[redacted] [redacted]

Solutions to Problem 1 of Homework 3 (12 points)

[redacted] [redacted]

[redacted]

An array $A[0 \ldots (n-1)]$ is called *rotation-sorted* if there exists some some cyclic shift $0 \le c < n$ such that $A[i] = B[(i + c \bmod n)]$ for all $0 \le i < n$, where $B[0 \ldots (n-1)]$ is the sorted version of A.[1] For example, $A = (2, 3, 4, 7, 1)$ is rotation-sorted, since the sorted array $B = (1, 2, 3, 4, 7)$ is the cyclic shift of A with $c = 1$ (e.g. $1 = A[4] = B[(4+1) \bmod 5] = B[0] = 1$). For simplicity, below let us assume that n is a power of two (so that can ignore floors and ceilings), and that all elements of A are distinct.

(a) (4 points) Prove that if A is rotation-sorted, then one of $A[0 \ldots (n/2-1)]$ and $A[n/2 \ldots (n-1)]$ is fully sorted (and, hence, also rotation-sorted with $c = 0$), while the other is at least rotation-sorted. What determines which one of the two halves is sorted? Under what condition *both halves* of A are sorted?

Solution: INSERT YOUR SOLUTION HERE ☐

(b) (8 points) Assume again that A is rotation-sorted, but you are not given the cyclic shift c. Design a divide-and-conquer algorithm to compute the minimum of A (i.e., $B[0]$). Carefully prove the correctness of your algorithm, write the recurrence equation for its running time, and solve it. Is it better than the trivial $O(n)$ algorithm? (**Hint**: Be careful with $c = 0$ an $c = n/2$; you might need to handle them separately.)

Solution: INSERT YOUR SOLUTION HERE ☐

[1] Intuitively, A is either completely sorted (if $c = 0$), or (if $c > 0$) A starts in sorted order, but then "falls off the cliff" when going from $A[n-c-1] = B[n-1] = \max$ to $A[n-c] = B[0] = \min$, and then again goes in increasing order while never reaching $A[0]$.

[redacted] [redacted]

Solutions to Problem 2 of Homework 3 (13+3 points)

[redacted] [redacted]

[redacted]

In the lecture, we look at splitting an n-bit integer into two equal parts and recursing on these smaller parts before combining the solutions. In this problem, we will look at generalizing the question by dividing into m equal parts, where we assume that $n \mod m = 0$. Let the m parts be $X_0, \ldots, X_{m-1}$ such that

$$X = \sum_{i=0}^{m-1} X_i \cdot 2^{\frac{i \cdot n}{m}}$$

Similarly, one can rewrite the other integer Y as:

$$Y = \sum_{i=0}^{m-1} Y_i \cdot 2^{\frac{i \cdot n}{m}}$$

Let $Z = XY$. Then, one can break up Z into $Z_0, \ldots, Z_{2m-2}$ where

$$Z_i = \sum_{j=0}^{i} X_j \cdot Y_{i-j}$$

for $i = 0, 1, \ldots, 2m-2$. We assume that $X_k = 0$ and $Y_k = 0$ for $k \geq m$.

(a) (4 points) Let $f(m, n)$ represent the non-recursive cost of combining the values, i.e., to compute the product $Z = XY$ given $Z_0, \ldots, Z_{2m-2}$. Specifically, $f(m, n)$ is the number of bit operations needed to compute Z given $Z_0, \ldots, Z_{2m-2}$. In other words, this is the process of combining the answers from the recursive calls, i.e., the non-recursive cost. Derive a Θ-order for $f(m, n)$. (**Hint**: Begin by expressing Z in terms of the individual Z_i values. Then estimate the cost of addition and multiplication, in terms of bits. Substituting $m = 2$ should give you the values as defined in Karatsuba's algorithm from the lecture.)

Solution: INSERT YOUR SOLUTION HERE □

(b) (4 points) There is literature available that shows that one can compute $Z_0, \ldots, Z_{2m-2}$ through $O(m \log m)$ multiplications (i.e. a maximum of $cm \log m$ for some $c > 0$) and $O(m \log m)$ additions over $k = n/m$-bit integers, from $X_0, \ldots, X_{m-1}$, $Y_0, \ldots, Y_{m-1}$. For example, when $m = 2$, it takes 3 multiplications, as observed in the standard Karatsuba algorithm discussed in the lecture.

Now, let $T_m(n)$ represent the running time of the algorithm to compute the product of two n-bit integers by splitting into m parts. Derive a recurrence relation $T_m(n) = a_m \cdot T_m(n/b) + f_m(n)$. For example, when $m = 2$ we know that $T_2(n) = 3T_2(n/2) + O(n)$. (**Hint**: You will use your results from part (a).)

