CS 344: Homework #2

Due on February 10, 2017 at 11:59pm

Professor Bahman Kalantari Section #1

Douglas Rudolph

1) Multiplication

Divide and Conquer

Consider the binary numbers $x = 10011011$ and $y = 10111010$. Decomposing x using $a = 1001$, $b = 1011$ and y using $c = 1011$, $d = 1010$, compute the product $x * y$ using the ordinary way and the way done in divide-and-conquer method by forming $w1 = a + b$, $w2 = c + d$, $u = w1 * w2$, $v = ac$, $w = bd$. Recall $xy = 2^n(v) + (2^{\frac{n}{2}} * (u - v - w)) + w$. Each multiplication by 2 amounts to a shift. Count the binary operations in each method.

We define binary operations as addition of 2 bits and multiplication by 2.

Divide & Conquer Approach: Applying Karatsuba's Algorithm
To start, we can rewrite the terms x and y to be written as an expanded sum of each of its original terms.

$$\begin{array}{lcl} x = & 10011011 & = & 1001 * 2^4 + 1011 \\ y = & 10111010 & = & 1011 * 2^4 + 1010 \end{array}$$

Now, let's formally define a, b, c, d as stated in the problem.

$$\begin{array}{ll} a = 1001 & b = 1011 \\ c = 1011 & d = 1010 \end{array}$$

Now we define the desired quantities:

$w_1 =$	$a + b =$	10100
$w_2 =$	$c + d =$	10101
$w_1 * w_2 =$	$(a + b)(c + d) =$	$multiply(a, c) + multiply(b, c) + multiply(a, d) + multiply(b, d) = 1101001000$
	$b * c =$	1111001
	$a * d =$	1011010
$v =$	$a * c =$	$1001 * 1011 = 1100011$
$w =$	$b * d =$	$1011 * 1010 = 1101110$

Now to putting the problem in terms of the original statement.

$$\begin{array}{rcl} x * y & = & 2^n(v) + (2^{\frac{n}{2}} * (u - v - w)) + w \\ & = & 2^8(1100011) + 2^4 * (110100100 - 1100011 - 1101110) + 1101110110001100000000 \\ & & +2^4 * (11010011) + 1101110 + 110001100000000 + 110100110000 + 1101110 \\ \\ & = & 111000010011110. \end{array}$$

The above requires **5 + 5 + 7 + 8 + 9 + 8 + 8 + 7 + 12 = 79 bit wise additions** and **12 shifts**, giving a **total of 91 operations** for this step.

To obtain 110100100, we recursively have to apply multiply to obtain the **sum: multiply(a,c) + multiply(b,d) + multiply(b,c) + multiply (a,d)**:

multiply(a,c) = 1100011:

Let $x = 1001$, $y = 1011$, $a = 10$, $b = 01$, $c = 10$, and $d = 11$

$$\begin{array}{lllll} w_1 & = & a + b = 11 & & \\ w_2 & = & c + d = 100 & & \\ w_1 * w_2 & = & 11 * 100 & = & 1100 \text{ (2 bit shifts)} \\ v & = & ac & = & 100 \text{ (1 bit shift)} \\ w & = & bd & = & 11 \end{array}$$

We now substitute the appropriate values:

$$\begin{array}{lll} x * y & = & 2^(v) + (2^{\frac{n}{2}} * (u - v - w)) + w: \\ & = & 2^4(100) + (2^2 * (1100 - 100 - 11)) + 11 \\ & = & 1100011 \end{array}$$

In total, the above requires **2 + 2 + 3 + 3 + 3 + 2 = 17 additions** and **9 bit shifts**, giving a **total of 26 operations**.

multiply(b,d) = 1101110:

Let x = 1011, y = 1010, a = 10, b = 11, c = 10, d 100

$$\begin{array}{lllll} w_1 * w_2 & = & 11 * 100 & = & 10 \\ w_1 & = & a + b & = & 101 \\ w_2 & = & c + d & = & 100 \\ w_1 * w_2 & = & 101 * 100 & = & 10100 \text{ (2 bit shifts)} \\ v & = & ac & = & 100 \text{ (bit shift by 2)} \\ w & = & bd & = & 110 \text{ (bit shift by 2)} \end{array}$$

