① Recap Partition for quicksort

Quicksort Recurrence

$$T(n) = \frac{2}{n}\sum_{k=1}^{n-1} T(k) + n$$

⋮ derived last time

$$T(n) \leq \frac{n+1}{n} T(n-1) + 2$$

do Says, both calculations.

---

② Median, order stats

- $\boxed{\text{min}}$
- max
- $\boxed{\text{median}}$: $\lfloor \frac{n+1}{2} \rfloor$, $\lceil \frac{n+1}{2} \rceil$; $\frac{n+1}{2}$

A) $\boxed{\text{ORDER}}$ sort, find i-th element $\rightarrow \Theta(n \log n)$

B) PARTITION $\Rightarrow$ gives pivot position then recursively only on one side

Assume (like quicksort) splits are never worse than $\frac{1}{10}/\frac{9}{10}$

$\Rightarrow T(n) \leq T(\frac{9}{10}n) + \Theta(n) \Rightarrow \Theta(n)$ Master Th

Average case: same idea as quick sort, only 1 side

possible splits $0:n-1$, $1:n-2$, ... $n-1:0$

$$T(n) \leq \frac{1}{n}\sum_{i=0}^{n-1} T(\max(i, n-1-i)) + \Theta(n)$$

$$T(n) \leq \frac{2}{n}\sum_{i=\frac{n-1}{2}}^{n-1} T(i) + \Theta(n)$$

$\frac{2c}{n}(\frac{n-1}{2} \cdot n) + \Theta(n)$

$= c(n-1) + \Theta(n)$

$\leq c(n)$

...

Assume $T(n) \leq cn$

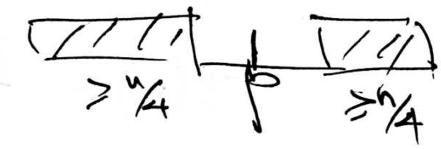
do Say calculation

$$T(n) \leq \frac{2}{n}\sum_{i=\frac{n-1}{2}}^{n-1} ci + \Theta(n) =$$

$$= \frac{2c}{n}\sum_{i=0}^{\frac{n-1}{2}} \left(\frac{n-1}{2} + i\right) + \Theta(n) = \frac{2c}{n}\left(\left(\frac{n-1}{2}\right)^2 + \frac{(\frac{n-1}{2})(\frac{n+1}{2})}{2}\right) + \Theta(n)$$

ORDER stats : worst case — Theoretical, not practical

- split groups of 5 $\Theta(n)$
- median each group $\Theta(n)$
- find median of medians $T(\frac{n}{5})$
- Partition by ~~median~~ p



it no worse than $3/4 - 1/4$ split

$$\Rightarrow T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{3n}{4}\right) + \Theta(n)$$

exercise: $\frac{n}{5} + \frac{3n}{4} < n \Rightarrow T(n)$ is $\Theta(n)$

---

③ LINEAR TIME SORT

- counting sort : example
- radix sort | OBS: counting sort does not change order to elements already sorted
- discussion on digits $r = \lg n \Rightarrow \Theta\left(\frac{bn}{\lg n}\right)$

---

④ GREEDY — makes local optimal choice (only works when = global)

- structure of the problem.

DP	GR
comb of solutions to subproblems	sequence of greedy choices
bottom up	top down
	optimal sol to pb contains optimal solution to sub-pb

- proof by induction : fract. Knapsack

- activity selection pb.

Ⓐ greedy choice (1) ∈ SOLUTION

Ⓑ greedy (i times works) ∈ SOLUTION $\Rightarrow$ greedy (i+1 times) ∈ SOL

OR "after the greedy choise i have the same pb as before", different parameters

VIP + SHOW sol combine

CS25-X94 Lecture 5 7/8/94

~~Reminder - using x-hour next Tuesday~~

Order statistics

Find the ith smallest of n elements.
(Elt. with rank i.)

$i=1 \Rightarrow$ min

$i=n \Rightarrow$ max

$i = \lfloor \frac{n+1}{2} \rfloor$ or $\lceil \frac{n+1}{2} \rceil \Rightarrow$ median

* → comment on simple solutions, $O(n)$, for just min, second smallest, etc.

* → One solution: sort, index ith elt

$O(n \lg n)$ time.

when extended to general case, it's $O(n^2)$.

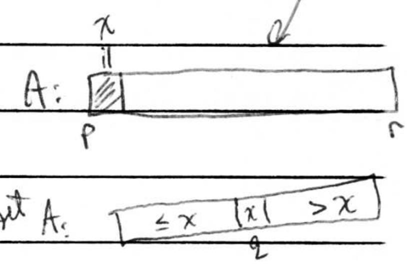
* → Idea: • partition around an element - explain on board w/ array $A[p..r]$ → $A[p]$ is pivot
• recurse on one-half or other...

