CS 445 Review

Number 1 from short answers

a) what if we resize by 1? Find runtime of add.

To do this make a table. We will use amortized cost.

Our table.

add #	size	cost
1	1	1
2	2	1 + 1 ← since we have to copy the original elements
3	3	1 + 2
4	4	1 + 3
5	5	1 + 4
⋮	⋮	⋮
n-1	n-1	1 + (n-2)
n	n	1 + (n-1)

The runtime is the sum of the cost column. We first sum all the 1's

$\underbrace{1+1+1+\dots+1}_{n \text{ times}} = O(n)$. Now the other side,

$1+2+3+\dots+n-2+n-1 = \frac{(n-1)(n)}{2} = O(n^2)$ ← Proof of this is not important

$O(n) + O(n^2) = O(n^2)$

Amortize by dividing by n.

$O\left(\frac{n^2}{n}\right) = O(n)$.

b) Double size of Array

Table will have one extra column

add #	size	cost	resize #
1	1	1	
2	2	1 + 1	0
3	4	1 + 2	1
4	4	1	
5	8	1 + 4	2
6	8	1	
7	8	1	
8	8	1	
9	16	1 + 8	3

add first

$\underbrace{1 + 1 + 1 + \ldots + 1}_{n \text{ times}} = O(n)$

$1 + 2 + 4 + \ldots + 2^i = \sum_{i=0}^{\log n} 2^i = O(\log n)$

$O(n) + O(\log n) = O(n)$

Amortized by dividing by n

$O(\frac{n}{n}) = O(1)$.

Recursion

Tail Recursion - Usually one recursive call at end of method. Ex:

```
factorial(n)
    if n == 1
        return 1
    return n * factorial(n-1)
```

Tail Recursive algorithms can be trivally translated to iterative algorithms.
- Usually done by compiler optimizations.

Divide and Conquer - use recursion to reduce problem size by some fraction with each recursive call.

Back tracking - Uses the advantage of the runtime stack to quickly find solutions to problems.

Ex] Sudoku Solvers
Path Finding Algorithms
Word Search Solvers.

General Pattern:

```
function solve(partSoln) {
    if (reject(partSoln)) return null
    if (isFullSolution(partSoln)) return partSoln
    attempt = extend(partSoln)
    while (attempt != null) {
        Soln = solve(attempt)
        if (soln != null) return Soln
        attempt = next(attempt);
    }
    return null;
}
```

Overheads of Recursion

- Takes time to create ARs
- Garbage collection
- many copies of local variables
- System calls to change program counter

when possible use iteration every time.

3 Requirements for recursion

1. Recursive case
2. Base case
3. Termination (Recursive case must lead to base case)

Ex of Divide and Conquer

Problem: Given B and n, determine B^n.

```
exp(int b, int n)
    if (n > 1)
    if (n > 2 && n%2 == 0)
        int temp = exp(b, n/2);
        return temp * temp;
    else if (n>1)
        return temp * temp * b;
    else
        return b;
```