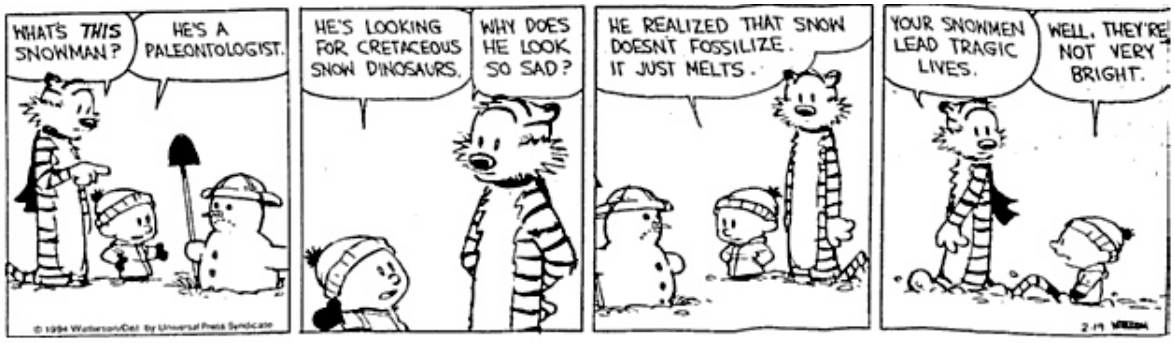
CS302 - Assignment 6

Due: Tuesday, Mar. 5 at the beginning of class
Hand-in method: paper



http://www.angelfire.com/wa/zzaran/calvin.html

For this assignment you must use latex to generate your work.

1. **[7 points]** Problem 9.3-9 (pg. 223) from the book. Average case linear is fine. You don't need to write full pseudocode, but clearly describe your solution and then you *must* show that your answer is correct.

   - *Hint 1:* Try drawing out a few examples and calculate the values for different configurations.
   - *Hint 2:* To show that your answer is correct, show that the pipeline length would increase if you move it either up or down from your chosen location (proof by contradiction).

2. **[7 points]** Linked Lists

   For each of the four types of lists in the following table, what is the asymptotic worst-case running time for each set operation listed? A *sorted* linked list is one in which the data is stored in sorted order. State any assumptions.

   - SEARCH: determine if k occurs in the list
   - INSERT: insert item k into the linked list
   - DELETE: given a reference to node n, delete node n from the linked list
   - PREDECESSOR: given a reference to node n, find the previous value (in sorted order) of the value associated with node n
   - SUCCESSOR: given a reference to node n, find the next value (in sorted order) of the value associated with node n

	unsorted, singly linked	sorted, singly linked	unsorted, doubly linked	sorted, doubly linked
SEARCH(k)				
INSERT(S, k)				
DELETE(S, n)				
SUCCESSOR(S, n)				
PREDECESSOR(S, n)				
MINIMUM(S)				
MAXIMUM(S)				

3. **[5 points]** Better than binary search?

   Binary search can determine if an element exists in a sorted array in $O(\log n)$ time. Prove that any algorithm that can only use comparisons (i.e. $>, <, =, ...$) must take $\Omega(\log n)$ steps, in particular when the element is not there (thereby also showing that binary search is asymptotically optimal).

4. **[14 points]** Halfsies

   In some situations, there is not a natural ordering to the data but we can check equality (e.g. images). Given an array of elements A, we would like to determine if there exists a value that occurs in more than half of the entries of the array. If so, return that value, otherwise, return *null*. Assume you can only check equality of elements in the array which takes time $O(1)$.

   Below are the beginnings of two divide and conquer approaches that attempt to solve this problem. Write pseudocode for each approach, argue that they are correct and state the running times.

   (a) Split the array in half and recurse.

   (b) Arbitrarily pair up the elements. If they're the same, keep one of the items. If they're different, discard both items. Repeat.