Randomized alg

Divide & conquer.

Use: randomized partitioning from quicksort.

Randomized-Partition(A,p,r)
$k \leftarrow$ Random(p,r)
exchange $A[k] \leftrightarrow A[p]$
return Partition(A,p,r)

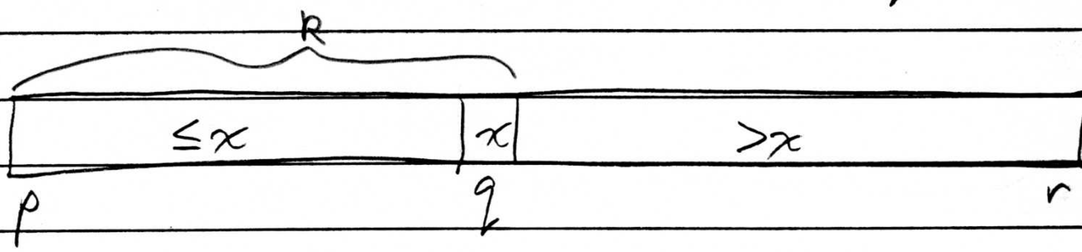


Note: if returns q, then $A[q]$ = pivot after return. $A[p..q-1] \le$ pivot, $A[q+1..r] >$ pivot!

Randomized-Select(A, p, r, i)
  if p = r
    then return A[p]
  q ← Randomized-Partition(A, p, r)
  k ← q − p + 1
  if i = k
    then return A[q]
  if i < k
    then return Randomized-Select(A, p, q−1, i)
    else return Randomized-Select(A, q+1, r, i−k)

added, not in book

slightly different from book



$i = k \Rightarrow A[q]$ is ith smallest in $A[p..r]$

$i < k \Rightarrow$ ith smallest in $A[p..r]$ is also ith smallest in $A[p..q-1]$

$i > k \Rightarrow$ " " " " $(i-k)$th smallest in $A[q+1..r]$

Analysis

Lucky case: All splits are at worst $\frac{9}{10}n : \frac{1}{10}n$

$T(n) \le T(\frac{9}{10}n) + \Theta(n)$.

~~$n^{\log_{10/9} 1} = n^0 \Rightarrow$ Case 3 of Master theorem $\Rightarrow \Theta(n)$.~~

$\log_{10/9} 1 = 0$ vs. $1 \Rightarrow$ Case 3 of MM $\Theta(n)$

<u>what if</u> $\frac{99}{100}n : \frac{1}{100}n$? $\Rightarrow$ still $\Theta(n)$

Worst case: all splits $0 : n-1$

$$T(n) = T(n-1) + \Theta(n)$$

$$\Rightarrow T(n) = \Theta(n^2)$$

<u>Average case analysis</u> : assume all splits equally likely

Splits : $\overbrace{0:n-1 \quad 1:n-2 \quad 2:n-3 \cdots \quad n-1:0}^{n}$ (general - $\underline{i : n-1-i}$)

$$T(n) \le \frac{1}{n} \sum_{i=0}^{n-1} T(\max(i, n-1-i)) + \Theta(n)$$ (assumes always recurse on larger half)

Consider n odd & n even, e.g.

$n=5$	$0{:}4 \quad 1{:}3 \quad \underline{2{:}2 \quad 3{:}1 \quad 4{:}0}$	$\cancel{\lfloor 5/2 \rfloor = 2}$ $\quad \lceil \frac{n-1}{2} \rceil = \lceil \frac{4}{2} \rceil = 2$
$n=6$	$0{:}5 \quad 1{:}4 \quad 2{:}3 \quad \underline{3{:}2 \quad 4{:}1 \quad 5{:}0}$	$\cancel{\lfloor 6/2 \rfloor = 3}$ $\quad \lceil \frac{n-1}{2} \rceil = \lceil \frac{5}{2} \rceil = 3$

$$T(n) \le \frac{2}{n} \sum_{i=\lceil \frac{n-1}{2} \rceil}^{n-1} T(i) + \Theta(n)$$

