**Author:** Parth Shah

**Title:** Chapter 4 Solutions

## Notes

## 4.1 Maximum Subarray Problem

**Problem 4.1-1** What does find maximum subarray return when all elements of A are negative.

**Solution 4.1-1** It still returns the best buy and sell. If all profits are negative then it returns the buy-sell combo with the least loss.

**Problem 4.1-2** Write pseudocode for the brute-force method of solving the maximum-subarray problem. Your procedure should run in $\Theta(n^2)$ time.

**Solution 4.1-2**

```
procedure BruteForceMaxSubArray(A)
    $max = -\infty$
    for $i$ from 1 to $A.length$ do
        for $j$ from $i + 1$ to $A.length$ do
            if $A[j] - A[i] > max$ then
                $max = A[j] - A[i]$
```

**Problem 4.1-3** Implement both the brute-force and recursive algorithms for the maximum-subarray problem on your own computer. What problem size $n_0$ gives the crossover point at which the recursive algorithm beats the brute-force algorithm? Then, change the base case of the recursive algorithm to use the brute-force algorithm whenever the problem size is less than $n_0$. Does that change the crossover point?

**Solution 4.1-3** Skip. It should improve the crossover because then the base case is faster which speeds up the recursive version.

**Problem 4.1-4** Suppose we change the definition of the maximum-subarray problem to allow the result to be an empty subarray, where the sum of the values of an empty subarray is 0. How would you change any of the algorithms that do not allow empty subarrays to permit an empty subarray to be the result?

**Solution 4.1-4** Add a statement at the end to check if the value is negative. If it is, replace it with the empty subarray which will by definition give a greater value.

---

**Problem 4.1-5** Develop a nonrecursive, linear-time algorithm for the maximum-subarray problem.

**Solution 4.1-5** Basically keep tab of a running min and a running max stock price. This is valid because if we encounter a new min than all values after will give a greater difference if you use the new min. Therefore only 1 value is required to keep a tab of for the buy point. At each index you check the value of the buy sell combo. If it improves the buy sell max then update the running max stock buy sell.

---

## 4.2 Strassen's Algorithm For Matrix Multiplication

---

**Problem 4.2-1** Use Strassen's algorithm to compute the following matrix product.

$$\begin{bmatrix} 1 & 3 \\ 7 & 5 \end{bmatrix} \begin{bmatrix} 6 & 8 \\ 4 & 2 \end{bmatrix}$$

**Solution 4.2-1** $M_1 = (1+5)(6+2) = 48$

$M_2 = (7+5)(6) = 72$
$M_3 = 1(8-2) = 6$
$M_4 = 5(4-6) = -10$
$M_5 = (1+3)(2) = 8$
$M_6 = (7-1)(6+8) = 84$
$M_7 = (3-5)(4+2) = -12$

$C_{1,1} = 48 + (-10) - 8 + (-12) = 18$
$C_{1,2} = 6 + 8 = 14$
$C_{2,1} = 62$
$C_{2,2} = 48 - 72 + 6 + 84 = 66$

---

**Problem 4.2-2** Write pseudocode for Strassen's Algorithm.

**Solution 4.2-2**

---

**Problem 4.2-3** How would you modify Strassen's algorithm to multiply $nxn$ matrices in which $n$ is not an exact power of 2? Show that the resulting algorithm runs in time $O(n^{\lg 7})$.

**Solution 4.2-3**

---

**Problem 4.2-4** What is the largest k such that if you can multiply $3x3$ matrices using k multiplications (not assuming commutativity of multiplication), then you can multiply $nxn$ matrices in time $O(n^{\lg 7})$? What would the running time of this algorithm be?

**Solution 4.2-4**

---

**Problem 4.2-5** V. Pan has discovered a way of multiplying $68 = x68$ matrices using 132,464 multiplications, a way of multiplying $70x70$ matrices using 143,640 multiplications, and a way of multiplying $72x72$ matrices using 155,424 multiplications. Which method yields the best asymptotic running time when used in a divide-and-conquer matrix-multiplication algorithm? How does it compare to Strassen's algorithm?

**Solution 4.2-5**

---

**Problem 4.2-6** How quickly can you multiply a $knxn$ matrix by an $nxkn$ matrix, using Strassen's algorithm as a subroutine? Answer the same question with the order of the input matrices reversed.

**Solution 4.2-6**

---

**Problem 4.2-7** Show how to multiply the complex numbers $a+bi$ and $c+di$ using only three multiplications of real numbers. The algorithm should take $a, b, c$, and $d$ as input and produce the real component $ac - bd$ and the imaginary component $ad - bc$ separately.

