Problems 1.

Selection Problem

Consider the following problem:
Given an unsorted list of $n$ distinct numbers $x_1, x_2, \cdots x_n$, and an integer $B$, find a maximum cardinality subset of the list which sums to at most $B$.

For example, given the list \{35, 27, 22, 55, 87, 74\} and $B=100$, a maximum cardinality set has three elements: \{35, 22, 27\}; since $35 + 22 + 27 \leq 100$ and any four elements of the original list sums to more than 100.

(a) [5pts] Give a simple $O(n \log n)$ algorithm to solve this problem.

(b) [15pts] Describe (and provide pseudocode for) how to use the Selection algorithm (discussed in class) to solve the problem in expected $O(n)$ time.

Justify your time bound. Note you may use the Selection algorithm as a black box routine which given any list, and a value $k$ returns the $k$th smallest element in expected linear time. (Hint: start by finding the median of the list, and testing if all the numbers smaller than the median sum to $B$ or less).
Problem 2.

Divide-and-Conquer

Given an array \(A = \langle A[1], A[2], \ldots, A[n] \rangle\), we want to find the minimum and maximum values in \(A\) and we do this by comparing elements of \(A\). Assume that \(n = 2^d\) for some \(d \geq 1\).

(a) [5 pts] The “obvious” algorithm makes \(2n - 2\) comparisons. Give pseudocode for the algorithm and explain the number of comparisons made.

(b) [7 pts] Can we do it better? Carefully specify (give pseudocode) a more efficient divide-and-conquer algorithm (for full credit, it should have the \(T(n)\) stated in part (d)).

(c) [5 pts] Let \(T(n)\) be the number of comparisons your algorithm makes. Write a recurrence relation for \(T(n)\).

(d) [7 pts] Show that your recurrence relation has as its solution \(T(n) = 3n/2 - 2\).

Problem 3.

Recurrences

For our first example of the substitution method, we solved the recurrence:

\[
T(n) = 2T(n/2) \text{ for } n > 1 \\
T(n) = d \text{ for } n = 1
\]

We correctly guessed \(f(n) = n\) and showed \(T(n) = O(n)\).

(a) [10 points] Show the steps of the substitution method with an incorrect guess of \(T(n) = O(n^2)\). Does it work (i.e. does the proof follow through)? If so, why? If not, explain why not.

(b) [10 points] Now show the steps of the substitution method with an incorrect guess of \(T(n) = O(1)\). Does it work? If so, why? If not, explain why not.

Problem 4.

Recurrences

Here is a technique for computing \(2^n\) for \(n \geq 0\):

If \(n = 0\) then \(2^n = 1\), if \(n = 1\) then \(2^n = 2\), if \(n > 1\) and \(n\) is even then \(2^n = 2^{n/2} \cdot 2^{n/2}\), and if \(n > 1\) and \(n\) is odd then \(2^n = 2 \cdot 2^{(n-1)/2} \cdot 2^{(n-1)/2}\).

(a) [10 points] Use this technique to write a simple (i.e. don’t worry about runtime) recursive function that takes as input a value \(n\) and returns \(2^n\).

(b) [7 points] Write a recurrence for your recursive function from (a). For simplicity, your recurrence should have just two cases: the base case is when \(n \leq 1\) and the recursive case is when \(n > 1\) (so do not distinguish between even and odd \(n\)). Also for simplicity, indicate an addition of constant time with +1.

(c) [5 points] Solve your recurrence. Hint: notice that this part of the problem is worth only 5 points.

(d) [5 points] Draw a tree that illustrates the sequence of calls to the function for \(n=8\). You may draw this by hand.

(e) [10 points] Modify your function from (a) so that it uses memoization.

(f) [7 points] What is the runtime of your function from (e)? Briefly explain.