Use substitution method to show $T(n) = O(n)$

Assume $\quad T(j) \le cj \quad \forall j < n$

Verify $\quad T(n) \le cn$

$$T(n) \le 2/n \sum_{i=\lceil \frac{n-1}{2} \rceil}^{n-1} T(i) + \Theta(n)$$

$$\le \frac{2}{n} \sum_{i=\lceil \frac{n-1}{2} \rceil}^{n-1} ci + \Theta(n)$$

$$= \frac{2}{n} \left[\sum_{i=1}^{n-1} ci - \sum_{i=1}^{\lceil \frac{n-1}{2} \rceil - 1} ci \right] + \Theta(n)$$

$$= \frac{2}{n} \left\{ c \cdot \frac{(n-1) \cdot n}{2} - c \cdot \frac{(\lceil \frac{n-1}{2} \rceil - 1)(\lceil \frac{n-1}{2} \rceil)}{2} \right\} + \Theta(n)$$

$$\le \frac{2c}{n} \left[\frac{(n-1)\, n}{2} - \frac{(\frac{n-1}{2} - 1)(\frac{n-1}{2})}{2} \right] + \Theta(n)$$

$$= c(n-1) - \frac{c}{2}\left(\frac{n}{2} - 1\right) + \Theta(n)$$

$$= cn - c - \frac{cn}{4} + c/2 + \Theta(n)$$

$$= cn - \left(\frac{c}{2} + \frac{cn}{4} - \Theta(n) \right)$$

$$\le cn \quad \text{if} \quad \frac{c}{2} + \frac{cn}{4} - \Theta(n) \ge 0$$

– choose c, n_0 large enough to ensure this ...

$$= c(n-1) - \frac{2c}{n}\left(\frac{(\frac{n-3}{2})(\frac{n-1}{2})}{2} \right) + \Theta(n)$$

$$= c(n-1) - \frac{c}{n}\left(\frac{n^2 - 4n + 3}{4} \right) + \Theta(n)$$

$$= cn - \cancel{c} - \frac{cn}{4} + \cancel{c} - \frac{3c}{4n} + \Theta(n)$$

$$= cn - \left(\frac{cn}{4} + \frac{3c}{4n} - \Theta(n) \right)$$

$$\le cn$$

$$\le \frac{2}{n}\left(c \cdot \frac{(n-1)\, n}{2} - \frac{c \cdot (\frac{n}{2} - 1)(\frac{n}{2})}{2} \right) + \Theta(n)$$

L6, L7, P1

CS 25 Algorithms

10/4/95

Last time (chap 10)

- Order statistics

Today (chap 10, 9)

- Order statistics (cont.)
- Lower bounds for sorting
- Sorting in linear time
  - counting sort
  - radix sort

Announcements

- BJ's office hours
- Homeworks & solutions

$= c(n-1) - \frac{c}{2}\left(\frac{n}{2}-1\right) + \Theta(n)$
$= cn - c - \frac{cn}{4} + \frac{c}{2} + \Theta(n)$
$= cn - \left(\frac{cn}{4} + \frac{c}{2} - \Theta(n)\right)$
$\leq cn$ if c is large enough so that $c\left(\frac{n}{4}+\frac{1}{2}\right)$ dominates const in $\Theta(n)$

Excellent algorithm in practice.

