Universidad Centroamericana "José Simeón Cañas"
Facultad de Ingeniería y Arquitectura
Departamento de Electrónica e Informática
Análisis de Algoritmos — Ciclo 02/2023
Guía de Ejercicios No. 1 — Algoritmos, eficiencia y recurrencias

Algoritmos y su eficiencia

1. Give at least 10 real world examples that requires sorting.
2. Other than speed, what other measures of efficiency might one use in real-world settings.
3. Select a data structure that you have used previously, and discuss its strengths and limitations.
4. Do the following:
   a) Research on the shortest path problem (the problem, not the algorithm that solves it).
   b) Research on the traveling salesman problem (idem).
   c) How are these two problems similar?
   d) How are they different?
5. Come up with a real world problem in which only the best solution will do.
6. Come up with a real world problem in which a solution that is "approximately" the best is good enough.
7. Give an example of a software application that requires algorithms to function properly, discuss the function of the algorithms involved.
8. Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, insertion sort runs in $8n^2$ steps, while merge sort runs in $64nlog_2n$ steps. For which values of n does insertion sort beat merge sort?
9. What is the smallest value of n such that an algorithm whose running time is $100n^2$ runs faster than an algorithm whose running time is 2^n on the same machine?
10. For each function $f(n)$ and time t in the following table, determine the largest size n of a problem that can be solved in time t, assuming that the algorithm to solve the problem takes $f(n)$ microseconds.

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
log_2n							
$\sqrt{n}$							
n							
$nlog_2n$							
n^2							
n^3							
2^n							
$n!$							

Sorting Problem

1. Illustrate the execution of INSERTION-SORT on the array $A = [31, 41, 59, 26, 41, 58]$.

2. Rewrite the INSERTION-SORT procedure to sort into nonincreasing instead of decreasing order.

3. Consider the ***searching problem***:
   Input: A sequence of n numbers $A = [a_1, a_2, ..., a_n]$ and a value v.

   Output: An index i such that $v = A[i]$ or the special value NIL if v does not appear in A.

   Write pseudocode for ***linear search***, which scans through the sequence, looking for v.

4. Discuss whether or not the pseudocode written in the previous exercise is correct.

5. Consider the problem of adding two n-bit binary integers, stored in two n-element arrays A and B. The sum of the two integers should be stored in binary form in an $(n + 1)$-element array C. State the problem formally and write pseudocode for adding the two integers.

6. Consider sorting n numbers stored in an array A by first finding the smallest element of A and exchanging it with the element in $A[1]$. Then find the second smallest element of A, and exchange it with $A[2]$. Continue in this manner for the first $n - 1$ elements of A.

   a) Write pseudocode for this algorithm.

   b) Why does it need to run only for the first $n-1$ elements, rather than for all n elements?

   c) Give the best-case and worst-case running times in Θ-notation.

7. Illustrate the operation of MERGE SORT on the array $A = [3, 41, 52, 26, 38, 57, 9, 49]$

8. Rewrite the MERGE function so that it does not use the *infinity* values, instead stopping once either array L or R has had all its elements copied back to A and the copying the remainder of the other array back into A.

9. We can express insertion sort as a recursive procedure as follows: In order to sort $A[1..n]$, we recursively sort $A[1..n - 1]$ and then insert $A[n]$ into the sorted array. Write a recurrence for the running time of this recursive version of insertion sort.

10. Referring back to the searching problem, observe that if the sequence A is sorted, we can check the midpoint of the sequence against v and eliminate half of the sequence from further consideration.

    a) If this is repeated until v is found, how is the algorithm called?

    b) Write pseudocode for such algorithm.

    c) Argue that the worst-case running time of this algorithm is $\Theta(log_2 n)$.

11. In INSERTION SORT, change the linear search the while loops uses for the search algorithm from the previous exercise. Is this new version $\Theta(nlog_2 n)$.

