COP3503 2/8/2024

Algorithm Analysis Day

Review CS1

Summations + Arith Sum + Geo Sum
Loop/Code Analysis
Sorted List Matching
~~MCSS?~~

New Stuff

Formal Def of O, Ω, Θ perkins to algorithms
MCSS
Probability for Expected Value
Experimental Run Time (can't figure out big-oh, run it a bunch ⇒ figure out)
Avg Case Analysis: Binary Search, Quick Sort
Quick Select

$f(n) = O(g(n))$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = C$, C is constant

$f(n) \leq g(n)$ "$f(n)$ is no bigger than $g(n)$ within a constfactor"

$f(n) = \Omega(g(n))$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} > 0$

$f(n) \geq g(n)$ "$f(n)$ is at least as big as $g(n)$ within a constfactor"

$f(n) = \Theta(g(n))$ iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

Sorting Stuff

	Best	Worst
Insertion Sort	$\sim n$	$\sim n^2$

runs in $O(n^2)$, runs in $\Omega(n)$

can't make a theta claim about insertion sort.

$$c_1 n \leq \boxed{?} \leq c_2 n^2$$

Merge Sort	$n \lg n$	$n \lg n$

runs in $O(n \lg n)$, runs in $\Omega(n \lg n)$

$\Longrightarrow \Theta(n \lg n)$

$$c_1 n \lg n \leq \boxed{?} \leq c_2 n \lg n$$

Whenever someone says provide a tight Big-Oh bound, what they mean is find $f(n)$ such that the algorithm runs in $O(f(n))$ time and such a statement can <u>not</u> be made for an asymptotically smaller function.

Insertion	$O(n^2)$
Quick	$O(n^2)$
Merge	$O(n \lg n)$
Heap	$O(n \lg n)$
Bubble	$O(n^2)$
Selection	$O(n^2)$

tight big oh bounds

$g(n)$ is smaller than $f(n)$ asymptotically

iff $\lim_{n \to \infty} \frac{g(n)}{f(n)} = 0$

Maximum Contiguous Subsequence Sum

input list: 2, 3, -1, -2, 3, -7, 6, -1, -2, 4, 5, -3, 2, 7, -2

contiguous subsequence

$-2+3-7+6-1 = -1$

Contiguous Subseq Sum from index x to index y

$$\sum_{i=x}^{y} a_i$$

Of all boxes we can draw what's the max sum?

```
int res = a[0];
for (int i = 0; i<n; i++) {          //start
    for (int j = i; j<n; j++) {      //end
        int cur=0;
        for (int k = i; k<=j; k++)
            cur += a[k];
        res = max(res, cur);
    }
}
//res is ans
```

move

remove

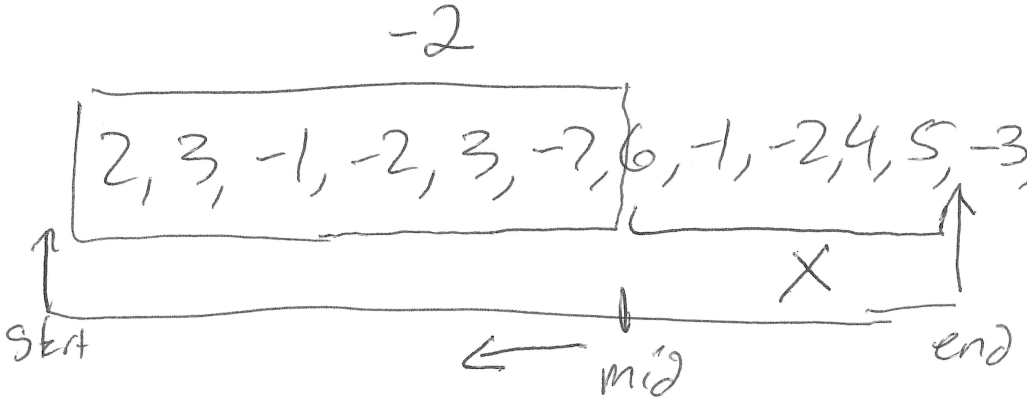
use j index

$O(n^3)$ w/more mem $\theta(n^3)$

$i=0, \; j=10 \quad sum = 8$

$i=0, \; j=11 \quad k=0,1,2,3 \quad cur=0$

Optimization #1: Don't reset sum to 0. As j loops, just update sum so you don't re-add lots of stuff

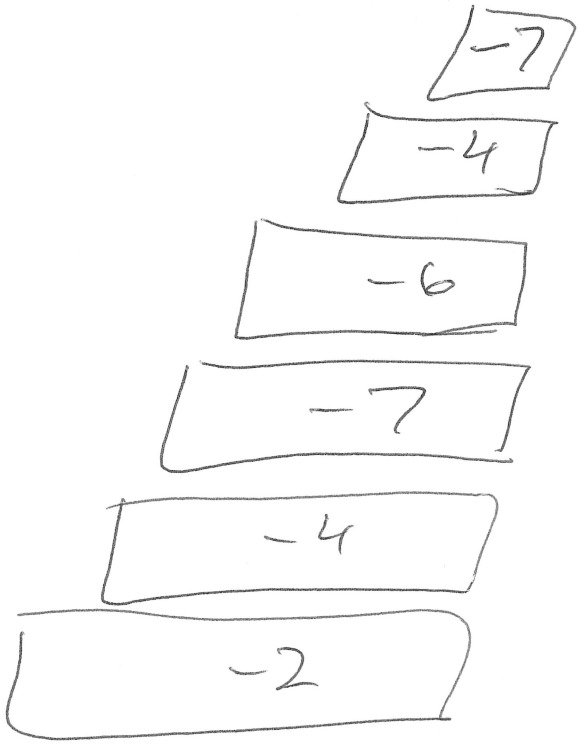


Sum(mid+1, end) = X

Sum(start, end) = X-2 (best?) NO

If mid exists, it never makes sense to look at start as a starting pt.