<u>Worst-case linear-time alg</u>

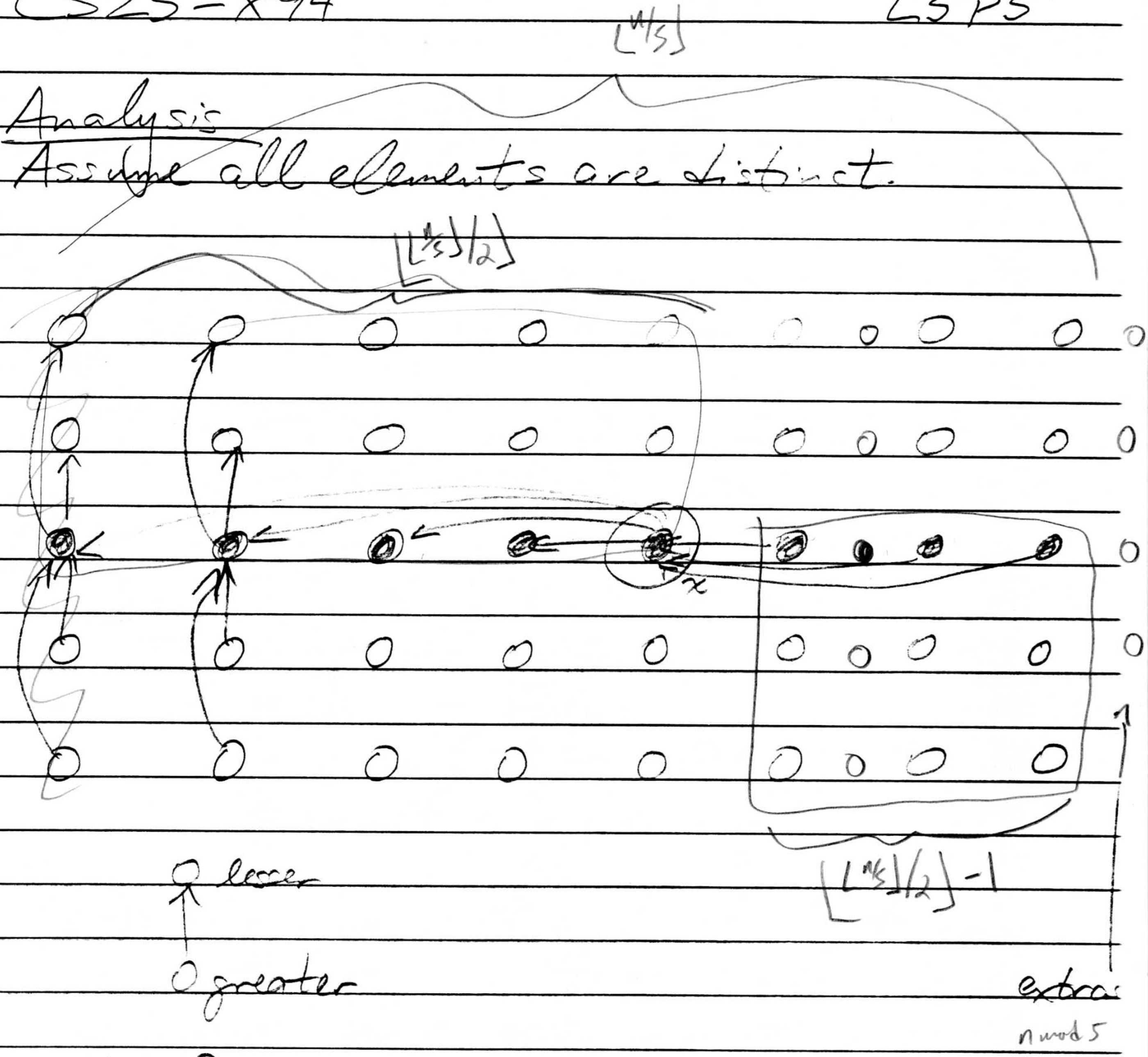
(*) Theoretical interest only
(*) Idea: always use a good pivot.
(*) Slightly different from book.

Select(i)

1. Divide n elts into $\lfloor \frac{n}{5} \rfloor$ groups of 5 with n mod 5 left over.
2. Find median of each of the $\lfloor \frac{n}{5} \rfloor$ groups of 5 by brute force.
3. Use select to recursively find the median x of the $\lfloor \frac{n}{5} \rfloor$ medians.
4. Partition the n elts around x. Let k = rank of x.
5. <u>if</u> $i = k$
   <u>then</u> <u>return</u> x
   <u>if</u> $i < k$
   <u>then</u> use Select to recursively find ith smallest in first
   <u>else</u> " " " " " $(i-k)$th " "second

Analysis

Assume all elements are distinct.



- $\geq \frac{1}{2}$ of the medians are $\leq x$
  $\Rightarrow \geq \lfloor \lfloor \frac{n}{5} \rfloor / 2 \rfloor = \lfloor \frac{n}{10} \rfloor$ medians $\leq x$
  $\Rightarrow \geq 3 \lfloor \frac{n}{10} \rfloor$ elts $\leq x$.

- $\geq \frac{1}{2}$ of medians, minus 1, are $> x$
  $\Rightarrow \geq \lfloor \lfloor \frac{n}{5} \rfloor / 2 \rfloor - 1 = \lfloor \frac{n}{10} \rfloor - 1$ medians $> x$
  $\Rightarrow \geq 3 \lfloor \frac{n}{10} \rfloor - 3$ elts $> x$.

$3\lfloor \frac{n}{10} \rfloor - 3 \geq 3(\frac{n}{10}-1) - 3 = \frac{3n}{10} - 6 \geq \frac{n}{4}$
if $n(\frac{3}{10} - \frac{1}{4}) \geq 6$
$\Leftrightarrow n \geq 120$

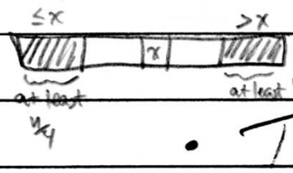
CS 25-X94 $\quad$ & $3\lfloor \frac{n}{10} \rfloor \geq n/4$ (otherwise) $\quad$ L5P6

- $n \geq 60 \Rightarrow 3\lfloor \frac{n}{10} \rfloor - 3 \geq \frac{n}{4}$.

  Get rid of floor, say RT for $n < 60$ is O(1)

not true! (try 61)

- $\therefore$ After partitioning around x, ~~step 5~~ is called on $\leq \frac{3}{4} n$ elt.