**Solution 4.2-7**

---

## 4.3 Substitution Method for Solving Recurrences

---

**Problem 4.3-1** Show that the solution of $T(n) = T(n-1) + n$ is $O(n^2)$.

**Solution 4.3-1**

---

**Problem 4.3-2** Show that the solution of $T(n) = T(\lceil n/2 \rceil) + 1$ is $O(\lg n)$.

**Solution 4.3-2**

---

**Problem 4.3-3** We saw that the solution of $T(n) = 2T(\lceil n/2 \rceil) + n$ is $O(n \lg n)$. Show that the solution of this recurrence is also $\Omega(n \lg n)$. Conclude that the solution is $\Theta(n \lg n)$.

**Solution 4.3-3**

**Problem 4.3-4** Show that by making a different inductive hypothesis, we can overcome the difficulty with the boundary condition $T(1) = 1$ for recurrence (4.19) without adjusting the boundary conditions for the inductive proof.

**Solution 4.3-4**

**Problem 4.3-5** Show that $\Theta(n \lg n)$ is the solution to the "exact" recurrence (4.3) for merge sort.

**Solution 4.3-5**

**Problem 4.3-6** Show that the solution to $T(n) = 2T(\lceil n/2 \rceil + 17) + n$ is $O(n \lg n)$.

**Solution 4.3-6**

**Problem 4.3-7** Using the master method in Section 4.5, you can show that the solution to the recurrence $T(n) = 4T(n/3) + n$ is $T(n) = \Theta(n^{log_3 4}$. Show that a substitution proof with the assumption $T(n) = cn^{log_3 4}$ fails. Then show how to subtract off a lower-order term to make a substitution proof work.

**Solution 4.3-7**

**Problem 4.3-8** Using the master method in Section 4.5, you can show that the solution to the recurrence $T(n) = 4T(n/2) + n^2$ is $T(n) = \Theta(n^2)$. Show that a substitution proof with the assumption $T(n) = cn^2$ fails. Then show how to subtract off a lower-order term to make a substitution proof work.

**Solution 4.3-8**

**Problem 4.3-9** Solve the recurrence $T(n) = 3T(\sqrt{n}) + \log n$. Your solution should be asymptotically tight. Do not worry about whether values are integral.

**Solution 4.3-9**

## 4.4 Recursion Tree Method for Solving Recurrences

**Problem 4.4-1** Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = 3T(\lfloor n/2 \rfloor) + n$. Use the substitution method to verify your answer.

**Solution 4.4-1**

**Problem 4.4-2** Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = T(n^2) + n^2$. Use the substitution method to verify your answer.

**Solution 4.4-2**

---

**Problem 4.4-3** Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = 4T(n/2 + 2) + n$. Use the substitution method to verify your answer.

**Solution 4.4-3**

---

**Problem 4.4-4** Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = 2T(n-1) + 1$. Use the substitution method to verify your answer.

**Solution 4.4-4**

---

**Problem 4.4-5** Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = T(n-1) + T(n-2) + n$. Use the substitution method to verify your answer.

**Solution 4.4-5**

---

**Problem 4.4-6** Argue that the solution to the recurrence $T(n) = T(n/3) + T(2n/3) + cn$, where c is a constant, is $\Omega(n \lg n)$ by appealing to a recursion tree.

**Solution 4.4-6**

---

**Problem 4.4-7** Draw the recursion tree for $T(n) = 4T(\lfloor n/2 \rfloor) + cn$, where $c$ is a constant, and provide a tight asymptotic bound on its solution. Verify your bound by the substitution method.

**Solution 4.4-7**

---

**Problem 4.4-8** Use a recursion tree to give an asymptotically tight solution to the recurrence $T(n) = T(n-a) + T(a) + cn$, where $a \geq 1$ and $c > 0$ are constants.

**Solution 4.4-8**

---

**Problem 4.4-9** Use a recursion tree to give an asymptotically tight solution to the recurrence $T(n) = T(\alpha n) + T((1-\alpha)n) + cn$, where $\alpha$ is a constant in the range $0 < \alpha < 1$ and $c > 0$ is also a constant.

**Solution 4.4-9**

---

## 4.5 Master Theorem for Solving Recurrences

---

**Problem 4.5-1** Use the master method to give tight asymptotic bounds for the following recurrences.

**a)** $T(n) = 2T(n/4) + 1$

**Solution**

**b)** $T(n) = 2T(n/4) + \sqrt{n}$

**Solution**

**c)** $T(n) = 2T(n/4) + n$

**Solution**

**d)** $T(n) = 2T(n/4) + n^2$

**Solution**

---

**Problem 4.5-2** Professor Caesar wishes to develop a matrix-multiplication algorithm that is asymptotically faster than Strassen's algorithm. His algorithm will use the divide- and-conquer method, dividing each matrix into pieces of size $n/4xn/4$, and the divide and combine steps together will take $\Theta(n^2)$ time. He needs to determine how many subproblems his algorithm has to create in order to beat Strassen's algorithm. If his algorithm creates a subproblems, then the recurrence for the running time $T(n)$ becomes $T(n) = aT(n/4) + \Theta(n^2)$. What is the largest integer value of a for which Professor Caesar's algorithm would be asymptotically faster than Strassen's algorithm?