Go backwards from mid



negative values that can be cutoff for better results

So no maximal subsequence starts in (start, mid] and ends after mid.

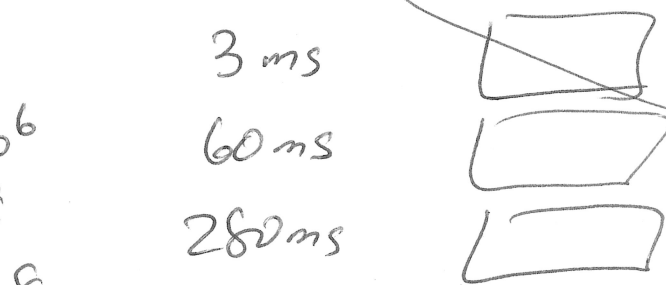
```
p s  int mess(int[] a) {
       int res = a[0]; cur = 0;
       for (int i = 0; i<a.length; i++)
           cur += a[i];
           if (cur > res) res = cur;
           if (cur < 0)  cur = 0;
       }
       return res;
}
```

O(n)

How to judge runtime experimentally

n	$f(n)=n^2$	t(n)
1000	10^6	3 ms
5000	$25*10^6$	60 ms
10000	10^8	280 ms
50000	$25*10^8$	⋮

$\frac{t(n)}{f(n)}$



$\frac{cn^2}{n^2} = c$

all same $O(n^2)$ is a good guess

increasing guessed too small

if decreasing guessed to big

Probability for the purpose of avg case run time analysis.

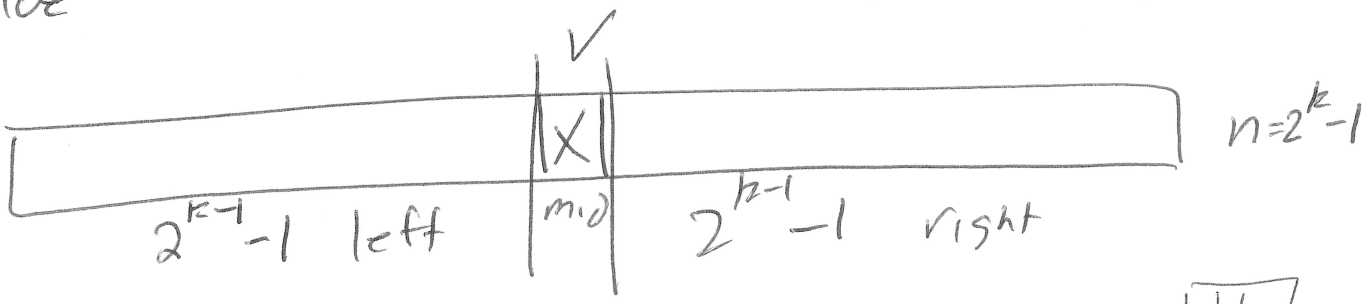
Discrete Random Variable is a variable that takes on different values w/ different probabilities

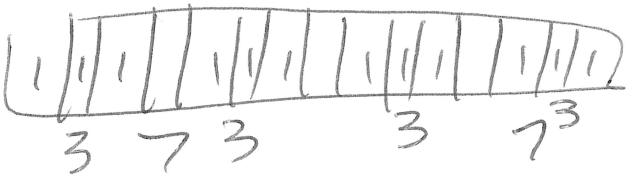
Out	
$2	1/36
$3	2/36
$4	3/36
$5	4/36
⋮	
$7	6/36
⋮	
$12	1/36

Avg income per ticker $= \frac{1}{36} \times \$2 + \frac{2}{36} \times \$3 + \cdots$

$$\sum_{x \in X} p_x \cdot x$$

for determining avg case run time make a chart of each poss run time + its associated probability then find its expected value





# steps	prob
1	$\frac{1}{n}$
2	$\frac{2}{n}$
3	$\frac{4}{n}$
⋮	
k	$\frac{2^{k-1}}{n}$

$n = 2^k - 1$

$$\text{Exp Runtime} = 1 \times \frac{1}{n} + 2 \times \frac{2}{n} + 3 \times \frac{4}{n} + \cdots k \times \frac{2^{k-1}}{n}$$

$$S = 1 \times 1 + 2 \times 2 \times 3 \times 2^2 + 4 \times 2^3 + \cdots k \times 2^{k-1}$$

$$-2S \qquad 1 \times 2 + 2 \times 2^2 + 3 \times 2^3 + \cdots (k-1) \times 2^{k-1} + k \cdot 2^k$$

$$-S = 1 + 2 + 2^2 + 2^3 + \cdots + 2^{k-1} - k \cdot 2^k$$

$$S = k \cdot 2^k - (1 + 2 + 2^2 + \cdots 2^{k-1})$$

$$S = k \cdot 2^k - (2^k - 1)$$

$$= k \cdot 2^k - 2^k + 1$$

$$= (k-1)2^k + 1 \quad , \; n = 2^k - 1$$

$n + 1 = 2^k$

$k = \log_2 n+1$

$$\text{Avg Runtime} = \frac{(k-1)2^k + 1}{2^k - 1} \sim \boxed{(k-1)} \quad O(\log n)$$