12. Although MERGE SORT runs in $\Theta(nlog_2 n)$ worst-case time and INSERTION SORT runs in $\Theta(n^2)$ worst-case time, the constant factors in inserstion sort can make it faster in practice for small problem sizes on many machines. Thus, it makes sense to use insertion sort within merge sort when subproblems become sufficiently small. Consider a modification to merge

sort in which n/k sublists of length k are sorted using insertion sort and the merged using the standard merging mechanism, where k is a value to be determined.

a) Show that insertion sort can sort the n/k sublists, each of length k, in $\Theta(nk)$ worst-case time.

b) Show how to merge the sublists in $\Theta(nlog_2(n/k))$ worst-case time.

c) Given that the modified algorithm runs in $\Theta(nk + nlog_2(n/k))$ worst-case time, what is the largest value of k as a function of n for which the modified algorithm has the same running time as standard merge sort, in terms of Θ-notation?

d) How should we choose k in practice?

Orders of Growth

1. Express the function $\frac{n^3}{1000} - 100n^2 - 100n + 3$ in terms of Θ-notation.

2. Consider again the ***linear search*** algorithm.

   a) How many elements of the input sequence need to be checked on the average assuming that the element being searched for is equally likely to be any element in the array?

   b) How could it be possible that two elements in the array have different chances of being searched for?

   c) How about in the worst case?

   d) What are the best and worst cases in Θ-notation? Justify your answer.

3. How can we modify almost any algorithm to have a good best-case running time?

4. Construct a $\Theta(nlog_2n)$-time algorithm that, given a set S of n integers and another integer x, determines whether or not there exist two elements in S whose sum is exactly x.

5. The following code fragment implements Horner's rule for evaluating a polynomial

$$\begin{aligned} P(x) &= \sum_{k=0}^{n} a_k x^k \\ &= a_0 + x(a_1 + x(a_2 + ... + x(a_{n-1} + xa_n)...)) \end{aligned}$$

   given the coefficients $a_0, a_1, ..., a_n$ and a value for x:

```
y=0
for i = n down to 0
        y = a_i + x*y
```

   a) In terms of Θ-notation, what is the running time of this code fragment for Horner's rule?

   b) Write pseudocode to implement the polynomial evaluation algorithm that computes each term of the polynomial from scratch.

   c) What is the running time of this pseudocode?

   d) How does it compare to Horner's rule?

6. Let $A[1..n]$ be an array of n distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair (i, j) is called an *inversion* of A.

a) List the five inversions of the array $[2, 3, 8, 6, 1]$.

b) What array with elements from the set $\{1, 2, ..., n\}$ has the most inversions? How many does it have?

c) What is the relationship between the running time of insertion sort and the number of inversions in the input array? Justify your answer.

d) Give an algorithm that determines the number of inversions in any permutation on n elements in $\Theta(nlog_2n)$ worst-case time. (*Hint:* Modify MERGE SORT).

7. Let $f(n)$ and $g(n)$ be asymptotically nonnegative functions. Using the basic definition of Θ-notation, prove that $max(f(n), g(n)) = \Theta(f(n) + g(n))$.

8. Show that for any real constants a and b, where $b > 0$, $(n + a)^b = \Theta(n^b)$.

9. Explain why the statement *"The running time of algorithm A is at least $O(n^2)$"* is meaningless.

10. Is $2^{n+1} = O(2^n)$?

11. Is $2^{2n} = O(2^n)$?

12. Prove that the running time of an algorithm is $\Theta(g(n))$ if and only if its worst-case running time is $O(g(n))$ and its best-case running time is $\Omega(g(n))$.

13. Prove that $o(g(n)) \cap \omega(g(n)) = \emptyset$.

14. Let

$$p(n) = \sum_{i=0}^{d} a_i n^i$$

where $a_d > 0$, be a degree-d polynomial in n, and let k be a constant. Use the definitions of the asymptotic notations to prove the following properties.

a) If $k >= d$, then $p(n) = O(n^k)$.

b) If $k <= d$, then $p(n) = \Omega(n^k)$.

c) If $k = d$, then $p(n) = \Theta(n^k)$.

d) If $k > d$, then $p(n) = o(n^k)$.

e) If $k < d$, then $p(n) = \omega(n^k)$.

