Date

Median Value

- If median value in n values

$i = 0;$
$i = size - 1;$] min max value → $O(n)$ min or max

hack

min & max

One loop vs two loops

loop

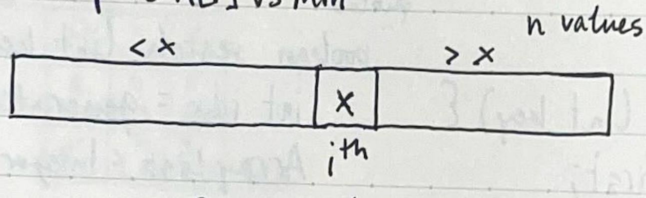
2n comparisons too many

⇓

$\frac{3n}{2}$ comparisons

Compare A[i] vs max

Compare A[i] vs min



The Selection Problem (k^{th} largest element)

- Sorting the array the indexing would be $O(n \log n)$
- The algorithm for finding the k^{th} largest is based on quicksort(). Expected $O(n)$ WC $O(n^2)$
- Randomized selection of pivot (probably good enough) (could use m-o-3)

```
randomized_select(A, p, r, i) {        index  find i^th 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 A[q]; }
  if (i<k) return randomized_select(A, p, q-1, i);
  return randomized_select(A, q+1, r, i-k);
}
```

Date

Huffman Codes

~ n(log n) ~~comparison~~ 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 none of the codes have the same prefix

Implementation

"the best" or "equivalent"

Optimal ~~Substituti~~ Substructure

- 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 ~~~ in binary
111 ~~~ can't distinguish these
so both these codes <u>cannot</u> be chosen

That's why 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 m, 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)

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.extractMin();

O(logn) Right = PQ.extractMin();

Attach left and right to T

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

O(logn) PQ.heapInsert(T);

Complexity

O(1) O(log n) O(n) O(n log n) O(n^2) O(n^3)

- Can all problems be solved in Polynomial Time?
  - ↳ NO

Halting Problem (Alan turing 1936 — Gödel 1931)

- "Given 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 Problem

Step 1:

```
bool doesItStop(program p, input i) {
    if (clever code here) {
        return true;
    } else {
        return false;
    }
}
```

Computer magic

Date

Step 2:

```
bool stopsOnSelf (program p) {
    return doesItStop(p, p);
}
```

Step 3:

```
bool wTF (program p) {      ← "woah there Fred"
    if (stopsOnSelf (p) {
        while (true); ← inf. loop
        return false;
    }
    return true;
}
```

Step 4

- Run wtF on itself
  1) Runs forever - stopsOnSelf
  2) Stops and returns true

  ] Contradiction

* Look up proofs of halting problems

Even with inf. time

$\boxed{\text{Unsolvable Problems}}$ →

E.G. Halting Problem

↳ NP
↳ Non-Deterministic Polynomial

- Problems whose status is unknown
- Problems for which no polynomial time algorithm has been found
- But if "someone hands you a solution" you can verify correctness in polynomial time

O(1) O(log n) O(n)

↳ P
↳ Polynomial

P ≠ NP ← Hasn't been prooved

Solution to NP problems

↳ Approximation

Bin Packing Problem

Given a set of n objects (size of each is (0.0-1.0])

$$0 < S_i \leq 1$$

pack all of the objects in the fewest number of unit sized bins (1.0)

- NP is the optimal solution
  - -Hard

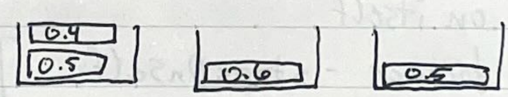
So not practical/done/doable

But we can do an approximate solution

↳ First Fit $O(n \log n)$

↳ Solution is $\sim \frac{17}{10}$ (about 2:1 of best solution)

↳ Heuristic, takes each object & places it in the first bin that can accomodate it



For as07, commands can have a line & a command

↳ Is always started as a #

↳ Ex: # insert here

Hint:

↳ Read whole line, search for #, if # is present remove/skip & go to next line