SIENAcollege
Computer Science

Computer Science 385
Design and Analysis of Algorithms
Siena College
Spring 2018

Lab 4: Brute-Force Algorithms

Due: Start of your next lab session

You will be assigned a partner to work with on this lab. Only one submission per group is needed.

Group members: _____

Learning goals:

1. to practice using the definitions of Big O, Big Ω, and Big Θ
2. to use empirical methods to study the behavior of algorithms
3. to review the brute-force sequential search algorithm and its worst and best case running times
4. to gain additional experience with exhaustive search algorithms

---

Getting Set Up

You will receive an email with the link to follow to set up your GitHub repository, which will be named `bruteforce-lab-yourgitname`, for this Lab. One member of the group should follow the link to set up the repository on GitHub, then that person should email the instructor with the other group members' GitHub usernames so they can be granted access. This will allow all members of the group to clone the repository and commit and push changes to the origin on GitHub.

Submitting

Once all written items are initialed to indicate completion, turn in one copy of this handout. Be sure names of all group members are clearly on the first page.

Your submission requires that all required deliverables are committed and pushed to the master for your repository's origin on GitHub. That's it! If you see everything you intend to submit when you visit your repository's page on GitHub, you're set.

Grading

Score: ____ / 100

For the first three questions, prove each assertion using the definitions of $O(n)$, $\Omega(n)$, and $\Theta(n)$, using small constants when possible.

Question 1:
$(2n^3 + 5n + 10000)^2 \in O(n^7)$ (3 points)

Question 2:
$12n^2 \log n \in \Omega(n^2)$ (3 points)

Question 3:

$94n^3 + 4n^2 - 997 \in \Theta(n^3)$ (6 points)

In class, we looked at the brute-force algorithm for finding the convex hull of a set of points. The `BruteForceConvexHull` program applies this algorithm to the the vertices of a METAL highway graph. Today, you will perform an empirical analysis study using this program. A version of this program is in your repository, as is a selection of METAL graph files you will use in the study.

Question 4:

Instrument the program so it measures the time taken to compute the convex hull, and so it counts the number of times it computes the `checkVal` variable in the computation loop. Be careful not to include the time taken to load the graph, or the time taken to print the results. The Java method `System.currentTimeMillis()` will come in handy here. (15 points)

Question 5:

Modify the code so that when the second command-line parameter given is "timings" your program prints a single line containing the name of the graph file, the number of vertices it contains, the time it took to compute the convex hull for that graph, and the number of times `checkVal` was computed for that graph. (5 points)

Question 6:

Generate timings using the modified program for each of the provided graphs. Load this data into a spreadsheet like Excel and create a plot of problem size ($|V|$) *vs.* the number of `checkVal` computations. Also compute (in the spreadsheet) and plot $|V|^2$ and $|V|^3$. Be sure your data is sorted by problem size, and plot using an XY Scatter chart. Include your spreadsheet in your GitHub repository. (20 points)

Question 7:

Do the results of this empirical study match your expectations based on this algorithm's theoretical complexity? Explain (5 points)

Question 8:

What does this say about how this algorithm performs relative to its best and worst cases on this specific set of data? (5 points)

Question 9:

Based on the trends you see in your data, about how many computations of `checkVal` and how long would you expect this program to take using the full `tm-master.tmg` graph, which contains about 350,000 vertices? Show your work. (5 points)

Question 10:

The brute-force **sequential search** algorithm searches for a *key* element k in an array of elements by starting at the beginning of the array and scanning through sequentially element by element until it finds k, or it reaches the end of the array without finding it. Below is pseudocode for sequential search with two parts missing. Study the code and then fill in the boxes with the missing pseudocode. (4 points)

ALGORITHM SEQUENTIALSEARCH(A, k)
//Input: an array $A[0..n-1]$
//Input: a key element k
//Output: The index of the first occurrence of k in A
$i \leftarrow 0$
while $i < n$ **and** [ ] **do**
$i \leftarrow i + 1$
if $i < n$ **then**
return i
else
return [ ]

Question 11:

For an n element input array, exactly how many *key comparisons* does the algorithm make in the worst case? (A key comparison is any comparison made between the key k and an element of the array. The number of key comparisons often determines the asymptotic running time of searching algorithms, which is why we count them.) (4 points)

Question 12:

The best-case efficiency of an algorithm is not nearly as important as the worst-case efficiency. But it is not completely useless either. For example, some sorting algorithms perform best on nearly sorted data, which is a common situation in practice. For an n element array, exactly how many key comparisons does sequential search make in the best case? (4 points)

Question 13:

A simple trick is often used in implementing sequential search: if the key k is added to the end of the array (assume there is space for it), then the search will find the key there (for an unsuccessful search), if not sooner (for a successful search). The advantage of this is that we can eliminate the check $i < n$ in each iteration of the **while** loop. Write pseudocode for this version of sequential search below. (6 points)

ALGORITHM SEQUENTIALSEARCH(A, k)
//Input: an array $A[0..n-1]$ and extra slot $A[n]$ to store k
//Input: a key element k
//Output: The index of the first occurrence of k in A

Question 14:

This problem is from the Nintendo DS game Professor Layton and the Diabolical Box, which is full of puzzles/problems of the type we encounter regularly in algorithms. In the picture below, collect all of the mushrooms in the forest as you pass through. Each circular clearing on the map contains mushrooms. You don't want to spend too long in this creepy forest though, so find a route through that visits each clearing only once. Stay on the roadways. (4 points)

Question 15:

Suppose you were to solve the mushroom problem above using a *very* exhaustive search. *I.e.*, for every possible ordering of the 18 mushrooms (not counting the start and goal mushrooms), check if the roadways exist that allow them to be visited in that order. In the worst case, how many different orderings would need to be checked? (5 points)

Question 16:

If your computer can check 1 billion orderings per second, how long would this exhaustive search approach take in the worst case? (4 points)

Question 17:

What well-known problem in computer science is illustrated in this puzzle? (2 points)