We now substitute the appropriate values into

$$\begin{array}{lll} x * y & = & 2^(v) + (2^{\frac{n}{2}} * (u - v - w)) + w: \\ & = & 2^4(100) + (2^2 * (10100 - 100 - 110)) + 110 \\ & = & 1101110 \end{array}$$

In total, this requires **2 + 2 + 3 + 3 + 3 + 4 = 17 additions** and **2 + 2 + 2 + 6 = 12 bit shifts**, giving a **total of 29 operations**.

multiply(b,c) = 1111001:

Let x = 1011, y = 1011, a = 10, b = 11, c = 10, d = 11

w_1	=	$a + b$	=	101
w_2	=	$c + d$	=	101
w_1 * w_2	=	$(a + d)(b + c)$	=	110 (1 bit shift) +100 (1 bit shift) +110 (1 bit shift) +1001 (base case); 4 operations split across 2 additions and 2 shifts) = 11001
v	=	ac	=	100 (1 bit shift)
w	=	bd	=	110 (1 bit shift)

We now substitute the appropriate values into

$$\begin{aligned} x * y &= 2^(v) + (2^{\frac{n}{2}} * (u - v - w)) + w: \\ &= 2^4(100) + (2^2 * (11001 - 100 - 110)) + 110 \\ &= 1111001 \end{aligned}$$

In total, this **requires 2 + 2 + 3 + 3 + 3 + 4 + 2 = 19 additions** and **1 + 1 + 1 + 2 + 1 + 1 + 4 + 2 = 13 shifts**, giving a **total of 32 operations**.

multiply(a,d) = 1011010:

Let x = 1001, y = 1010, a = 10, b = 01, c = 10, d = 10

w_1	=	$a + b$	=	11
w_2	=	$c + d$	=	100
w_1 * w_2	=	$11 * 100$	=	1100 (2 bit shifts)
v	=	$a * c$	=	100 (1 bit shift)
w	=	$b * d$	=	10 (base case, mult by 1)

We now substitute the appropriate values into

$$\begin{aligned} x * y &= 2^(v) + (2^{\frac{n}{2}} * (u - v - w)) + w: \\ &= 2^4(100) + (2^2 * (1100 - 100 - 10)) + 10 \\ &= 1011010 \end{aligned}$$

In total, this requires, **2 + 2 + 3 + 2 + 2 + 4 = 15 additions** and **2 + 1 + 4 + 2 = 9 bit shifts**, giving a **total of 26 operations**.

In total, the above **required 147 total 2-bit additions and 55 total bit shifts**, giving us a **grand total of 202 operations**.

Using the conventional way:

							1	0	0	1	1	0	1	1
						+	1	0	1	1	1	0	1	0
							0	0	0	0	0	0	0	0
						1	0	0	1	1	0	1	1	0
					0	0	0	0	0	0	0	0	0	0
				1	0	0	1	1	0	1	1	0	0	0
			1	0	0	1	1	0	1	1	0	0	0	0
		1	0	0	1	1	0	1	1	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	1	0	0	1	1	1	1	0

In total, this **requires 28 shifts and 49 additions** giving us a **total of 77 operations**.

Problem 2)

You are given an infinite array A[i] in which the first n elements are integers in sorted order and the rest are filled with ∞, You are not given n. Describe an algorithm that takes as input an integer x and finds a position in the array containing x, if such a position exists, in O(log n) time.

Algorithm:
1) Let i = 0 and let j = 1.
2) While true, do *#3*: While A[i] is not equal to ∞
3) Set i = j and j = i * 2. If A[j] returns ∞, then we break to *#4*. If A[j] is greater than x, then we break to *#5*. Otherwise, go to *#2*.
4) Run a binary search on A[i, j] to find the last element that is not equal to ∞, that is, A[n-1]. Set j = n - 1 when it is found and go to *#5*.
5) Run a binary search on A[i, j] searching for x. If x is found, return the index, otherwise return -∞ to indicate not found.

