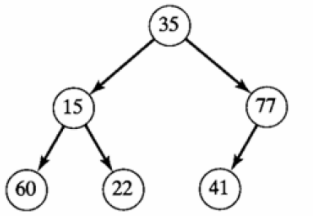
Announcements (0:00-12:00)

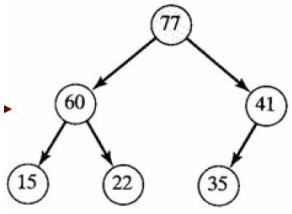
- In the news: Last night, Apple made an iPhone patch available to fix the hole that was hacked a few weeks ago. Just hours later, it was hacked into yet again.
- Regarding quiz 1, we understand that it was harder than quiz 0. No need to freak out. Just make sure you can go back over the solutions and understand how we would have solved the problems.
- Problem set 5, now on the website, challenges you to implement an efficient spellchecker. On the course website, you can find a "big board" that compares the speeds of different students' implementations. David's the only one there now; see if you can beat him!
- This week, help us help you. Think about the problem set in advance and submit a design document by Tuesday evening.

Heapsort (12:00-25:00)

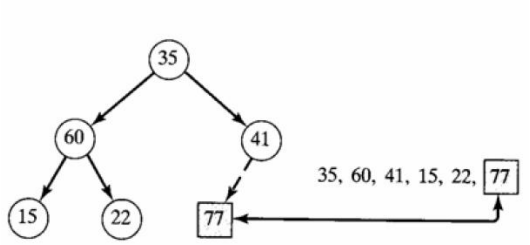
- What is the need of heapsort? As we will see, it runs in the same time as merge sort, but uses less memory. All we will need is a single array to represent our heap.
- Here's how it works. T
- Suppose we want to sort this list: 35 15 77 60 22 41
- Step 1: Drop them into a complete binary tree in that order, top to bottom, left to right



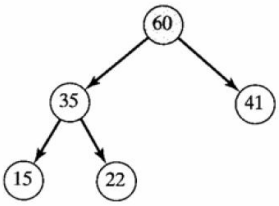
- Step 2: Heapify. For a detailed explanation of how we heapify an almost heap and how long this algorithm takes, see Monday's scribe notes.



- Step 3: Pluck them off in reverse sorted order.
    - Recall that the root node is the maximum element. Swap it with the bottommost rightmost node.
    - Pluck it off and put it in an array.

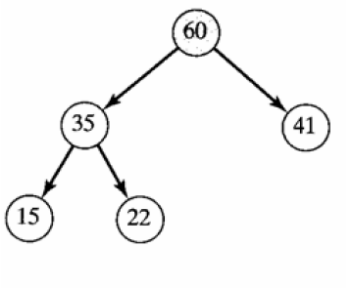


- The topmost element in the heap is now out of place. Bubble it down to re-heapify. The maximum element should now be at the top of the heap.



- Repeat until no there is a single node remaining, and stick it in the array.

- How long does this plucking algorithm take? Each time we re-heapify, we are really just performing a series of swaps on our new, out-of-place root node to get it in the right position. Since the tree is of height log n, there can be, at most, log n of these swaps.
- So each plucking entails at most log n steps. How many plucks do we perform? We do it for each element, so the number of plucks is equal to the number of elements, n. This gives an O(n log n) algorithm.
- There's also a startup cost of n log n to do the initial heapification. But asymptotically, we consider an algorithm of runtime 2 n log n to be O(n log n).
- How do we represent this using memory? There are two ways.
- Option 1: use the tree data structure we discussed last time in which each node is a structure containing a value and two pointers to children.
- Option 2: use an array that contains only the values of the nodes. Rather than pointers, you can use mathematics on the indices to get to a particular nodes' children.
- You can see how a complete binary tree "maps" to an array in the following diagram



60	35	41	15	22
0	1	2	3	4

- Observation: the children of a[i] are a[2i+1] and a[2i+2]
- You can use an array to represent a tree only when you are dealing with a complete binary tree (why?)

- As this shows, heapsort runs in the same time as mergesort, but can be performed with a single array.

