1 Administrative Topics

• We return the quiz.

2 Searching

Suppose we have a list of all the grades received in a particular class. Those grades are stored in ascending order. We want to know if a particular grade (3.3) was given. How could we find out if it is in the list?

There are two common methods for searching:

• Linear Search: Look at every element until we find the one we want.

• Binary Search: Take advantage of the fact that the list is sorted, and repeatedly cut the search space in half every time we examine one element.

Let’s look at the simplest one first.
2.1 Linear Search

Suppose you have a list of numbers. You want to learn whether or not 42 is in the list. The strategy is to look at the items one at a time. If we get to the end of the list, we know it isn’t there. If the list is sorted, then we can stop sooner. We can stop once we have either reached 42 (and we have found it!) or the first number greater than 42 (42 isn’t in the list!)

Here is the code for the version that doesn’t require it to be sorted.

```python
# return True if goal is in the list
def linear_search( the_list, goal ):
    for item in the_list:
        if item == goal:
            return True
    return False
```

Here is the code for the version that assumes it has been sorted.

```python
# return True if goal is in the list, given
# that the list is in ascending order
def linear_search_sorted( the_list, goal ):
    for item in the_list:
        if item == goal:
            return True
        if item > goal:
            return False
    return False
```

- Pros: Linear search is simple and doesn’t require the list to be sorted.
- Cons: You need to examine many elements of the list (possibly all of them). This algorithm scales with the number of elements in the list (if list A is twice as long as list B, then searches in list A will be, on average, twice as long as searches in list B). We call this “order N” (where N is the number of elements in the list).
2.2 Binary Search

If the list is sorted, we can search much more efficiently. The idea is to look at the element in the middle of the list. If the goal is to its left, then we can search just the first half of the list. We then repeat the process for this half-list. We continue until we have found the goal in the list, or the sub-list has gotten too short.

We keep track of the indices of the list we are looking at. We begin with the entire list, so left_idx is set to 0 and right_idx is set to \( N - 1 \) (where \( N \) is the length of the list). We compute the index of the middle element. If we are at the goal, then we return. Otherwise, we determine which half of the list to restrict the next step of our search to. We keep restricting the search area until it becomes too small.

```python
# return True if goal is in the list, given
# that the list is in ascending order
def binary_search( the_list, goal):
    N = len( the_list )
    left_idx = 0
    right_idx = N-1
    while left_idx <= right_idx:
        mid_idx = (left_idx + right_idx)/2
        if the_list[mid_idx] == goal:
            return True
        if the_list[mid_idx] < goal:
            left_idx = mid_idx + 1
        else: # the_list[mid] > goal:
            right_idx = mid_idx - 1
    return False
```

For more information on the binary search, check out the article on Wikipedia (https://en.wikipedia.org/wiki/Binary_search_algorithm).

- Pros: This search is fast. The algorithm scales with the log (base 2) of the number of elements of the list.
- Cons: This search requires the list to be sorted. Sorting takes longer than a linear search, so it will be worth it to sort first only if we think we need to search often (or if it is easy to construct the list in sorted order to begin with).
3 Including Code Snippets in your write-up

We looked at a couple of examples in class.

When including code snippets in your write-up, you should use them to make your project as easy to understand as possible. Here are some guidelines for determining whether or not your write-up is properly including code snippets. You should be able to answer “yes” to all these questions:

1. Does the text describe the context of the code snippet within the code?
2. Does the text describe the purpose of the code snippet (i.e. what your goal was when you wrote the code)?
3. Does the text describe the overall strategy taken by the code snippet? (e.g. loop over the list so that each element could be capitalized)
4. Does the code snippet itself make the write-up clearer than it would be without the code snippet? (Does the specific example help?)
5. Is the code snippet as short as possible? (It should be possible to make your point with 5-10 lines of code).