Homework 1: Linear Search

Instructor: Mehmet Emre

CS 32 Spring '22

Due: 4/6 12:30pm
Name & Perm #: Bharat Kathi (5938444)
Homework buddy (leave blank if you worked alone):

1 (10 pts)

Before attending your first discussion section (lab), please ensure that you can access your CSIL / College of Engineering Account. Either visit CSIL (the lab on first floor ocean side of Harold Frank Hall), and try to login, or use an ssh client to see if you can access csil.cs.ucsb.edu.

If you need help creating or accessing your CoE/CSIL account, visit `https://accounts.engr.ucsb.edu`

Then, write your College of Engineering account username below. DO NOT WRITE YOUR PASSWORD. DO NOT EVER WRITE YOUR PASSWORD!!

Answer: bkathi

Course Textbooks

- **PS9**: *Problem Solving with C++, 9th Edition* by Walter Savitch (used in CS16 - Purple/Blue cover. There is a newer edition available, but I don't expect everyone to buy a new edition if you already have the 9th edition from when you took CS 16).
- **DS4**: *Data Structures and Other Objects Using C++, 4th edition* by Michael Main and Walter Savitch (used in some iterations of CS24 - Brown cover)
- **OSTEP**: *Operating Systems: Three Easy Pieces* by Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau (only some chapters, available online)

Both books are required for this course. OSTEP is available legally and for free online. This homework will only refer to DS4 (the brown book), but future homework assignments may draw on either or both of the two books.

Reading: In DS4 read pages 584-594.

Also, read the "C++ Feature" box about `size_t` on p. 100. You only need to read the part in the sans-serif font beside the "C++ Feature" logo, but if you read a few extra lines of the main text, you'll see better how `size_t` is used. Since `size_t` is used in Section 12.1, it's helpful to review this material from Chapter 3 in case you glossed over it during CS24.

Then, answer the questions on the remaining pages of this homework.

Questions about DS4, pages 584-586:

2 (10 pts)

Serial search is also called linear search. Suppose you are writing a standalone C++ function called `linearSearch` that searches through an unsorted C++ builtin array array of int values using this technique. Write the function prototype for this function—ONLY the function prototype. Be sure to include the name and type all the input parameters you would need, and a suitable type for the return value. Assume the function returns the index of the value if found, and -1 if the value is not found. Choose parameter names that make sense and represent good programming style (i.e. clearly document the purpose of the parameters that are being passed in.)

Answer: .

```
int linearSearch(int arr[], int val, int size);
```

3 (10 pts)

For this question, please review pp 584-586 and read both sides of the handout (`http://www.cs.ucsb.edu/~emre/cs32/hw/1-handout.pdf`). Then answer this question about the expected number of array accesses for linear search.

- Suppose there are nine array elements, `a[0]` through `a[8]`.
- Each of the odd elements, `a[1]`, `a[3]`, `a[5]`, and `a[7]` has a 10% chance (0.1) of being the one searched for.
- Each of the even elements `a[0]`, `a[2]`, `a[4]`, `a[6]` and `a[8]` has a 5% change (0.05) of being the one searched for.
- With probability 35% (0.35), the element being sought is not found in the array.

In this case, what is the expected number of array accesses?

Answer: .

$E[X] = 0.1(2+4+6+8) + 0.05(1+3+5+7+9) + 0.35(10) = 6.4$

In CS24, you should have learned about big-O analysis, i.e. asymptotic worst-case run-time analysis.

4 (10 pts)

What is the big-O analysis of a find(key) operation on a sorted array of n keys, implemented using *binary search*? Circle your answer.

Note that log, in the context of big-O, typically means log based 2, but since all logs are related by a constant factor, i.e. for any value of b, there is a constant C such that $\log_2(x) = C \log_b(x)$.

$$O(1) \quad O(\log n) \quad O(n) \quad O(n \log n) \quad O(n^2) \quad O(n^3)$$

Answer: .

O(log n)

5 (10 pts)

Explain why this is the running time for binary search. A "formal proof" is not required—just a *brief* intuitive explanation that shows you understand why this is the running time.

Answer: .

Binary searching seperates itself from the standard linear search by eliminating half of the values in an array each step rather than just eliminating one. The first step involves checking the middle element. Each succesive step eliminates half the remaining elements since we only have to check the next middle element greater than or less than our initial middle element (depending on what our search value is).

For example, if we have an array with 16 elements, the worst case search results in 4 steps. If we have a 32 element array, the worst case steps is 5. The length of the array doubled but the steps only increased by one, therefore giving us a logarthimic complexity of O(log n).