15. Indicate, for each pair of expressions (A, B) in the table below, whether A is O, o, Ω, ω, or Θ of B. Assume that $k >= 1$, $\epsilon > 0$, and $c > 1$ are constants. Your answer should be in the form of the table with "yes" or "no" written in each box.

A	B	O	o	Ω	ω	Θ
$log_2^k n$	n^ϵ					
n^k	c^n					
$\sqrt{n}$	$n^{\sin n}$					
2^n	$2^{n/2}$					
$n^{log_2 c}$	$c^{log_2 n}$					
$log_2(n!)$	$log_2(n^n)$					

16. Let $f(n)$ and $g(n)$ be asymptotically positive functions. Prove or dispose each of the following conjectures:

a) $f(n) = O(g(n))$ implies $g(n) = O(f(n))$.

b) $f(n) + g(n) = \Theta(min(f(n), g(n)))$.

c) $f(n) = O(g(n))$ implies $log_2(f(n)) = O(log_2(g(n)))$, where $log_2(g(n)) >= 1$ and $f(n) >= 1$ for all sufficiently large n.

d) $f(n) = O(g(n))$ implies $2^{f(n)} = O(2^{g(n)})$.

e) $f(n) = O((f(n))^2)$.

f) $f(n) = O(g(n))$ implies $g(n) = \Omega(f(n))$.

g) $f(n) = \Theta(f(n/2))$.

h) $f(n) + o(f(n)) = \Theta(f(n))$.

Max Subarray Problem

The pseudo code for this problem is as follows:

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 1  left-sum = −∞
 2  sum = 0
 3  for i = mid downto low
 4       sum = sum + A[i]
 5       if sum > left-sum
 6            left-sum = sum
 7            max-left = i
 8  right-sum = −∞
 9  sum = 0
10  for j = mid + 1 to high
11       sum = sum + A[j]
12       if sum > right-sum
13            right-sum = sum
14            max-right = j
15  return (max-left, max-right, left-sum + right-sum)
```

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
 1  if high == low
 2       return (low, high, A[low])          // base case: only one element
 3  else mid = ⌊(low + high)/2⌋
 4       (left-low, left-high, left-sum) =
             FIND-MAXIMUM-SUBARRAY(A, low, mid)
 5       (right-low, right-high, right-sum) =
             FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
 6       (cross-low, cross-high, cross-sum) =
             FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
 7       if left-sum ≥ right-sum and left-sum ≥ cross-sum
 8            return (left-low, left-high, left-sum)
 9       elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
10            return (right-low, right-high, right-sum)
11       else return (cross-low, cross-high, cross-sum)
```

1. What does FIND-MAXIMUM-SUBARRAY return when all elements of A are negative?

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

3. Use the following ideas to develop a nonrecursive, linear-time algorithm for the maximum-subarray problem. Start at the left end of the array, and progress toward the right, keep track

of the maximum subarray seen so far. Knowing a maximum subarray of $A[1..j]$, extend the answer to find a maximum subarray ending at index $j+1$ by using the following observation: a maximum subarray of $A[1..j+1]$ is either a maximum subarray of $A[1..j]$ or a subarray $A[i..j+1]$, for some $1 <= i <= j+1$. Determine a maximum subarray of the form $A[i..j+1]$ in constant time based on knowing a maximum subarray ending at index j.

Strassen's Method

The traditional method for multiplying two matrices A and B is:

```
SQUARE-MATRIX-MULTIPLY(A, B)
1  n = A.rows
2  let C be a new n × n matrix
3  for i = 1 to n
4      for j = 1 to n
5          c_ij = 0
6          for k = 1 to n
7              c_ij = c_ij + a_ik · b_kj
8  return C
```

A recursive version of this procedure is:

```
SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)
 1  n = A.rows
 2  let C be a new n × n matrix
 3  if n == 1
 4      c_11 = a_11 · b_11
 5  else partition A, B, and C as in equations (4.9)
 6      C_11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_11, B_11)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_12, B_21)
 7      C_12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_11, B_12)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_12, B_22)
 8      C_21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_21, B_11)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_22, B_21)
 9      C_22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_21, B_12)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A_22, B_22)
10  return C
```

Given all of this, Strassen's method consists of four steps:

1. Divide the input matrices A and B and output matrix C into $\frac{n}{2} \times \frac{n}{2}$ submatrices. This step takes $\Theta(1)$ time by index calculation, just as in SQUARE-MATRIX-MULTIPLY-RECURSIVE.

