6.16. Programming Exercises

1. Set up a random experiment to test the difference between a sequential search and a binary search on a list of integers.
2. Use the binary search functions given in the text (recursive and iterative). Generate a random, ordered list of integers and do a benchmark analysis for each one. What are your results? Can you explain them?
3. Implement the binary search using recursion without the slice operator. Recall that you will need to pass the list along with the starting and ending index values for the sublist. Generate a random, ordered list of integers and do a benchmark analysis.
4. Implement the `len` method (__len__) for the hash table Map ADT implementation.
5. Implement the `in` method (__contains__) for the hash table Map ADT implementation.
6. How can you delete items from a hash table that uses chaining for collision resolution? How about if open addressing is used? What are the special circumstances that must be handled? Implement the `del` method for the `HashTable` class.
7. In the hash table map implementation, the hash table size was chosen to be 101. If the table gets full, this needs to be increased. Re-implement the `put` method so that the table will automatically resize itself when the loading factor reaches a predetermined value (you can decide the value based on your assessment of load versus performance).
8. Implement quadratic probing as a rehash technique.
9. Using a random number generator, create a list of 500 integers. Perform a benchmark analysis using some of the sorting algorithms from this chapter. What is the difference in execution speed?
10. Implement the bubble sort using simultaneous assignment.

(DiscussionQuestions.html) (../Trees/toctree.h

11. A bubble sort can be modified to “bubble” in both directions. The first pass moves “up” the list, and the second pass moves “down.” This alternating pattern continues until no more passes are necessary. Implement this variation and describe under what circumstances it might be appropriate.

12. Implement the selection sort using simultaneous assignment.

13. Perform a benchmark analysis for a shell sort, using different increment sets on the same list.

14. Implement the `mergeSort` function without using the slice operator.

15. One way to improve the quick sort is to use an insertion sort on lists that have a small length (call it the “partition limit”). Why does this make sense? Re-implement the quick sort and use it to sort a random list of integers. Perform an analysis using different list sizes for the partition limit.

16. Implement the median-of-three method for selecting a pivot value as a modification to `quickSort`. Run an experiment to compare the two techniques.

user not logged in

(DiscussionQuestions.html)

(../Trees/toctree.h