This algorithm runs in *logn* time: Searching for the first ∞ requires that we iterate through the loop by doubling the index each time, which implies that n = 2^k. This implies that k = $log(n)$, which means that this step takes $log(n)$ time.
The search for the first non ∞ element (i.e. the index n-1) takes $log(n)$ time as this is done via binary serach.
Finally, searching for x in the subarray A[$\frac{n-1}{2}$, n-1] takes $log(\frac{n-1}{2})$ time as we are performing binary search on an array of maximum length $\frac{n-1}{2}$.
In total, the algorithm's runtime can be described by $2log(n) + log(\frac{n-1}{2})$, which is $O(log(n))$ as desired.

Problem 3)

Solve each of the recurrence relations and give bound for each. You can use the master theorem if applicable.

(a) T(n) = 5T($\frac{n}{4}$) + n

Using the Master Theorem: Here, a = 5, b = 4, and d = 1. This means that $log_b a$ equals $log_4 5 \approx 1.16$, which is greater than d. This implies that T(n) = $O(n^{log_b a})$ by the Master Theorem, which means that T(n) = $\Theta(n^{1.16})$.

(b) T(n) = 7T($\frac{n}{7}$) + n

Using the Master Theorem: Here, a = 7, b = 7, and d = 1. This means that $log_b a$ equals $log_7 7 = 1$, which equals d. This implies that T(n) = $O(n^d logn)$ by the master theorem, which means that T(n) = $\Theta(nlogn)$.

(c) T(n) = 9T($\frac{n}{4}$) + n^2

Using the Master Theorem: Here, a = 9, b = 4 and d = 2. This means that $log_b a$ equals $log_4 9 \approx 1.58$, which is less than d. This implies that T(n) = $O(n^d)$ by the master theorem, which means that T(n) = $\Theta(n^{1.58})$.

(d) T(n) = 8T($\frac{n}{2}$) + n^3

Using the Master Theorem: Here, a = 8, b = 2, and d = 3. This means that $log_b a$ equals $log_2 8 = 3$, which equals d. This implies that T(n) = $O(n^d logn)$ by the master theorem, which means that T(n) = $\Theta(n^3 logn)$.

(e) T(n) = 49T($\frac{n}{25}$) + $n^{3.5} logn$

We *cannot* use the master theorem here because we have a non-polynomial term $n^{3.5} logn$) in the recurrence relation. As such, we can solve this recurrence relation using iteration:

$$\begin{aligned} T(n) &= 49T(\tfrac{n}{25}) + n^{3.5}logn \\ &= 49(49T(\tfrac{n}{25^2}) + n^{3.5}logn) + n^{3.5}logn \\ &= 49^2T(\tfrac{n}{25^2}) + 49n^{3.5}logn + n^{3.5}logn \\ &= 49^kT(\tfrac{n}{25^k}) + (1{+}49{+}\ldots{+}49^k)n^{3.5}logn \end{aligned}$$

Let b = 49^kT($\frac{n}{25^k}$) and then let c = $(1 + 49 + \ldots + 49^k) = \sum_{i=0}^{k} 49^i$. We can then rewrite the above as:

$$49^k\text{T}(\tfrac{n}{25^k}) + (1 + 49 + \ldots + 49^k)\ n^{3.5}logn = \text{b} + \text{c}{*}n^{3.5}logn$$

Clearly, this recurrence relation is $O(n^{3.5}logn)$, which gives us $\Theta(n^{3.5}logn)$

(f) T(n) = T($n^{.5}$) + 1.