- $T(n) \leq T(\lfloor \frac{n}{5} \rfloor) + T(\frac{3}{4} n) + \Theta(n)$

  $\leq T(\frac{n}{5}) + T(\frac{3}{4} n) + \Theta(n)$ $\qquad \frac{n}{5} + \frac{3}{4} n = \frac{19}{20} n$

- have class guess solution
- give intuition

Show: $T(n) \leq cn$ by substitution.

$$T(n) \leq \frac{cn}{5} + \frac{3cn}{4} + \Theta(n)$$
$$= \frac{19}{20} cn + \Theta(n)$$
$$= cn - (\frac{1}{20} cn - \Theta(n))$$
$$\leq cn \text{ if } c \text{ big enough.}$$

- Intuition: work at each level of recursion is const factor — $\frac{19}{20}$ — smaller $\Rightarrow$ geometric series $\Rightarrow$ work at root ($\Theta(n)$ term) dominates.

Lower bounds for sorting

CS 25-X94 Lecture 6 7/11/94

~~Reminder - using x-hour tomorrow.~~

How fast can we sort?
Will provide LB, then beat it by making different assumptions.

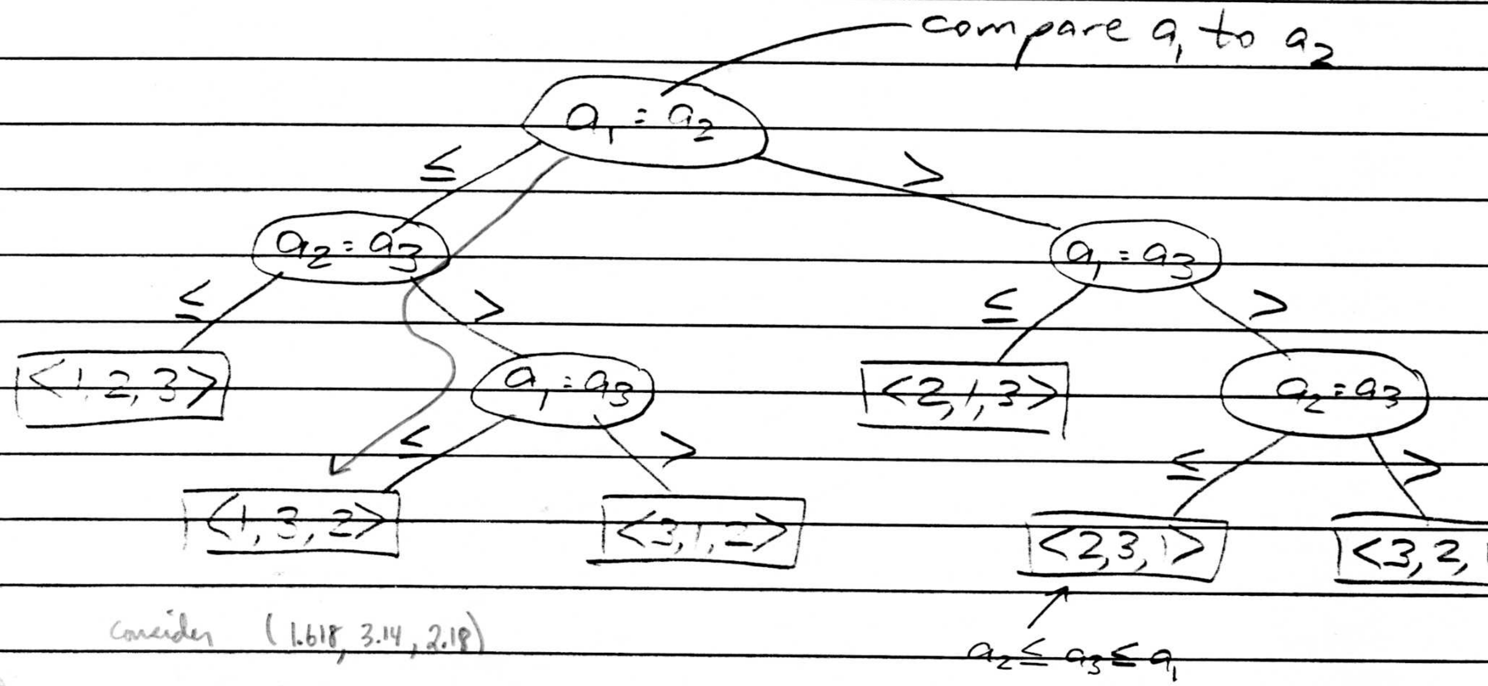
<u>Comparison sorting</u> - the <u>sorted order determined</u> is based <u>only on comparisons</u> between input elts.

E.g. Insertion sort, merge sort, quicksort, heapsort
Bubble sort

<u>Lower bound</u>
$\Omega(n)$ to examine all the input.
We'll show $\Omega(n \lg n)$.

