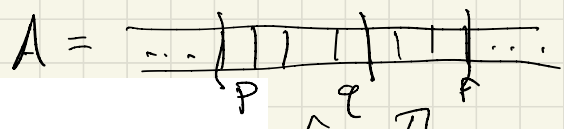
(1) Pretty fast for "small input"

(2) Insertion sort sorts in-place (only needed constant amount of additional memory)

(3) Insertion sort is stable
(equal keys maintain same relative order before and after).

§3.2 Merge Sort





Merge will merge these two subarrays.

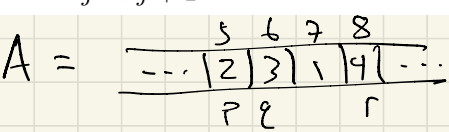
```
MERGE(A, p, q, r)
 1  n_1 = q − p + 1
 2  n_2 = r − q
 3  let L[1..n_1 + 1] and R[1..n_1 + 1] be new arrays
 4  for i = 1 to n_1
 5      L[i] = A[p + i − 1]
 6      // Copies A[p..q] into L[1..n_1]
 7  for j = 1 to n_1
 8      R[j] = A[q + j]
 9      // Copies A[q + 1..r] into R[1..n_1]
10  L[n_1 + 1] = ∞
11  R[n_2 + 1] = ∞
12  i = 1
13  j = 1
14  for k = p to r
15      if L[i] ≤ R[j]
16          A[k] = L[k]
17          i = i + 1
18      else A[k] = R[j]
19          j = j + 1
```

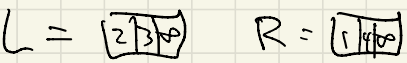
Copy A[p..q] and A[q+1..r] into new arrays L and R.

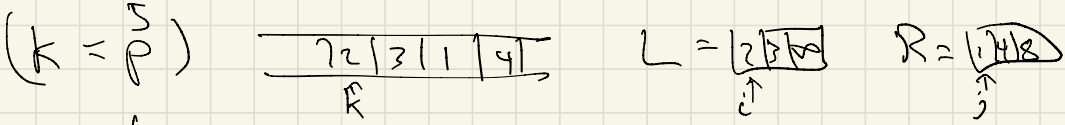
"sentinels"

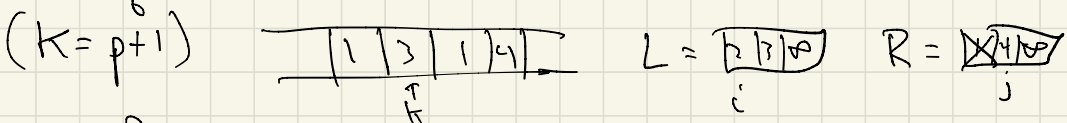
last elements of L and R signifying we are done w/ that array.

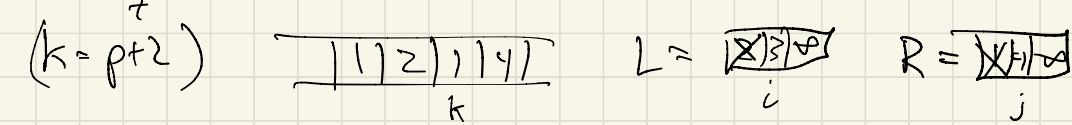
Merges L and R back into A[p..r].

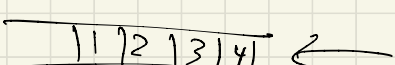




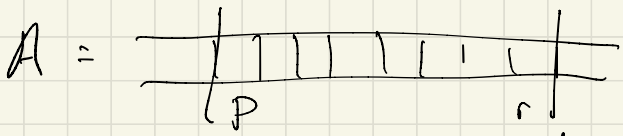








← merged version of [2|3] and [1|4]




```
MERGE-SORT(A, p, r)
1  if p < r                          <-- check if A[p..r] has length 1.
2      q = ⌊(p + r)/2⌋               q = p
3      MERGE-SORT(A, p, q)           r - q - 1
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

Merge(A, p, r)
~ will sort A[p..r]

If A[p..r] length > 1
something to do.

line 2 q roughly cuts A[p..r] in half
lines 3, 4 each half subarray gets sorted recursively
line 5 merges the now-sorted A[p..q] and A[q+1..r] into A[p..r]. Done ✓.

Correctness Proofs

Proposition 3.2.1. *Suppose A is an array and $1 \leq p \leq q < r \leq A.length$. If $A[p..q]$ and $A[q+1..r]$ are sorted, then after running* MERGE(A, p, q, r) *the subarray $A[p..r]$ will be sorted and consist of the original elements in $A[p..r]$.*

Proof:

(Loop Invariant) After each time line 12 is run, the subarray $A[p..k-1]$ contains the $k-p$ smallest elements of $L[1..n_1+1]$ and $R[1..n_2+1]$ in sorted order. And, $R[i]$, $L[j]$ are the smallest elements in their resp. arrays which have yet to be copied back into A.

(See lecture notes for details)

```
RLoop(n)
1  if n ≤ 1
2      return n
3  value = 0
4  for j = 1 to n
5      value = value + RLoop(j-1)
6  return value.
```

$\frac{\infty}{\infty}$

if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ then $f(n) = o(g(n))$

$f \ll g$

$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \lim_{x\to\infty} \frac{f'(x)}{g'(x)} = \dots$

RLoop(4)

Need to know RLoop(0), ..., RLoop(3) for this

RLoop(0) = 0

RLoop(1) = 1

$n^{f(n)} = e^{\ln n^{f(n)}} = e^{f(n)\ln n}$

RLoop(2) = 1

j=1 value = 0
value + RLoop(0) = 0 + 0

j=2 value = 0
value + RLoop(1) = 0 + 1 = 1

j=3 value = 1
↩ return

RLoop(3) = 3

j=1 value = 0
value + RL(0) = 0 + 0 = 0

j=2 value = 0
value + RL(1) = 0 + 1 = 1

j=3 value = 1
value + RL(2) = 1 + 2 = 3

j=4 value = 3
↩ return

Theorem 3.2.2. *Given an array A and indices $1 \le p \le r \le A.length$, after* MERGE-SORT$(A, p, r)$ *is called, the elements of the subarray $A[p..r]$ will consist of the original elements of $A[p..r]$, except in sorted order.*

Proof: Since Merge-Sort recursive, will do proof by induction on $n \geq 0$:

$P(n)$: "For every such $p \leq r$, if $r - p \leq n$, then after $MS(A,p,r)$ is called the elements of $A[p..r]$ are sorted and ..."

(Base case) $n = 0$ so $p = r$ so $A[p..r]$ has length 1 so already sorted.

(Inductive Step) Assume $n \geq 0$ and $P(n)$ true.

Let $p \leq r$ be s.t. $r - p = n+1 \geq 1$.

In line 2 set $q := \lfloor (p+r)/2 \rfloor$

Then $q - p$ and $r - q - 1 \leq n+1$ so $P(n)$ applies to lines 3–4

so prev. Prop ~~sorts~~ merges these two sorted subarrays.

Done ✓