CSE 321 - Introduction to Algorithm Design
Homework 02

Deadline: 23:55 November 7th, 2016

1. Sort the array A = {3, 44, 38, 5, 47, 15} in increasing order by selection sort, bubble sort, insertion sort, quick sort. Show steps of sorting algorithms as well.

[SOLUTION]

Check your answer step by step at https://visualgo.net/sorting

2. Briefly explain your answers for questions below.

a) Is selection sort stable?

b) Is bubble sort stable?

c) Is it possible to implement selection sort for linked lists with the same Θ(n2) efficiency as the array version?

d) Is it possible to implement insertion sort for sorting linked lists? Will it have the same $O(n^2)$ efficiency as the array version? Recall that we can access elements of a singly linked list only sequentially.

[SOLUTION]

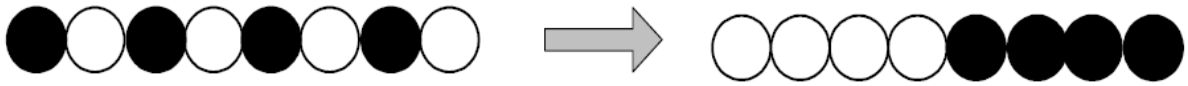
a) Selection sort is not stable: In the process of exchanging elements that are not adjacent to each other, the algorithm can reverse an ordering of equal elements. The list 2'', 2', 1 is such an example.

b) Bubble sort is stable. It follows from the fact that it swaps adjacent elements only, provided A[j +1] < A[j].

c) Yes. Both operations–finding the smallest element and swapping it–can be done as efficiently with a linked list as with an array.

d) Yes, but we will have to scan the sorted part left to right while inserting A[i] to get the same $O(n^2)$ efficiency as the array version.

3. *Alternating disks*: You have a row of 2n disks of two colors, n dark and n light. They alternate: dark, light, dark, light, and so on. You want to get all the dark disks to the right-hand end, and all the light disks to the left-hand end. The only moves you are allowed to make are those which interchange the positions of two neighboring disks.



Design an algorithm for solving this puzzle and determine the number of moves it makes. Hint: Thinking about the puzzle as a sorting-like problem may and may not lead you to the most simple and efficient solution.

[SOLUTION]

Here is a simple and efficient (in fact, optimal) algorithm for this problem: Starting with the first and ending with the last light disk, swap it with each of the i $(1 \le i \le n)$ dark disks to the left of it. The ith iteration of the algorithm can be illustrated by the following diagram, in which 1s and 0s correspond to the dark and light disks, respectively.

$$\underbrace{00..0}_{i-1}\underbrace{11..1}_{i-1}\mathbf{1}0..10 \quad \Rightarrow \quad \underbrace{00..0}_{i}\mathbf{0}\underbrace{11..1}_{i}10..10$$

The total number of swaps made is equal to $\sum_{i=1}^{n} i = n(n+1)/2$.

4. Give an example of a text of length n and a pattern of length m that constitutes the worst-case input for the brute-force string-matching algorithm. Exactly how many character comparisons are made for such input? Hint: It will suffice to limit your search for an example to binary texts and patterns.

[SOLUTION]

The text composed of n zeros and the pattern $\underbrace{0 \ldots 0}_{m-1}1$ is an example of the worst-case input. The algorithm will make $m(n-m+1)$ character comparisons on such input.

5. Consider the problem of counting, in a given text, the number of substrings that start with an A and end with a B. For example, there are four such substrings in CABAAXBYA.

a) Design a brute-force algorithm for this problem and determine its efficiency class.

b) Design a more efficient algorithm for this problem.

[SOLUTION]

a. Note that the number of desired substrings that starts with an A at a given position i $(0 \leq i < n-1)$ in the text is equal to the number of B's to the right of that position. This leads to the follwing simple algorithm:

Initialize the count of the desired substrings to 0. Scan the text left to right doing the following for every character except the last one: If an A is encountered, count the number of all the B's following it and add this number to the count of desired substrings. After the scan ends, return the last value of the count.

For the worst case of the text composed of n A's, the total number of character comparisons is

$$n + (n-1) + ... + 2 = n(n+1)/2 - 1 \in \Theta(n^2).$$

b. Note that the number of desired substrings that ends with a B at a given position i $(0 < i \leq n-1)$ in the text is equal to the number of A's to the left of that position. This leads to the follwing algorithm:

Initialize the count of the desired substrings and the count of A's encountered to 0. Scan the text left to right until the text is exhausted and do the following. If an A is encountered, increment the A's count; if a B is encountered, add the current value of the A's count to the desired substring count. After the text is exhausted, return the last value of the desired substring count.

Since the algorithm makes a single pass through a given text spending constant time on each of its characters, the algorihtm is linear.

6. *Ferrying soldiers*: A detachment of n soldiers must cross a wide and deep river with no bridge in sight. They notice two 12-year-old boys playing in a rowboat by the shore. The boat is so tiny, however, that it can only hold two boys or one soldier. How can the soldiers get across the river and leave the boys in joint possession of the boat? How many times need the boat pass from shore to shore? Hint: Solve the problem for n=1.