<u>Decision tree</u>
Abstraction of any comparison sort.
Represents comparisons made by a sorting alg. on input of a given size.
Ignores everything else - control, data movement, etc.

For insertion sort: on 3 elements



Each leaf annotated by the permutation the alg. determines.

Show insertion sort for sample array <9,2,6>, ending up at <2,3,1> ⟹ sorted order is $\langle a_2, a_3, a_1 \rangle = \langle 2, 6, 9 \rangle$.

How many leaves? $\geq n!$ or else there's a missing perm.

Don't need $> n!$, but an alg could have more if redundant decisions or paths that will never be executed.

CS25-X94 L6P3

DT can model a comparison sort.

For a particular sorting alg:

- 1 tree for each n.
- view as if alg splits into 2 at each decision.
- tree of all possible execution traces.

What's length of longest path root → leaf in insertion sort tree? $\Theta(n^2)$

Merge sort tree? $\Theta(n \lg n)$

Lemma: Binary tree of height h has $\leq 2^h$ leaves. ($\forall$ binary trees of height h, # leaves $\leq 2^h$

height = max depth

root = depth 0

Proof: By induction on h. (strong induction)

Basis: $h = 0$. Tree is just 1 node, which is a leaf, $2^h = 1$.

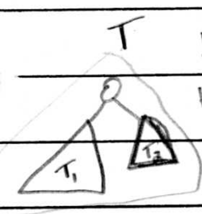
~~Inductive step: Assume true for h−1. Extend tree of height h−1 by making as many new leaves as possible. Each leaf becomes parent to 2 new leaves.~~

~~# leaves for height h~~

~~$\leq 2 \cdot$ (# leaves for height h−1)~~

~~$\leq 2 \cdot 2^{h-1}$~~

~~$= 2^h$.~~ $\boxtimes$



$H(T) = h$

$H(T_1) = h_1 \leq h-1$

$H(T_2) = h_2 \leq h-1$ · remove root

· at most 2 subtrees of height at most h−1

$L(T) = L(T_1) + L(T_2)$ (construction)

$\leq 2^{h_1} + 2^{h_2}$ (ind. hyp.)

$\leq 2^{h-1} + 2^{h-1}$ (prop of trees)

$= 2^h$ ✓ (alg.)

CS 25 - Algorithms

10/6/95

Last time (chap 10, 9)

- Finish order stat.
- Lower bounds for sort.

Today (chap 9, 16.1)

- Counting sort
- Radix sort
- Dynamic programming

Handouts

- HW 1 Solutions
- HW 1 graded papers
- HW 3

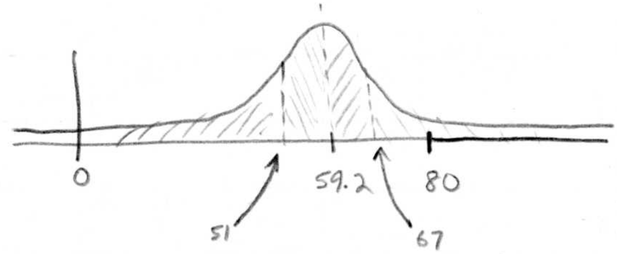
HW 1

max = 80

$\bar{x} = 59.2$

$s = 12.1$

$\frac{2}{3}s \approx 8$



Note:

$[\min, \bar{x} - \frac{2}{3}s] = [0, 51] \approx$ lowest 25%

$[\bar{x} - \frac{2}{3}s, \bar{x}] = [51, 59] \approx 2^{nd}$ 25%

$[\bar{x}, \bar{x} + \frac{2}{3}s] = [59, 67] \approx 3^{rd}$ 25%

$[\bar{x} + \frac{2}{3}s, \max] = [67, 80] \approx$ top 25%

$\bar{x} = 93.4$

$\frac{2}{3}\sigma = 3.7$

Sorting in linear time

Non-comparison sorts.

E.g. Consider sorting people by
- age in years
- height in inches
- shoe size
- weight in pounds
- etc.

Counting sort

Depends on key assumption:

Numbers to be sorted are integers $\{1, 2, \ldots, k\}$.

Input: $A[1..n]$, $A[j] \in \{1, 2, \ldots, k\}$

Output: $B[1..n]$, sorted

Uses $C[1..k]$ as aux storage

Counting-Sort(A, B, n, k)

$\Theta(k)$ for $j \leftarrow 1$ to k
  do $C[j] \leftarrow 0$

$\Theta(n)$ for $j \leftarrow 1$ to n
  do $C[A[j]]$ ++ ← increment

$\Theta(k)$ for $j \leftarrow 2$ to k
  do $C[j] \leftarrow C[j] + C[j-1]$

$\Theta(n)$ for $j \leftarrow n$ downto 1
  do $B[C[A[j]]] \leftarrow A[j]$
    $C[A[j]]$ -- ← decrement

$\Theta(n+k)$

Example for $A = 3_1\ 6_1\ 4_1\ 1_1\ 3_2\ 4_2\ 1_2\ 4_3$ (next page)

$\boxed{\text{Stable}}$ because of how last loop works.