2. Create 10 matrices $S_1, S_2, ..., S_{10}$ each of which is $\frac{n}{2} \times \frac{n}{2}$ and is the sum or difference of two matrices created in step 1. We can create all 10 matrices in $\Theta(n^2)$ time.

3. Using the submatrices created in step 1 and the 10 matrices created in step 2, recursively compute seven matrix products $P_1, P_2, ..., P_7$. Each matrix P_i is $\frac{n}{2} \times \frac{n}{2}$.

4. Compute the desired submatrices $C_{11}, C_{12}, C_{21}, C_{22}$ of the result matrix C by adding and substracting various combinations of the P_i matrices. We can compute all four submatrices in $\Theta(n^2)$ time.

==========

1. Use Strassen's algorithm to compute the matrix product

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

   Show your work.

2. Write pseudocode for Strassen's algorithm.

3. How would you modify Strassen's algorithm to multiply $n \times n$ matrices in which n is not an exact power of 2? Show that the resulting algorithm runs in time $\Theta(n^{log_2 7})$.

4. What is the largest k such that if you can multiply 3×3 matrices using k multiplications, then you can multiply $n \times n$ matrices in time $o(n^{log_2 7})$? What would the running time of this algorithm be?

Recurrences

1. Use mathematical induction to show that when n is an exact power of 2, the solution of the recurrence

$$T(n) = \begin{cases} 2 & if\, n = 2, \\ 2T(\frac{n}{2}) + n & if\, n = 2^k,\ for\, k > 1 \end{cases}$$

   is $T(n) = nlog_2 n$.

2. Show that the solution of $T(n) = T(n-1) + n$ is $O(n^2)$. Use all methods.

3. Show that the solution of $T(n) = T(\lceil n/2 \rceil) + 1$ is $O(log_2 n)$. Use all methods.

4. We saw that the solution of $T(n) = 2T(\lfloor n/2 \rfloor) + n$ is $O(nlog_2 n)$. Show that the solution of this recurrence is also $\Omega(nlog_2 n)$. Conclude that the solution is $\Theta(nlog_2 n)$.

5. Show that the solution to $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$ is $O(nlog_2 n)$. Use all methods.

6. Using the Master Theorem you can show that the solution to the recurrence $T(n) = 4T(n/3) + n$ is $\Theta(n^{log_3 4})$. Show that a substitution method with hypothesis $T(n) <= cn^{log_3 4}$ fails.

   a) How could you make it work?

7. Using the Master Theorem you can show that the solution to the recurrence $T(n) = 4T(n/2) + n^2$ is $\Theta(n^2)$. Show that a substitution method with hypothesis $T(n) <= cn^2$ fails.

   a) How could you make it work?

8. Solve the recurrence $T(n) = 3T(\sqrt{n}) + log_{10} n$. (*Hint:* A change of variable could help).

9. 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 justify the answer.

10. Use a recursion tree to determine a good asymptotic upper bound on the recurrence $T(n) = 3T(n/2) + n^2$.Use the substitution method to justify the answer.

11. 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 justify the answer.

12. 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 justify the answer.

13. 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 justify the answer.

14. Argue that the solution to the recurrence $T(n) = T(n/3)+T(2n/3)+cn$, where c is a constant, is $\Omega(nlog_2n)$ by appealing to a recursion tree.

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

16. Use a recursion tree to give an asymptotic solution to the recurrence $T(n) = T(n-a) + T(a) + cn$, where $a >= 1$ and $c > 0$ are constants.

17. Use a recursion tree to give an asymptotic solution to the recurrence $T(n) = T(\alpha n) + T((1-\alpha)n) + cn$, where α is a constant in the range $0 < \alpha < 1$ and $c > 0$ is also a constant.

18. Use the Master Method to give asymptotic bounds for the following recurrences:

    a) $T(n) = 2T(n/4) + 1$.
    b) $T(n) = 2T(n/4) + \sqrt{n}$.
    c) $T(n) = 2T(n/4) + n$.
    d) $T(n) = 2T(n/4) + n^2$.

