Medians and order statistics

Find the min and max

- Finding the min or max is $O(n)$
- You can find the min and max in $O(\frac{3}{2}n)$

$(0 \rightarrow \underset{\text{max}}{O}\ \underset{\text{min}}{O})\ O\ O\ O\ O\ O$

Kth largest element => The selection problem

- Sorting the array the indexing would be $O(n \lg n)$
- The algorythm for finding the kth largest is based on quicksort(). Expected $O(n)$, wc $O(n^2)$

```
randomized_select(A,p,r,i) {        i: index
    if (p==r) return A[p]
    q = randomized_partition(A,p,r);   //randomized pivot, like quicksort() 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);
}
```

Exponential problems

The problem => P/NP

- $O(k^n)$

* Solvable with infinite time, but we do not have infinite time, so the results are estimated
- Checking if an exponential solution is correct can usually be done in polynomial time
- Is P = NP is one of the greatest problems in CS and mathematics
- If one NP problem is solved in P, many will be able to be solved

Bin packing problem => An example of an exponential problem

- How much stuff can we pack into a bin
- Series of n objects $i = 1...n$ with size $0 < S_i < 1$, pack these items into the fewest number of unit-sized bins.
- Category P-hard
- Approximate solution first_fit gives a solution in $O(n \lg n)$ which is $\frac{17}{10}$ of the optimal solution
- first_fit uses a heuristic which takes each object and places it in the first bin it fits in.

Unsolvable problems

Overview

- Unsolvable problems are problems which no matter the amount of time the problem cannot be solved

Halting problem ⇒ Proof it is unsolvable

Step 1:

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

Step 2:

```
bool stops_on_self(program p){
    return does_it_stop(p, p);
}
```

Step 3:

```
bool wtf (program p){
    if (stops_on_self()){
        while (true) {;}  //infinite loop
        return false;
    }
    return true;
}
```

Step 4:

- Run wtf on itself
- (1) Runs forever - stops_on_self
- (2) Stops and returns true - stops_on_self returns false, runs forever

Huffman codes

Overview ⇒ $n(\lg n)$ compression algorythm

- Created by the founder of the CS program at UCSC
* Saves 20 to 90 % depending on data
* Variable length codes

Fixed vs variable length codes

- 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

- A greedy algorythm finds an optimal solution by making a series of choices, each choice is the

"best choice" with limited/local information. If something can be solved by a greedy algorythm it is defined as an optimal substructure

- Build a binary tree in which 0 is left and 1 is right
- Alphabet is size n, tree with n leaves, n-1 nodes
- Assume S is a set of n characters each with frequency f(x)
- Uses a ~~max~~ min priority queue

Pseudo code => Results in tree (PQ)

```
huffman() ← Incorrect, Steve will fix next class
    store characters/frequency in PQ (buildheap())
    repeat n-1 times
        make a new node T
        L = PQ.extract_max()
        R = PQ.extract_max()
        Attach L and R to T
        f(T) = f(L)+f(R)
        PQ.heap_insert(T)
```