Section 07: Sorting and Design Decisions

1. Design Decisions

We've now talked about a few ADTs and a variety of data structures. For each of the following scenarios:

(a) Say which ADT you think most closely corresponds to what you want to do with the data.

(b) Describe a data structure you would use to implement that ADT.

(c) Justify your decisions

For these questions we care about the justification. In some cases there may be more than one "right" ADT or data structure.

(a) You are running a gaming server, where each player has a rating. Each time a player makes a `NewGame` request, you should be able to suggest another player who has a similar rating.

(b) Your project manager wants to make her job of approving code changes easier. Each push to your code base comes with an automatically generated number, its `magnitude` –an integer representing the size of the change made. Your PM wants to be able to do two things

(i) `reviewBiggestChange` finds the unapproved change of largest `magnitude`

(ii) `approveSmallChanges(k)` given a number k, automatically approves all changes to the code base of `magnitude` less than k.

(c) In your user-updated encyclopedia, when someone types certain keywords into the searchbar, they should be taken directly to an associated article (instead of to a search results page). You need to store a list of keywords along with the page that each keyword will redirect to.

(d) You're running a restaurant for cats. Normally, your restaurant is first-come, first-serve: you want to feed the cats in the order they arrive. But if some of the cats waiting in line are really hungry, they will disturb the other customers by meowing loudly, so you also want to be able to find the hungriest kitties and feed them first.

2. Sorting

(a) Demonstrate how you would use quick sort to sort the following array of integers. Use the first index as the pivot; show each partition and swap.

[6, 3, 2, 5, 1, 7, 4, 0]

(b) Show how you would use merge sort to sort the same array of integers.

3. Sorting Decisions

For each of the following scenarios, say which sorting algorithm you think you would use and why. As with the design decision problems, there may be more than one right answer.

(a) Suppose we have an array where we expect the majority of elements to be sorted "almost in order". What would be a good sorting algorithm to use?

(b) You are writing code to run on the next Mars rover to sort the data gathered each night (Think about sorting with limited memory and computational power).

(c) You're writing the backend for the website `SortMyNumbers.com`, which sorts numbers given by users.

(d) Your artist friend says for a piece she wants to make a computer sort every possible ordering of the numbers $1, 2, \ldots, 15$. Your friend says something special will happen after the last ordering is sorted, and you'd like to see that ASAP.

4. Memory – Short Answer

(a) What are the two types of memory locality?

(b) Does this more benefit arrays or linked lists?

5. Memory – In Context

(a) Based on your understanding of how computers access and store memory, why might it be faster to access all the elements of an array-based queue than to access all the elements of a linked-list-based queue?

(b) Why might `f2` be faster than `f1`?

```
public void f1(String[] strings) {
    for (int i=0; i < strings.length; i++) {
        strings[i] = strings[i].trim();        // omits trailing/leading whitespace
    }
    for (int i=0; i < strings.length; i++) {
        strings[i] = strings[i].toUpperCase();
    }
}

public void f2(String[] strings) {
    for (int i=0; i < strings.length; i++) {
        strings[i] = strings[i].trim(); // omits trailing/leading whitespace
        strings[i] = strings[i].toUppercase();
    }
}
```

(c) Consider the following code:

```
public static int sum(IList<Integer> list) {
    int output = 0;
    for (int i = 0; i < 128; i++) {
        // Reminder: foreach loops in Java use the iterator behind-the-scenes
        for (int item : list) {
            output += item;
        }
    }
    return output;
}
```

You try running this method twice: the first time, you pass in an array list, and the second time you pass in a linked list. Both lists are of the same length and contain the exact same values.

You discover that calling `sum` on the array list is consistently 4 to 5 times faster then calling it on the linked list. Why do you suppose that is?

(d) Suppose you are writing a program that iterates over an `AvlTreeDictionary` – a dictionary based on an AVL tree. Out of curiosity, you try replacing it with a `SortedArrayDictionary`. You expect this to make no difference since iterating over either dictionary using their iterator takes worst-case $\Theta(n)$ time.

To your surprise, iterating over `SortedArrayDictionary` is consistently almost 10 times faster!

Based on your understanding of how computers organize and access memory, why do you suppose that is? Be sure to be descriptive.

(e) Excited by your success, you next try comparing the performance of the `get(...)` method. You expected to see the same speedup, but to your surprise, both dictionaries' `get(...)` methods seem to consistently perform about the same.

Based on your understanding of how computers organize and access memory, why do you suppose that is?

(Note: assume that the `SortedArrayDictionary`'s `get(...)` method is implemented using binary search.)