[SOLUTION]

First, the two boys take the boat to the other side, after which one of them returns with the boat. Then a soldier takes the boat to the other side and stays there while the other boy returns the boat. These four trips reduce the problem's instance of size n (measured by the number of soldiers to be ferried) to the instance of size $n-1$. Thus, if this four-trip procedure repeated n times, the problem will be solved after the total of $4n$ trips.

7. *Celebrity problem*: A celebrity among a group of n people is a person who knows nobody but is known by everybody else. The task is to identify a celebrity by only asking questions to people of the form: "Do you know him/her?" Design an efficient algorithm to identify a celebrity or determine that the group has no such person. How many questions does your algorithm need in the worst case? Hint: Solve first a simpler version in which a celebrity must be present.

[SOLUTION]

The problem can be solved by a recursive algorithm based on the decrease-by-one strategy. Indeed, by asking just one question, we can eliminate the number of people who can be a celebrity by 1, solve the problem for the remaining group of $n-1$ people recursively, and then verify the returned solution by asking no more than two questions. Here is a more detailed description of this algorithm:

If $n = 1$, return that one person as a celebrity. If $n > 1$, proceed as follows:

Step 1 Select two people from the group given, say, A and B, and ask A whether A knows B. If A knows B, remove A from the remaining people who can be a celebrity; if A doesn't know B, remove B from this group.

Step 2 Solve the problem recursively for the remaining group of $n-1$ people who can be a celebrity.

Step 3 If the solution returned in Step 2 indicates that there is no celebrity among the group of $n-1$ people, the larger group of n people cannot contain a celebrity either. If Step 2 identified as a celebrity a person other than either A or B, say, C, ask whether C knows the person removed in Step 1 and, if the answer is no, whether the person removed in Step 1 knows C. If the answer to the second question is yes," return C as a celebrity and "no celebrity" otherwise. If Step 2 identified B as a celebrity, just ask whether B knows A: return B as a celebrity if the answer is no and "no celebrity" otherwise. If Step 2 identified A as a celebrity, ask whether B knows A: return A as a celebrity if the answer is yes and "no celebrity" otherwise.

The recurrence for $Q(n)$, the number of questions needed in the worst case, is as follows:

$$Q(n) = Q(n-1) + 3 \quad \text{for } n > 2, \quad Q(2) = 2, \quad Q(1) = 0.$$

Its solution is $Q(n) = 2 + 3(n-2)$ for $n > 1$ and $Q(1) = 0$.

8. *Flipping pancakes*: There are n pancakes all of different sizes that are stacked on top of each other. You are allowed to slip a flipper under one of the pancakes and flip over the whole sack above the flipper. The purpose is to arrange pancakes according to their size with the biggest at the bottom.

[SOLUTION]

Here is a decrease-and-conquer algorithm for this problem. Repeat the following until the problem is solved: Find the largest pancake that is out of order. (If there is none, the problem is solved.) If it is not on the top of the stack, slide the flipper under it and flip to put the largest pancake on the top. Slide the flipper under the first-from-the-bottom pancake that is not in its proper place and flip to increase the number of pancakes in their proper place at least by one.

The number of flips needed by this algorithm in the worst case is $W(n) = 2n - 3$, where $n \geq 2$ is the number of pancakes. Here is a proof of this assertion by mathematical induction. For $n = 2$, the assertion is correct: the algorithm makes one flip for a two-pancake stack with a larger pancake on the top, and it makes no flips for a two-pancake stack with a larger pancake at the bottom. Assume now that the worst-case number of flips for some value of $n \geq 2$ is given by the formula $W(n) = 2n - 3$. Consider an arbitrary stack of $n+1$ pancakes. With two flips or less, the algorithm puts the largest pancake at the bottom of the stack, where it doesn't participate in any further flips. Hence, the total number of flips needed for any stack of $n + 1$ pancakes is bounded above by

$$2 + W(n) = 2 + (2n - 3) = 2(n + 1) - 3.$$

In fact, this upper bound is attained on the stack of $n + 1$ pancakes constructed as follows: flip a worst-case stack of n pancakes upside down and insert a pancake larger than all the others between the top and the next-to-the-top pancakes. (On the new stack, the algorithm will make two flips to reduce the problem to flipping the worst-case stack of n pancakes.) This completes the proof of the fact that

$$W(n + 1) = 2(n + 1) - 3,$$

which, in turn, completes our mathematical induction proof.

9. Consider the problem of finding, for a given positive integer n, the pair of integers whose sum is n and whose product is as large as possible. Design an efficient algorithm for this problem and indicate its efficiency class.

[SOLUTION]

Let x be one of the numbers in question; hence, the other number is $n-x$. The problem can be posed as the problem of maximizing $f(x) = x(n-x)$ on the set of all integer values of x. Since the graph of $f(x) = x(n-x)$ is a parabola with the apex at $x = n/2$, the solution is $n/2$ if n is even and $\lfloor n/2 \rfloor$ (or $\lceil n/2 \rceil$) if n is odd. Hence, the numbers in question can be computed as $\lfloor n/2 \rfloor$ and $n - \lfloor n/2 \rfloor$, which works both for even and odd values of n. Assuming that one division by 2 takes a constant time irrespective of n's size, the algorithm's time efficiency is clearly in $\Theta(1)$.

10. During World War II, mathematician Alan Turing tries to crack the enigma code with help from fellow mathematicians. Watch The Imitation Game movie and write an essay.