[redacted]

Solution: INSERT YOUR SOLUTION HERE $\square$

(c) (3 points) (**Extra Credit**) For every $\epsilon > 0$, argue there exists a constant m (depending on of ϵ, but independent of n) such that $T(n)$ in part (b) satisfies $T(n) = \Theta(n^{1+\epsilon})$. (**Hint**: Express the solution in the form $n^{1+\epsilon(m)}$ for some function $\epsilon(m)$, and then argue that $\epsilon(m) \to 0$ as $m \to \infty$.)

Solution: INSERT YOUR SOLUTION HERE $\square$

(d) (5 points) Let X be an n-bit integer and Y be an m-bit integer. Further, let n be a multiple of m, i.e., $n = mk$ for some positive integer k. Present an algorithm that can multiply X, Y in time faster than directly applying Karatsuba. Briefly argue the correctness of your algorithm and state the running time of your algorithm.

Solution: INSERT YOUR SOLUTION HERE $\square$

[redacted]

[redacted]

Solutions to Problem 3 of Homework 3 (20 (+8) points)

[redacted] [redacted]

[redacted]

Assume $A[1 \ldots n]$ is an array of numbers, where each $A[i] \in \{0, 1, 2\}$. A *span* of A is any interval $[start, end]$ such that $\{0, 1, 2\} \subseteq \{A[start], \ldots, A[end]\}$. For example, if $A = (0, 1, 1, 0, 2, 1, 0)$, then $[1, 5]$ and $[4, 6]$ are spans of A, while $[1, 4]$ is not (since $2 \notin \{0, 1, 1, 0\}$). The *cardinality* of the span $[start, end]$ is defined to be $(end - start + 1)$. Finally, a span $[start, end]$ is *minimum* if it has minimum cardinality among all other spans. For the example above, $[4, 6]$ is a minimum span (of cardinality 3), while $[1, 5]$ is not a minimum span. Finally, $[\infty, \infty]$ will correspond to the case when no valid span exists for that array.

(a) (3 points) Let us assume that $(start, end)$ is a minimum span. Then, show that the subarray $A[start, \ldots, end]$ is of the form $ab^{end-start-1}c$ where (a, b, c) is a permutation of $(0, 1, 2)$. In other words, a minimal span has a given structure where the end points are distinct integers, and internally it simply repeats the third remaining integer.

Solution: INSERT YOUR SOLUTION HERE □

(b) (2 points) Prove that if $[start, end]$ is a minimum span, then either $end \leq n/2$ or $start > n/2$, or $start \leq n/2 < end$.

Solution: INSERT YOUR SOLUTION HERE □

(c) (6 points) In part (a), you showed that a minimum span $[start, end]$ has a certain structure. In this part, you will return $[start, end]$ if there exists $A[start], \ldots, A[end]$ satisfying the structure from part (a) and if $start \leq n/2 < end$. If not, you will return (∞, ∞).

For example, if $A = [2, 1, 0, 0, 2, 2]$, the minimum span of A is $[1, 3]$. However, PartC will return [2,5] as it satisfies the structure from before, and also satisfies $start \leq n/2 < end$. Present an iterative pseudocode for this algorithm - PartC($A[\ell, \ldots, m]$) that follows the brief of the question. Briefly justify the running time of your algorithm and argue its correctness.

Solution: INSERT YOUR SOLUTION HERE □

(d) (6 points) Use the previous parts as black-box (including part (c) above), to present a pseudocode for the recursive algorithm, MinimumSpan($A[\ell, \ldots, m]$), based on the divide-and-conquer paradigm to find the minimum span of the array A. Prove the correctness of your algorithm.

Solution: INSERT YOUR SOLUTION HERE □

(e) (3 points) Formulate a recurrence relation for your above algorithm, with appropriate base cases. Solve for it using any method of your choice.

Solution: INSERT YOUR SOLUTION HERE □

(f) (8 points) (**Extra Credit**) Improve your solution to part (c)-(e) to get $O(n)$ time recursive algorithm. You may assume that n is a power of 2 for simplicity.

Do not forget to argue the correctness of your algorithm, derive a recurrence relation (as always with appropriate base cases), and finally solve it to justify the running time of your algorithm. (**Hint**: Try to compute more than the minimum span in your subproblems. Something which will make part (c) run in time $O(1)$.)

Solution: INSERT YOUR SOLUTION HERE □

[redacted]