CS 131 – Spring 2017 – Lab 8

Question 1 This exercise deals with the problem of finding the largest sum of consecutive terms of a sequence of n real numbers. When all terms are positive, the sum of all terms provides the answer, but the situation is more complicated when some terms are negative. For example, the maximum sum of consecutive terms of the sequence $-2, 3, -1, 6, -7, 4$ is 3 + (-1) + 6 = 8. We first look at the brute-force algorithm for solving this problem; then we develop a divide-and-conquer algorithm for solving it.

a) Use pseudocode to describe an algorithm that solves this problem by finding the sums of consecutive terms starting with the first term, the sums of consecutive terms starting with the second term, and so on, keeping track of the maximum sum found so far as the algorithm proceeds.

b) Determine the computational complexity of the algorithm in part (a) in terms of the number of sums computed and the number of comparisons made.

c) Devise a divide-and-conquer algorithm to solve this problem. [Hint:Assume that there are an even number of terms in the sequence and split the sequence into two halves. Explain how to handle the case when the maximum sum of consecutive terms includes terms in both halves.]

d) Use the algorithm from part (c) to find the maximum sum of consecutive terms of each of the sequences: -2, 4, -1, 3, 5, -6, 1, 2; 4, 1, -3, 7, -1, -5, 3, -2; and -1, 6, 3, -4, -5, 8, -1, 7.

e) Find a recurrence relation for the number of sums and comparisons used by the divide-and-conquer algorithm from part (c).

f) Use the master theorem to estimate the computational complexity of the divide-and-conquer algorithm. How does it compare in terms of computational complexity with the algorithm from part (a)?

Solution

a)
```
procedure largest sum(a1,...,an)
     best := 0 {empty subsequence has sum 0}
     for i := 1 to n
         sum := 0
         for j := i + 1 to n
              sum := sum + aj
              if sum > best then best := sum
             {best is the maximum possible sum of numbers in the list}
```

b) $O(n^2)$

c) We divide the list into a first half and a second half and apply the algorithm recursively to find the largest sum of consecutive terms for each half. The largest sum of consecutive terms in the entire sequence is either one of these two numbers or the sum of a sequence of consecutive terms that crosses the middle of the list. To find the largest possible sum of a sequence of consecutive terms that crosses the middle of the list, we start at the middle and move forward to find the largest possible sum in the second half of the list, and move backward to find the largest possible sum in the first half of the list; the desired sum is the sum of these two quantities. The final answer is then the largest of this sum and the two answers obtained recursively. The base case is that the largest sum of a sequence of one term is the larger of that number and 0.

d) 11, 9, 14

e) $S(n) = 2S(n/2) + n$, $C(n) = 2C(n/2) + n + 2$, $S(1) = 0$, $C(1) = 1$

f) $O(n \log n)$, *better than* $O(n^2)$.

Question 2 Determine whether each of these functions is $O(x)$.

a) $f(x) = 10$

b) $f(x) = 3x + 7$

c) $f(x) = x^2 + x + 1$

d) $f(x) = 5 \log x$

e) $f(x) = \lfloor x \rfloor$

f) $f(x) = \lceil x/2 \rceil$

Solution

a) $C = 1$, $k = 10$

b) $C = 4$, $k = 7$

c) No

d) $C = 5$, $k = 1$

e) $C = 1$, $k = 0$

f) $C = 1$, $k = 2$

Question 3 Explain what it means for a function to be O(1).

Solution

All functions for which there exist real numbers k *and* C *with* $|f(x)| \leq C$ *for* $x > k$. *These are the functions* $f(x)$ *that are bounded for all sufficiently large* x.

Question 4 Determine whether each of the functions 2^{n+1} and 2^{2n} is $O(2^n)$.

Solution

2^{n+1} *is* $O(2^n)$ *but* 2^{2n} *is not.*

Question 5 Give a big-O estimate for each of these functions. For the function g in your estimate that $f(x)$ is $O(g(x))$, use a simple function g of the smallest order.

a) $n \log(n^2 + 1) + n^2 \log n$

b) $(n \log n + 1)^2 + (\log n + 1)(n^2 + 1)$

c) $n^{2^n} + n^{n^2}$

Solution

a) $O(n^2 \log n)$

b) $O(n^2 (\log n)^2)$

c) $O(n^{2^n})$