Analysis: $\Theta(n+k)$
$= \Theta(n)$ if $k = O(n)$

How big a k is practical?
  Good for sorting 32-bit values? No.
  16-bit? Probably not.
  8-bit? Maybe.
  4-bit? Probably.

Will use in radix sort. (two pages)

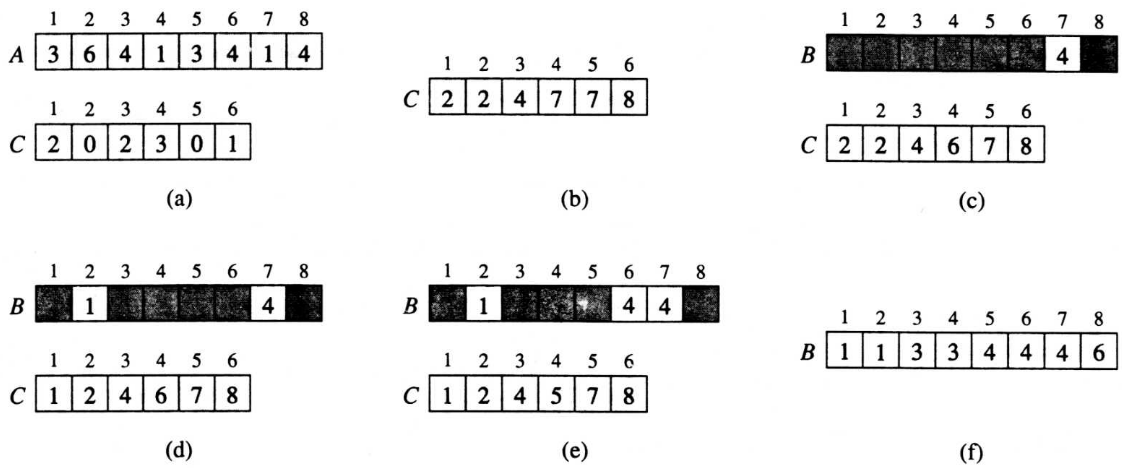


Figure 9.2 The operation of COUNTING-SORT on an input array $A[1\,..\,8]$, where each element of A is a positive integer no larger than $k = 6$. **(a)** The array A and the auxiliary array C after line 4. **(b)** The array C after line 7. **(c)–(e)** The output array B and the auxiliary array C after one, two, and three iterations of the loop in lines 9–11, respectively. Only the lightly shaded elements of array B have been filled in. **(f)** The final sorted output array B.

```
COUNTING-SORT(A, B, k)
 1  for i ← 1 to k
 2       do C[i] ← 0
 3  for j ← 1 to length[A]
 4       do C[A[j]] ← C[A[j]] + 1
 5  ▷ C[i] now contains the number of elements equal to i.
 6  for i ← 2 to k
 7       do C[i] ← C[i] + C[i − 1]
 8  ▷ C[i] now contains the number of elements less than or equal to i.
 9  for j ← length[A] downto 1
10       do B[C[A[j]]] ← A[j]
11          C[A[j]] ← C[A[j]] − 1
```

Counting sort is illustrated in Figure 9.2. After the initialization in lines 1–2, we inspect each input element in lines 3–4. If the value of an input element is i, we increment $C[i]$. Thus, after lines 3–4, $C[i]$ holds the number of input elements equal to i for each integer $i = 1, 2, \ldots, k$. In lines 6–7, we determine for each $i = 1, 2, \ldots, k$, how many input elements are less than or equal to i; this is done by keeping a running sum of the array C.

Finally, in lines 9–11, we place each element $A[j]$ in its correct sorted position in the output array B. If all n elements are distinct, then when we first enter line 9, for each $A[j]$, the value $C[A[j]]$ is the correct final position of $A[j]$ in the output array, since there are $C[A[j]]$ elements less

Radix sort

explain afterwards

How IBM made its money.
Card sorter – can sort cards based on one column at a time.
How to sort on all columns?
For card sorter, need a human operator, who becomes part of the alg.

start here

Key idea: sort __least__ signif digit first.

Radix-Sort(A, d)
  __for__ i ← 1 __to__ d
    __do__ use a stable sort to sort array A on digit i

put up example now... (next page)

⟹ P6.1

Correctness

Claim (∀i) After i passes of Radix Sort, the numbers are sorted based on least significant i digits.

