UNIVERSITY OF CALIFORNIA, MERCED

BERKELEY · DAVIS · IRVINE · LOS ANGELES · MERCED · RIVERSIDE · SAN DIEGO · SAN FRANCISCO     SANTA BARBARA · SANTA CRUZ

# CSE 15: Discrete Mathematics Laboratory 8

Fall 2020

- **This lab assignment will be graded and its grade will count towards the course grade**.
- **This lab must be solved individually.** You can discuss your ideas with others, but when you prepare your solution you must work individually. Your submission must be yours and yours only. No exceptions, and be reminded that the CSE academic honesty policy discussed in class will be enforced.
- Your solution must be exclusively submitted via CatCourses. Pay attention to the posted deadline because **the system automatically stops accepting submissions when the deadline passes. Late submissions will receive a 0**. You can upload one or more `.py` files.
- **Important**: in this assignment you have to not only submit your code, but also enter a table in catcourses (see instructions in question 3).
- Start early.

## Comparing the performance of search algorithms

In class we have seen two different algorithms for searching an element in a sequence, i.e., the *linear search* algorithm and the *binary search* algorithm. Recall that the two algorithms have different time complexity, i.e., $O(n)$ versus $O(\log n)$, where $n$ is the number of elements in the sequence. Also recall that binary search can only be applied to sequences that are sorted, while linear search can be applied to any sequence. In this lab you will implement both of them and then compare their performance for *large inputs*. In this assignment we use the python lists to represent sequences.

1. write a function `linear_search(s,k)` that takes as parameters a list of integers `s` and an intger `k` and determines if `k` is in `s` or not. If `k` is in `s`, the function returns the first index `i` such that `s[i]=k`. If `k` is not in `s`, then the function returns -1. The function should implement the linear search algorithm presented in class.
2. write a function `binary_search(s,k)` that takes as parameters a sorted list of integers `s` and an intger `k` and determines if `k` is in `s` or not. If `k` is in `s`, the function returns an index `i` such that `s[i]=k`. If `k` is not in `s`, then the function returns -1. The function should implement the binary search algorithm presented in class.
   When implementing this function, refer to the slides given in class and keep in mind the following:
   (a) lists in python a 0-indexed, i.e., the first element is at position 0, not at position 1. This means you have to adapt line two in the pseudocode.
   (b) the algorithm uses the operator $\lfloor\ \rfloor$ to round down the result of a division. To do that you can use python's builtin operator `//`.

(c) if `s` is a list of integers, you can sort it by calling the member function sort, i.e., `s.sort()` will rearrange the elements in `s` so that they are sorted in ascending order.

(d) **important**: download the latest version of the slides, as there is a small change in the pseudocode for `binary_search`.

3. After implementing the two functions, measure how their performance changes with the size of the input. To this end, copy paste the content of the provided file `generatedata.py` at the top of your implementation. The provided code includes the function `gen_random_list(n)` that accepts one parameter and returns a random list of non-negative integers. Use that function to generate lists of increasing lengths ($10^2, 10^3$, all the way to $10^9$) and measure how the algorithms perform in the worst-case. To ensure that the worst case is encountered, search for a negative number because the lists will not include negative numbers.
   How do you measure the time spent by an algorithm? Add the line `import time` at the beginning of your code. Then, if you want to measure the time spent by calling a function `myfunction`, you can use the following code:

```
start_time = time.perf_counter()  # records current time
myfunction() # do something
spent_time = time.perf_counter() - start_time # difference between current and past
print("Time spent calling myfunction: ", spent_time) #time spent (in seconds)
```

   After you are done, tabulate the results for the two algorithms and include them as part of your solution in CatCourses. Explain the results you obtain.

   *Note*: if your computer takes too much time when the sequences are too long, you can stop before $10^9$.