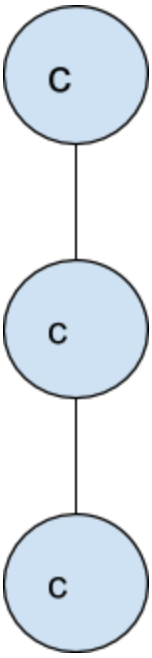
Recitation 11 (11/14 - 11/18)

Recurrence Relations - Special Case T(n) = T(n/2)+O(1)

Things to remember from last week:
q = number of subproblems
x = size of each subproblem
cn/x^{level} = work done for each subproblem
$q^j(cn/2^j) = (q/j)^j * cn$ = total about of work done at each level
$log_x n$ = levels of recursion

Case q = 1 : Binary Search

Each level in our unrolling will have no *q* value. Instead, it will just have time *c* plus the time for the previous.



T(n/2) + O(1) → Size of the input is halved every time but work at each level is constant (solved on page 243)

We see that the amount of work being done at each level is constant and there are $\log_2 n$ levels.

The critical question ends up being how many levels of constant work: **$O(\log_2 n)$**

(From last week)

Page 219: **T(n/2) + O(n)** (if we were combining back together)

At an arbitrary level *j*, there is one instance, size *n/2^j* and contributes *cn/2^j* to the running time.

$$T(n) \leq \sum_{j=0}^{\log_2 n-1} \frac{cn}{2^j} = cn \sum_{j=0}^{\log_2 n-1} \left(\frac{1}{2^j}\right)$$

Working out this geometric sum, we will find that it converges to 2.
Thus, we can say, for recurrence T(n/2) +O(n):

$$T(n) \leq 2cn = O(n).$$

Adversarial Lower Bound

Lower Bounds Previously: Show the lower bound for a *specific* algorithm
Goal of Adversarial Lower Bound: Show lower bound for *any* correct algorithm that solves the problem

Adversary - one's opponent in a contest or dispute

Algorithm ↔ Adversary

- algorithm wants answer to algo quickly
- the adversary is trying to come up with input that will make algorithm take a long time
- We want to show that adversary could force algorithm to do some amount of work which becomes the lower bound

Theorem

Every correct algorithm that solves the problem of searching in a sorted list needs Ω(logn) time.

There exist algorithms that work well for the specific case and otherwise are O(log n) in general. We want an algorithm whose lower bound is strictly $\Omega(\log n)$.

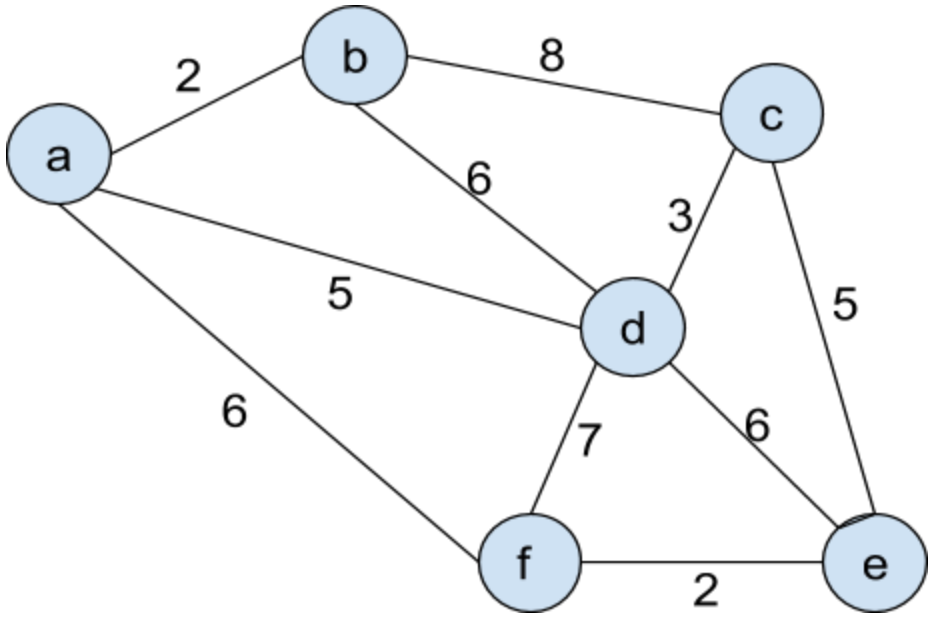
Discuss the two instances in the support pages where v is not in the input and where v is exactly the middle value of the input.

Example: Binary Search - Go over proof and reasoning after the video from below:

http://www-student.cse.buffalo.edu/~atri/cse331/support/lower-bound/index.html

Prim's Algorithm

- Similar in functionality to Dijkstra's
- Run example on graph below



Solution