Induction on # of passes.
Assume digits 1, ..., i−1 are ordered.
Show stable sort on digit i leaves digits 1, ..., i ordered:

Pf.

- 2 digits in pos i differ ⟹ sorting by digit i is correct – lower-order digits irrelevant.
- 2 digits in pos i same ⟹ stable sort puts them in right order.

Important: intermediate sorting alg must __be stab__

Example:

329		720		720		329
457		355		329		355
657		436		436		436
839	⇒	457	⇒	839	⇒	457
436		657		355		657
720		329		457		720
355		839		657		839

Thm: Any decision tree that sorts n elts has height $\Omega(n \lg n)$.

Proof: Tree has $\geq n!$ leaves. Let h be its height. By lemma, $\cancel{n! \leq 2^h}$ $\left(\frac{n}{2}\right)^{n/2} \leq n! \leq (\#\text{leaves}) \leq 2^h$

Take logs: $h \geq \lg(n!)$.

By Stirling's approx: $n! > \left(\frac{n}{e}\right)^n$.

$\therefore h \geq \lg\left(\frac{n}{e}\right)^n$

$= n \lg n - n \lg e$

$= \Omega(n \lg n)$. $\boxtimes$

OR $\left(\frac{n}{2}\right)^{n/2} \leq n! \leq 2^h$

$\Leftrightarrow h \geq \frac{n}{2} \lg \frac{n}{2}$

$\Rightarrow h = \Omega(n \log n)$ $\boxtimes$

Cor: Heapsort & merge sort are asympt. optimal.

Sorting in linear time

Non-comparison sorts.

Counting sort

Depends on key assumption:

Numbers to be sorted are integers $\{1, 2, \ldots, k\}$.

Input: $A[1..n]$, $A[j] \in \{1, 2, \ldots, k\}$

Output: $B[1..n]$, sorted

Uses $C[1..k]$ as aux storage

Thm: Any decision tree that sorts n elts has height $\Omega(n \lg n)$.

Proof: Tree has $\geq n!$ leaves. Let h be its height. By lemma, $(\frac{n}{2})^{n/2} \leq n! \leq (\#\text{leaves}) \leq 2^h$

Take logs: $h \geq \lg(n!)$.

By Stirling's approx: $n! > \left(\frac{n}{e}\right)^n$.

$\therefore h \geq \lg\left(\frac{n}{e}\right)^n$

$= n\lg n - n\lg e$

$= \Omega(n \lg n)$. $\boxtimes$

OR $(\frac{n}{2})^{n/2} \leq n! \leq 2^h$

$\Leftrightarrow h \geq \frac{n}{2} \lg \frac{n}{2}$

$\Rightarrow h = \Omega(n \log n)$ $\boxtimes$

Cor: Heapsort & merge sort are asympt. optim.

Sorting in linear time

Non-comparison sorts.

Counting sort

Depends on key assumption:

Numbers to be sorted are integers $\{1, 2, \ldots, k\}$.

Input: $A[1..n]$, $A[j] \in \{1, 2, \ldots, k\}$

Output: $B[1..n]$, sorted

Uses $C[1..k]$ as aux storage

Analysis

here, k is radix

Use counting sort as intermediate sort.

$\Theta(n+k)$ per pass.

$\Theta(d(n+k))$ total.

$k = O(n) \Rightarrow \Theta(dn)$.

Breaking keys into digits

n keys, b bits/key.

View each key as $d = \frac{b}{r}$ digits of r bits each.

Example: $b = 32 \Rightarrow$ can view as 4 8-bit digits.

r-bit digit has value in $\{0, \ldots, 2^r - 1\}$, so for counting sort, $k = 2^r$.

Time for radix sort $= \Theta\left(\frac{b}{r}(n + 2^r)\right)$.

How to pick r? b, n given.

Increase $r \Rightarrow$ fewer passes, more time & space per pass.

Choosing $\boxed{r \approx \lg n} \Rightarrow \Theta\left(\frac{b}{\lg n}(n+n)\right)$
$= \Theta\left(\frac{bn}{\lg n}\right)$.

Compare: when is $\frac{bn}{\lg n} \le n \lg n$?

$b \le \lg^2 n$

$\sqrt{b} \le \lg n$

$n \ge 2^{\sqrt{b}}$

say $b = 32$, then for $n \ge 2^{\sqrt{32}} \approx 50$

Comparison to other sorts

2000 32-bit integers to sort.

Merge sort, quicksort make $\ge \lg 2000 \approx 11$ passes over data.

Radix sort with $\lg 2000 = 11$-bit digits makes only 3 passes? Not really — each Counting-Sort call makes 2 passes over data + 2 passes over C
$\Rightarrow$ 12 passes. Radix sorts starts winning for $n \ge 2000$ or so.