19. Show that the recurrence for the Binary Search algorithm is $T(n) = T(n/2) + \Theta(1)$.

20. Show that the solution to the recurrence for the Binary Search algorithm is $\Theta(log_2n)$.

21. Can the Master Method be applied to the recurrence $T(n) = 4T(n/2) + n^2log_2n$? Why or why not?

22. Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences: (Assume $T(n)$ is constant for sufficiently small n)

    a) $T(n) = 2T(n/2) + n^4$.
    b) $T(n) = T(7n/10) + n$.
    c) $T(n) = 16T(n/4) + n^2$.
    d) $T(n) = 7T(n/3) + n^2$.
    e) $T(n) = 7T(n/2) + n^2$.
    f) $T(n) = 2T(n/4) + \sqrt{n}$.
    g) $T(n) = 2T(n-2) + n^2$.
    h) $T(n) = 4T(n/3) + nlog_2n$.
    i) $T(n) = 3T(n/3) + \frac{n}{log_2n}$.
    j) $T(n) = 4T(n/2) + n^2\sqrt{n}$.
    k) $T(n) = 3T(n/3 - 2) + n/2$.
    l) $T(n) = 2T(n/2) + \frac{n}{log_2n}$.
    m) $T(n) = T(n/2) + T(n/4) + T(n/8) + n$.

n) $T(n) = T(n-1) + 1/n$.

ñ) $T(n) = T(n-1) + log_2 n$.

o) $T(n) = T(n-2) + \frac{1}{log_2 n}$.

p) $T(n) = \sqrt{n}T(\sqrt{n}) + n$.

Mathematical induction (optional)

1. Prove that for each odd number n, the number $n^2 - 1$ is divisible by 8.
2. Prove that $n! > 3^{n-2}$, for every natural number $n >= 3$.
3. Prove that 3 is a divisor of $n^3 + 2n$ for every positive integer n.
4. Prove that if n is a positive integer, then the following is true

$$1^3 + 2^3 + ... + n^3 = \frac{n^2(n+1)^2}{4}$$

5. The game of Nim is played between two people with the following rules: n tokens are put on the table, each player in its turn an take 1, 2, or 3 tokens. The player that takes the last token loses. Show that the first player has a winning strategy as long as $n \neq 1\,(mod\,4)$.
6. Prove that the total number of diagonals that a convex polygon of n sides has, where $n >= 3$, is equal to $\frac{n(n-3)}{2}$.
7. Prove that for every positive integer n

$$\frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + ... + \frac{1}{n(n+1)} = \frac{n}{n+1}$$

8. Consider the Fibonacci sequence $F_1, F_2, ...$ and show that

$$(F_1)^2 + (F_2)^2 + ... + (F_n)^2 = F_n \cdot F_{n+1}$$

9. Prove that $(3n)! > 2^{6n-4}$ for every positive integer n.

Summations (optional)

1. Calculate the following summations:

a) $\sum_{i=1}^{5} i(i+1)$

b) $\sum_{n=0}^{3} 2^n$

c) $\sum_{r=1}^{100} 4r$

d) $\sum_{h=-3}^{10} (6h-1)$

e) $\sum_{z=2}^{4} log(z^4)$

2. Express with a phrase the meaning of the following summations:
Example: $\sum_{p=1}^{5} 3p$ is the sum of all the multiples of 3 from 3 to 15.

a) $\sum_{s=1}^{10} s^2$

b) $\sum_{q=1}^{n} \frac{x_q}{n}$

c) $\sum_{n=0}^{49}(2n+1)$

3. Express with a summation the following phrases:

   a) The sum of the first 10 even prime numbers.

   b) The sum of all multiples of 4 from 36 to 80.

   c) The sum of the square root of the first 20 odd prime numbers.

   d) The sum of the power of 2 of all natural numbers from 1 to 30.

   e) The scalar product of the vectors $(t_1, t_2, \ldots, t_m)$ and $(w_1, w_2, \ldots, w_m)$.