CSE 101 Homework 6

Brian Masse, Kreshiv Chawla, Taira Sakamoto, Emily Xie, Annabelle Coles

November 25, 2025

1. **Recurrence Relations**

   (a) $T(n) = 4T(\lfloor n/4 \rfloor) + cn^3, n \geq 2, T(1) = c$

   (b) $T(n) = 2 * T(n-3)$ for $n \geq 4$, $T(1) = c$

   (c) $T(n) = T(\lfloor n/4 \rfloor) + c, T(1) = c$

   (d) $T(n) = (\log n)T(\lfloor n/2 \rfloor), n \geq 2; T(1) = c.$

2. **High Frequency Element**

Say we have an array of numbers $A[1...n]$. x is *high frequency* in A if it occurs more than $n/2$ times in A, i.e., there are strictly more than $n/2$ many indices $1 \leq i \leq n$ with $A[i] = x$. Here's an algorithm that finds a high frequency element if one exists (it may return an element if no high frequency element exists):

HF(A[1..n])

(a) If $n = 0$ return "No HF element".

(b) If $n = 1$ return $A[1]$.

(c) If n is odd do:

(d) $\quad Count = 0$

(e) $\quad$ FOR $I = 1$ to n IF $A[I] = A[n]$ THEN $Count++$

(f) $\quad$ IF $Count > n/2$ THEN Return $A[n]$

(g) Initialize an array $B[1..\lfloor n/2 \rfloor]$.

(h) $I = 1, J = 1$.

(i) While $I < n$ do:

(j) $\quad$ IF $A[I] == A[I+1]$ THEN $B[J] = A[I], J++$

(k) $\quad I = I + 2$.

(l) Return $HF(B[1..J-1])$.

Questions:

(a) Give the recursive calls the algorithm would make on input $A[1..7] = (1, 4, 3, 3, 2, 3, 3)$.

(b) Prove that if x is a high-frequency element in $A[1..n]$, then x is a high frequency element in $B[1...J-1]$ at the end of the loop. (Hint: look at the number of times each of the following occur for consecutive eelements of A: Both are x, both are some $y \neq x$, exactly one is x, neither is x and are not equal.)

(c) Use this to prove that if x is a high-frequency element in $A[1..n]$, then $HF(A[1..n]) = x$.

(d) Give a worst-case time analysis for this algorithm.

3. **Weighted Median**

Say that we are given a list of pairs of values and weights, with weights greater than 0, $(v_1, w_1), ..(v_n, w_n)$. Let $W = \sum_{1 \leq i \leq n} w_i$ be the total weight. The weighted median is a value v_j with $\sum_{1 \leq i \leq n, v_i < v_j} w_i \leq W/2$ and $\sum_{1 \leq i \leq n, v_i > v_j} w_i \leq W/2$.

For example, if the list had elements $(2, .1), (4, .2), (-3, .2), (1, .4), (5, .1)$,we can compute $W = 1$, so $W/2 = .5$. A weighted median is 1, because the sum of the weights of values less than 1 is .2 and the sum of the weights larger than 1 is $.1 + .2 + .1 = .4$.

Give an efficient algorithm (faster than sorting the list) to compute a weighted median. Hint: use the select procedure from class as a subroutine, but don't try to modify it. Must be $o(nlogn)$.)

4. **Weighted Independent Set for Trees**

A subset S of vertices in an undirected graph is an *independent set* if it doesn't contain both ends of any edge. If we assign every vertex a weight $w(x) > 0$, the *maximum weight independent set* is to find the independent set of the graph that has maximum possible total weight of its vertices. While the maximum weight independent set problem is NP-hard, some special cases can be solved efficiently. In particular, consider the special case when the underlying graph is a complete binary tree of depth k, so $|V| = 2^{k+1} - 1$. Give a divide-and-conquer algorithm for this problem that runs in polynomial time. (Hint: have your algorithm return not just the weight of the maximum weighted independent set, but both of the values for the two cases, the root is in S and the root is not in S. This problem can be solved using dynamic programming, but it is not required and the dynamic programming solution will not be given any credit for this assignment.) (5 points clear algorithm description, 5 points for short correctness argument, 5 points for correct time analysis, and 5 points for efficiency; the best algorithm is $O(|V|)$ time.)

5. **Karatsuba implementation**

This problem is designed to teach you an idea that makes divide-and-conquer algorithms that only improve time for huge inputs into ones that give substantial improvements even for moderate sized inputs.

Implement both the Karatsuba method for multiplying polynomials from the ungraded problems above and a straight-forward $O(n^2)$ polynomial multiplication algorithm. Collect average running times for a wide range of input sizes n, say powers of 2 until the algorithm is taking over 15 minutes to run, for random polynomials with co-efficients 0 or 1. Graph these on a log-log scale, log n vs. log (algorithm time). How big are inputs where the Karatsuba method is faster, or how big would you interpolate such inputs to be from your data (if there are no actual data points where Karatsuba is better)? Then try a hybrid algorithm, where we use the Karatusba recursion when $n > T$ and the quadratic time method in recursive calls when $n \leq T$. For a wide range of possible T's, starting with rather small values such as $T = 16$, graph the hybrid's performance and compare to the other two. What do you conclude from this experiment?

(2 points clearly describing features of implementation and experiment, such as PL, libraries, type of computer, etc. For each of Karatsuba, quadratic algorithm and hybrid, 2 points for running experiment with adequate variety of input sizes and 2 points for clearly presenting results. 6 points for conclusions based on comparing results.)