Midterm 2 practice problems

CS 133

October 17, 2018

1 Binary search

► Write a binary search over vectors, iteratively (i.e., using a loop), with the following prototype:

```
int binary_search(const vector<int>& vs, int target);
```

► Write a binary search over vectors, recursively, with the following prototype:

```
int binary_search(const vector<int>& vs, int target, int start, int end);
```

You can assume that this will be called with `start = 0` and `end = vs.size()-1`.

► Write a binary search over arrays, iteratively, with the following prototype (returning `nullptr` if the target element is not found):

```
int* binary_search(int* begin, int* end, int target);
```

► Write a binary search over arrays, recursively, with the following prototype (returning `nullptr` if the target element is not found):

```
int* binary_search(int* begin, int* end, int target);
```

► Assume that we run a binary search over a vector of size 100. What is the maximum number of steps (i.e., loop iterations or recursive calls) that could possibly be required for this input?

2 Sorting

▶ Write selection sort, iteratively. (Find the smallest element, move it to the beginning. Repeat until there are no unsorted elements left.)

▶ Write selection sort, recursively. Use the following prototype:

```
void selection_sort(int* begin, int* end);
```

(To do this, after finding the minimum, swap it with `*begin` and then recurse with `begin + 1`. The base case is when `begin >= end`.)

▶ Write insertion sort, iteratively. (For each unsorted element, inserted it into the sorted portion of the array until none are left.)

▶ Write bubble sort, iteratively. (Compare adjacent pairs of elements, swapping if they are out of order. After one pass, the largest element will have "bubbled" up to the end of the array. Repeat with remaining elements until everything is sorted.)

▶ Write the `merge` function that underlies the MERGESORT algorithm. Use the following prototype:

```
void merge(vector<int> input, int mid, vector<int>& output);
```

You may assume that `output.size() == input.size()` initially.

▶ Write the in-place `partition` function that underlies the QUICKSORT algorithm.

▶ Why is the choice of the *pivot* important in QUICKSORT? Why are the first and last elements of the input particular *bad* choices for the pivot?

▶ What is the best-case big-O complexity of Mergesort? What is the worst case complexity? What is the best-case big-O complexity of Quicksort? What is the worst case?

3 Binary search trees

For any problems that need it, you can assume the following defintion of `node`:

```
struct node {
    int key;
    node* left;
    node* right;
    node* parent;
}
```

▶ State the binary search tree order property.

▶ For a binary search tree containing n elements, what is the *best case* runtime complexity of a `find` operation? What is the *worst* case complexity? When do these occur?

▶ The worst-case unbalanced binary search tree is equivalent to a linked list. Write a sequence of 10 integer values that, when inserted into an empty tree using the standard `insert` function, will create a degenerate tree.

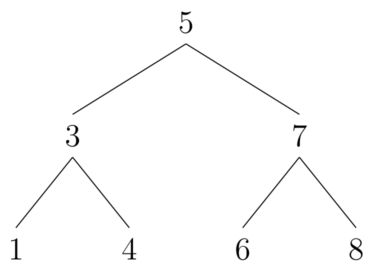
▶ Draw the tree that will result from inserting the following elements into an empty tree, using the standard (no rebalancing) `insert` function.

$$1 \quad 2 \quad 7 \quad 3 \quad 0 \quad 5 \quad -4$$

▶ Insert the above into an AVL tree.

▶ Insert the above into a splay tree.

▶ Here is a binary search tree:



Draw the tree that will result from deleting 5 from the tree, using the standard (no rebalancing) `remove` function; use successor replacement.

▶ Starting from the above tree, draw the tree that will result if we perform a rotation at node 3.

▶ Starting from the above tree, draw the tree that will result if we perform a rotate at node 7.

▶ Starting from the above tree, draw the tree that would result if we did a splay-`find` for node 6.

▶ Given a binary search tree, how do we find the *largest* element? Sketch a function

```
node* largest(node* root);
```

that would do so.

▶ Given the above binary search tree, list the elements as they would be processed in an *inorder*, *preorder*, and *postorder* traversal:

Inorder:

Preorder:

Postorder:

Levelorder:

4 Binary heaps

▶ State the heap order property, for max-heaps.

▶ Does the following tree represent a valid max heap? If not, why not?

```
    7
   / \
  5   6
 / \   \
3   4   1
```

▶ Draw the heap that would result from inserting the following elements in left-to-right order into an empty max-heap.

$$1 \quad 2 \quad 7 \quad 3 \quad 0 \quad 5 \quad -7 \quad 4$$

▶ Here is a max heap:

```
     7
   /   \
  5     6
 / \   / \
1   3 2   4
```

Draw the heap that would result after a single `extract` operation.

▶ Given a node in a heap with an array-based representation at index n (1-based indexing), what is the formula for the *parent* of n? For the *left* and *right* children?

▶ How can we use a max heap to sort a sequence of values? Sketch the Heapsort algorithm.