- If median value in n values  i=0;	Median Value 1 (1991) Man Not lemon how
i=0; min man of max  i=size-1; min of max  lack 2n comparisons  too nony  loop 2n comparisons  loop 2n comparisons  loop 2n compare Ali] vs max  compare Ali] vs max  compare Ali] vs min n values  x x  ith  The Sclection Problem (kth largest element)  - Sorting the array the indexing would be Olutogn)  - The algorithm for finding the kth largest is based on quicksort(). Expected O(n) We O(n²)  - Randomized sclection of pivot (probably good enough)  (could use m-o-3)  randomized_select (A,p,r,i) { index find ith smallest  if (p==r) { return Alp]; }  q= randomized partition (A,p,r); //randomized pivot, like qs() partition  k=g-p+1;  if (i==k) { return Alp]; }  if (i==k) { return randomized_select (A,p,q-1,i):	- If median value in n values
hack)  2n comparisons  100 toop) vs two loops  100 compare A[i] vs max  100 compare A[i] vs max  100 compare A[i] vs min  100 compare A[i] vs min	i=0; min or max i=size-1; ] min or max
One loop vs two loops  Sompare Alil vs mas  Compare Alil vs mas  Compare Alil vs min  In values  Expected Oln Lice Oln 2  - Randomized selection of pivot (probably good enough)  Could use m-o-3)  Frandomized select (A, p, r, i) { index find ith smallest if (p==r) { return Alp!; }  g= randomized partition (A, p, r); //randomized pivot, like gs() partition  k=g-p+1;  if (i==k) { return Alg!; }  if (i=k) { return randomized select (A, p, q-1, i); }	A SISTEM SELVENT SELVE
Compare A[i] vs max  Compare A[i] vs min  ith  The Selection Problem (kth largest element)  - Sorting the array the indexing would be Olulogn)  - The algorithm for finding the kth largest is based on quicksort(). Expected Oln) Lic Oln2)  - Randomized selection of pivot (probably good enough)  (could use m-o-3)  randomized_select (A, p, r, i) { index find ith smallest if (p=r) { return A[p];}  q=randomized-partition (h, p, r); //randomized pivot, like qs() partition (k=q-p+1; if (i=k) { return randomized_select (A, p, q-1, i);}	(hack) 2n comparisons
Compare A[i] vs max  Compare A[i] vs min  ith  The Selection Problem (K+m largest element)  - Sorting the array the indexing would be Olnlogn)  - The algorithm for finding the k+m largest is based on quicksorte). Expected O(n) LC O(n²)  - Randomized selection of pivot (probably good enough)  (could use m-o-3)  randomized_select (A, p, r, i) { index find ith smallest if (p==r) { return A[p]; }  g= randomized_partition (A, p, r); //andomized pivot, like qs() partition k=g-p+1; if (i==k) { return randomized_select (A, p, q-1, i);	min & max
Compare A[i] vs min  [compare A[i] vs min  [x]  ith  The Selection Problem (Kth largest element)  - Sorting the array the indexing would be Olulogen)  - The algorithm for finding the kth largest is based on quicksort(). Expected O(n) WC O(n²)  - Randomized selection of pivot (probably good enough)  [could use m-o-3)  randomized select (A, p, r, i) { index find ith smallest if (p==r) { return A[p]; }  g= randomized partition (A, p, r); //andomized pivot, like qs() partition (k=g-p+1;  if (i==k) { return randomized select (A, p, q-1, i):	1000
Compare A[i] vs min  (x)  ith  The Selection Problem (kth largest element)  - Sorting the array the indexing would be Olnlogn)  - The algorithm for finding the kth largest is based on quicksort(). Expected O(n) WC O(n²)  - Randomized selection of pivot (probably good enough)  (could use m-o-s)  randomized select (A,p,r,i) { index find ith smallest if (p==r) { return A[p]; }  q= randomized-partition (A,p,r); //randomized pivot, like qs() partition (k=q-p+1;  if (i==k) { return randomized select (A,p,q-1,i):	compare ALIJ VS max
The Selection Problem (Kth largest element)  - Sorting the array the indexing would be Olnlogn)  - The algorithm for finding the kth largest is based on quicksort(). Expected O(n) LC O(n2)  - Randomized selection of pivot (probably good enough)  (could use m-o-3)  randomized select (A, P, r, i) { index find ith smattest if (p==r) { return A[p]; }  q= randomized partition (A, P, r); //randomized pivot, like qs() partition (k=q-p+1; if (i==k) { return A[q]; }  if (i<=k) return randomized select (A, P, q-1, i):	Compare A[i] vs min
The Selection Problem (Kth largest element)  - Sorting the array the indexing would be Olnlogn)  - The algorithm for finding the kth largest is based on quicksort(). Expected O(n) LC O(n2)  - Randomized selection of pivot (probably good enough)  (could use m-o-3)  randomized select (A, P, r, i) { index find ith smattest if (p==r) { return A[p]; }  q= randomized partition (A, P, r); //randomized pivot, like qs() partition (k=q-p+1; if (i==k) { return randomized select (A, P, q-1, i);	The same of the X of the Manual State of the
[could use m-o-3]  randomized_select (A, p, r, i) { index find ith smallest if (p==r) { return A[p]; }  g= randomized_partition (A, p, r); //randomized pivot, like qs() partition (f (i==k)) { return A[q]; }  if (i==k) { return randomized_select (A, p, q-1, i);	The Selection Problem (kth largest element) - Sorting the array the indexing would be Olnlogn) - The algorithm for finding the kth largest is based on
if (p==r) { return A[p];}  g= randomized_partition (A, p, r); //randomized pivot, like qs() partition k=g-p+1; if (i==k) { return A[q];}  if (i <k) (a,="" a-1,="" i);<="" p,="" randomized_select="" return="" td=""><td>(could use m-0-3)</td></k)>	(could use m-0-3)
if (i==k) {return A[q];}  if (i=k) return randomized_select (A, p, q-1, i);	if (p==r) { return A[p];}
if (i<1) return randomized_select (A, P, a-1, i);	
return randomized_select (A, a+1, ri-1c).	if (i<1) return randomized select (A = 1)
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	return randomized_select (A, q+1, r, i-1c);