**Solution 4.5-2**

---

**Problem 4.5-3** Use the master method to show that the solution to the binary-search recurrence $T(n) = T(n/2) + \Theta(1) is T(n) = \Theta(\lg n)$. (See Exercise 2.3-5 for a description of binary search.)

**Solution 4.5-3**

---

**Problem 4.5-4** Can the master method be applied to the recurrence $T(n) = 4T(n/2) + n^2 \lg n$? Why or why not? Give an asymptotic upper bound for this recurrence.

**Solution 4.5-4**

---

**Problem 4.5-5** Consider the regularity condition $af(n/b) \leq cf(n)$ for some constant $c < 1$, which is part of case 3 of the master theorem. Give an example of constants $a \geq 1$ and $b > 1$ and a function $f(n)$ that satisfies all the conditions in case 3 of the master theorem except the regularity condition.

**Solution 4.5-5**

---

## 4.6 Proof of Master Theorem

---

**Problem 4.6-1** Give a simple and exact expression for $n_j$ in equation (4.27) for the case in which $b$ is a positive integer instead of an arbitrary real number.

**Solution 4.6-1**

---

**Problem 4.6-2** Show that if $f(n) = \Theta(n^{log_b a} \lg^k n$, where $k \geq 0$, then the master recurrence has solution $T(n) = \Theta(n^{log_b a} \lg^{k+1} n)$. For simplicity, confine your analysis to exact powers of $b$.

**Solution 4.6-2**

---

**Problem 4.6-3** Show that case 3 of the master theorem is overstated, in the sense that the regularity condition $af(n/b) \leq cf(n)$ for some constant $c < 1$ implies that there exists a constant $\epsilon > 0$ such that $f(n) = \Omega(n^{log_b a+\epsilon})$.

**Solution 4.6-3**

---

## Problems

---

**Problem 4-1** Give as tight as possible asymptotic upper and lower bounds for the following:

**a)** $T(n) = 2T(n/2) + n^4$

**Solution**

**b)** $T(n) = T(7n/10) + n$

**Solution**

**c)** $T(n) = 16T(n/4) + n^2$

**Solution**

**d)** $T(n) = 7T(n/3) + n^2$

**Solution**

**e)** $T(n) = 7T(n/2) + n^2$

**Solution**

**f)** $T(n) = 2T(n/4) + \sqrt{n}$

**Solution**

**g)** $T(n) = T(n-2) + n^2$

**Solution**

---

**Problem 4-2** We have assumed that passing parameters takes constant time. Pointer to array is usually passed. But lets see the implication of three different parameter-passing strategies.

1. An array is passed by pointer. Time = $\Theta(1)$.

2. An array is passed by copying. Time = $\Theta(N)$, where $N$ is the size of the array.

3. An array is passed by copying only the subrange that might be accessed by the called procedure. Time = $\Theta(q - p + 1)$ if the subarray $A[p...q]$ is passed.

**a)** Consider the recursive binary search algorithm for finding a number in a sorted array (see Exercise 2.3-5). Give recurrences for the worst-case running times of binary search when arrays are passed using each of the three methods above, and give good upper bounds on the solutions of the recurrences. Let $N$ be the size of the original problem and $n$ be the size of a subproblem.

**Solution**

**b)** Redo part (a) for the MERGE-SORT algorithm from Section 2.3.1.

**Solution**

---

**Problem 4-3** Give as tight as possible asymptotic upper and lower bounds for the following:

**a)**

**Solution**

**b)**

**Solution**

**c)**

**Solution**

**d)**

**Solution**

**e)**

**Solution**

**f)**

**Solution**

**g)**

**Solution**

**h)**

**Solution**

**i)**

**Solution**

**j)**

**Solution**

---

**Problem 4-4**

**a)**

**Solution**

**b)**

**Solution**

**c)**

**Solution**

**d)**

**Solution**

---

**Problem 4-5**

**a)**

**Solution**

**b)**

**Solution**

**c)**

**Solution**

---

**Problem 4-6**

**a)**

**Solution**

**b)**

**Solution**

**c)**

**Solution**

**d)**

**Solution**

**e)**

**Solution**

---