We *cannot* use the master theorem here because we have a non-polynomial term ($n^{.5}$) in the recurrence relation. As such, we can solve this recurrence relation using substitution:

Let n = 2^k. We can rewrite the above as :

$$\text{T(n)} = \text{T}(2^k) = \text{T}((2^{\frac{k}{2}}) + 1$$

Let S(k) = T(2^k). We can then write S(k) as:

$$\text{S(k)} = \text{S}(\tfrac{k}{2}) + 1$$

The above function is in a form that can be solved by the master theorem as S is a function of a linear term and a constant. As such, let a = 1, b = 2, and d = 0. It follows that:

$log_b a$ equals $log_2 1 = 0 =$ d, which implies that S(k) = $O(k^d \log(k)) = O(k^0 \log(k)) = O(\log(k))$

Recall that n = 2^k. We can solve for k as follows:

$$n = 2^k$$
$$log(n) = \log(2^k)$$
$$log(n) = k$$

We can plug this formula into $O(log(k)$ to get the solution to the recurrence relation:

$$O(log(k)) = O(log(log(n))) \rightarrow \text{T(n)} = \Theta(log(logn))$$

As such, we conclude that our theta bound is T(n) = $\Theta(log(logn))$.

Problem 4)

Find the coefficients of the polynomial p(x) = $a_2x^2+a_1x+a_0$ such that p(1) = 2, p(2) = 1, p(3) = 0

To find these coefficients, we can set up a series of linear equations:

$$\text{p(1): } a_2x^2+a_1x+a_0 = 2$$
$$\text{p(2): } a_2x^2+a_1x+a_0 = 1$$
$$\text{p(3): } a_2x^2+a_1x+a_0 = 0$$

Plugging in for x, we get:

$$\text{p(1): } 1a_2 + 1a_1 + a_0 = 2$$
$$\text{p(2): } 4a_2 + 2a_1 + a_0 = 1$$
$$\text{p(3): } 9a_2 + 3a_1 + a_0 = 0$$

These linear equations can be combined to form a linear system of the form Ax = b:

$$\begin{bmatrix} 1 & 1 & 1 \\ 4 & 2 & 1 \\ 9 & 3 & 1 \end{bmatrix} * \begin{bmatrix} a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}$$

Using Gaussian Elimination, we can solve for the coefficients by forming the augmented matrix [A b]:

$$\rightarrow \begin{bmatrix} 1 & 1 & 1 & 2 \\ 4 & 2 & 1 & 1 \\ 9 & 3 & 1 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 1 & 2 \\ 0 & 2 & 3 & 7 \\ 0 & -6 & -8 & -18 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 1 & 2 \\ 0 & 2 & 3 & 7 \\ 0 & 0 & 1 & 3 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 1 & 2 \\ 0 & 2 & 0 & -2 \\ 0 & 0 & 1 & 3 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 1 & 1 & 1 & 2 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 3 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 3 \end{bmatrix}$$

If we split the result matrix back into the matrix A and the vector b, then we can easily solve Ax = b:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 3 \end{bmatrix}$$

It is easy to see that $a_2 = 0$,$a_1 = -1$, and $a_0 = 3$, which means that the polynomial is:

3 - x

Problem 5)

(20 pts) Consider the array 25, 34, 63, 29, 66, 47, 12, 17. Apply the Partition (Split) procedure in Quicksort, as described in class, to this array using the first element as the pivot.

Starting with our initial array, we start at A[p + 1] (34) and go right until we find the first element that is less than 25 (A[p]), which is 12. We swap them as follows:

[25, 34, 63, 29, 66, 47, 12, 17] → [25, 12, 63, 29, 66, 47, 34, 17]

We resume split at A[p + i + 1] (one after where we stopped, so 17) and find that 17 is less than 25. We then swap it with the first element of the => section:

[25, 12, 63, 29, 66, 47, 34, 17] → [25, 12, 17 29, 66, 47, 34, 63]

We have now reached the end of the array, so all that is left to do is to swap 25 (A[p]) with the last element in the < section, which is 17:

[25, 12, 17, 29, 66, 47, 34, 63] → [17, 12, 25, 29, 66, 47, 34, 63]

Final Array: [17, 12, 25, 29, 66, 47, 34, 63]

We know that the above is correct because every element to the left of 25 is less than it while every element to the right of 25 is greater than it.