de Kepal topul stall

ACCOMPLISHAN

- n (logn) compression algorithm

Data Representation

Fixed Length Encoding vs Variable Length Encoding

Fixed => ascii, always 7 bits

- Variable => Morse, variable lengths of encoding, common letters have shorter codes

- A prefix code is a code in which home of the codes have the same prefix

Optimal Substituti Substructure

Implementation "the best" or "equivalent" - A greedy algorithm obtains an optimal solution by making a series of choices. At each point in the algorithm, the choice that seems the best is chosen. (this works for some problems, not all, in general no way to tell)

1 m in binary 111 ~ ] can't distinguish these so both these codes cannot be chosen

That's mhy we need prefix codes

Prefix Code

- No code is the prefix of another code

- optimal - Full binary tree Alphabet - has size m

· Build a binary tree 0= left 1= right

· Alphabet is size no, should have m leaves, and m-1 internal nodes

· Assume S is a set of m characters, each with frequency f(x)

· Uses a min priority queue (queued on frequency)

Date

huffman!)

stores characters/frequency in PQ (build heap() O(n))

O(n) repeat n-1 times

make a new node T

O(logn) Left = PQ. extract Min();

O(logn) Right = PQ. extract Min();

Attach (eft and right to T

f(T) = f(Left) + f(Right)

Ollegn) PQ. heaplasert (1).

Complexity

O(1) O( livgn) O(n) O(n togn) O(n2) O(n3)

- Can all problems be solved in Polynomial Time?

5 NO

Halting Problem (Alan turing 1936 - Go del 1931)

- "Criven a program and an input to that program,
determine (programatically) if that program will
eventually stop with that input"

Quick 'Proof' of Halting Problem Claims Solves Halting
Step 1:

Bool does It stop (program p, input i) {

if Lolever code here) {

return true;

} clsc {

computer magic

return false;
}

```
bool stops Unself (program p) {
Step 2:
   return does Itstop (p,p);
Step 3: "woah there Free!"

bool wTF (program p) {

if (stops On | Self (p) {
        white (true); < inf. loop
      return false;
     } ... Flee ... (mentalo ... 13 mand 51
 return true;
   A Homelie there exchedout it places to he ?
Step 4 tidshamas no talk aid tot
   · Run with on itself
      1) Runs forever - stops Onself | Contradiction
      2) Stops and returns true
* Look up proofs of halting problems
                 > 0(1) O(logn) O(n)
Even with inf. time
  Ungolvable Problems
                     [Non-Deterministic Polynomial
   E.G. Halfing Problem
                       Polynomial.
                         · Problems whose status is unknown
 P + NP - Hasn't been prooved
                         · Problems for which no polynomial time
                          algorithm has been found
Solution to NP problems
                         · But if "someone hands you a solution"
 4> Approximation
                           you can verify correctness in
```

polynomial fime

Bin Packing Problem

Given a set of n objects (size of each is (0.0-1.01))  $0 < S_i \le 1$ pack all of the objects in the fewert number of unit sized bins (1.0)

- NP is the optimal solution

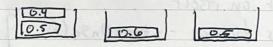
So not practical/done/doable

But we can do an approximate solution

4> First FIT Olutogn)

4> Solution is ~ 17/10 (about 2:1 of best solution)

4> Heuristic, takes each object & places it in the first bin that can accompadate it



For as07, commands can have a line of a command is 1s always started as a # 
SEX: # insert here

HINT:

ship & go to next line