This Is Jeopardy! (25:00-44:00)

- Who needs Alex Trebek when you've got David J. Malan?
- Check out Friday's lecture video for cs50's own version of America's Favorite Quiz Show ®
- In case you can't hear some of the contestants' responses, here are the q's and a's to go over on your own. But make sure you try to answer them yourself before looking at the answers on the next page! =)

1. (Week 2) Explain why a Vigenere cipher is harder to crack by trial and error than Caesar ciphers.
2. (Week 0) What is the largest decimal number that you can store with 8 bits?
3. (Week 1) What type does malloc() return by default?
4. (Week 2) What are the benefits of using global variable and what are the drawbacks?
5. (Week 3) What is the difference between selection and bubble sort?
6. (Week 4) What is the purpose of memoization?
7. (Week 0) What is wrong with the following?

```
if (x = 0)
   a=1;
```

8. (Week 1) Why would someone use a do-while loop?
9. (Week 4) Why is a program like GDB better to debug with than simply printf-ing?
10. (Week 1) What does associativity mean?
11. (Final Jeopardy) Decrypt this: pbeerpg

Answers

1. In a Caesar cipher, the key consists of exactly one letter, so there are only 26 different possible keys. This is small enough that an adversary could simply try every key to see which gives a sensible answer. In a Vigenere cipher, the key consists of n letters, for some n > 0, so there are 26^n different possible keys. Since the number of possible keys grows exponentially in n, this number very quickly becomes too large for the adversary to do an exhaustive search.
2. We can store $2^8 = 256$ different numbers, but since we include zero, the highest decimal number we can store is 255.
3. malloc() returns a void *. (A pointer of void type.)
4. The benefits of using a global variable are that you can access it from many different functions and even programs, and can avoid constantly needing to pass the same variables among functions. The drawback (besides losing design points!) is that a global variable lacks "locality." You may end up with multiple functions depending on it and therefore one another. Your program becomes a tangled web rather than a functional hierarchy. Bugs involving the global variable are difficult to isolate.

5. In bubble sort, we repeatedly walk through a list swapping elements that are side by side by out of order. After each walk through the list, we are guaranteed to have at least one more element in the right place. In selection sort, we repeatedly walk through the list to find the minimum, then swap that element with the first element. Again, we are guaranteed to have at least one more element in its correct position after each iteration. We also perform the same number of comparisons and end up with a $O(n^2)$ algorithm in both cases. However, in selection sort, we perform exactly one swap in each iteration whereas in bubble sort, we may perform up to n-1 swaps. (*Question*: what is the difference between selection sort and insertion sort?)
6. Memoization allows us to "remember" previously calculated values in a recursive algorithm. Recall that when we solve a problem using recursion, we solve the problem for n based on our solutions for numbers less than n. The fifth Fibonacci number, for instance, is the sum of the fourth and third Fibonacci numbers. But the fourth is the sum of the third and second Fibonacci numbers, and the third is the sum of the second and first Fibonacci numbers. As you can see, the same numbers are involved in multiple calculations. If we were not using a memo, we'd calculate each of these numbers as many times as we needed it in a calculation. For higher Fibonacci numbers, this becomes extremely inefficient, as we calculate lower Fibonacci numbers over and over again. But with a memo, we "write down" Fibonacci numbers as we calculate them, so when we need them in later calculations all we've got to do is fetch them from the array.
7. A single = is an assignment operator, so x = 0 assigns the value of 0 to the variable x. What we want is the logical operator ==, which tests the equality of x and 0.
8. You use a do-while loop to make the contents of the loop get executed once before the condition is tested. This is useful, for example, when fetching input from the user. (*Fun fact:* It is estimated that less than five percent all loops are do-whiles![1])
9. Print statements do not scale. The longer your program is, the more print statements you have to enter and then get rid of. This wastes your time. Print statements can also be difficult to decipher if you're not careful to include labels each time you print out a variable. In GDB, you go through step by step so you always know where you are in the program and what's being printed out.
10. Associativity is the left-to-right or right-to-left order for grouping operands to operators that have the same precedence. Precedence is the priority for grouping different types of operators with their operands.[2] For instance, if you say *p.x, the dot operator *associates* with the p, so this is interpreted as *(p.x) rather than (*p).x.
11. CORRECT. This was encrypted with a key of k = 13.

[1] Kernighan and Ritchie, 1978.
[2] http://publib.boulder.ibm.com/infocenter/macxhelp/v6v81/index.jsp?topic=/com.ibm.vacpp6m.doc/language/ref/clrc05preeval.htm