Chapter 14

Algorithms Exercises

The problems that follow are intended for self-study as the course proceeds. A good number of them will be assigned as homework problems during the semester.

14.1 Asymptotics and Recurrence Equations

1.0. Imagine pairs of programs requiring the operation counts given below. For each pair, determine the range of \( n \) where each program is faster (i.e., requires fewer operations). It is convenient to take the logarithms as base 2. (Note that for iii, a straightforward approach to find the \( n \) where the two function counts are equal yields a transcendental equation, which we cannot solve, in general. Nevertheless, algorithmic ideas and common sense can overcome this difficulty.)

(i) A program performing \( 32n \) steps and a program performing \( n \log n \) steps.
(ii) A program performing \( 32n \) steps and a program performing \( n^2 \) steps.
(iii) A program performing \( 32n \log n \) steps and a program performing \( n^2 \) steps.

Morals: (a) \( n \) grows much faster than \( \log n \).

(b) Asymptotics are just that: they do not necessarily apply to problems of reasonable size.

1.1. Order in terms of asymptotic size the following 10 functions: \( n^{\log n} \), \( 2 \), \( (\log n)^n \), \( n^{1/\log n} \), \( 1001n^{1001001} \), \( 1.0000001^n \), \( (1/2)^n \), \( 10010n + 1.0000000000001n \), \( (\log n)^{\log n} \), \( n^{\log \log n} \).

1.2. Write down the recurrence equations for the running times of the following functions, but DO NOT solve them.

a. function \( T(n : \text{integer}) : \text{integer} \);
   begin
   if \( n \leq 1 \)
   then return(1)
   else return\( (n \cdot T(n - 1)) \)
   end; \{T\}
b. function \( V(n : \text{integer}) : \text{integer}; \)
begin
  if \( n \leq 1 \)
    then return(1)
  else return(2 \cdot V(n - 1))
end: \{ V \}

c. function \( W(n : \text{integer}) : \text{integer}; \)
begin
  if \( n \leq 1 \)
    then return(1)
  else return(\( W(n - 1) + W(n - 1) \))
end: \{ W \}

d. function \( X(n : \text{integer}) : \text{integer}; \)
begin
  if \( n \leq 1 \) then return(1)
  elseif \( n = 2 \) then return(4)
  else return(3 \cdot X(n - 1) + X(n - 2) + n^2) \)
end: \{ X \}

1.3. Answer the following questions with true or false.
Given \( f = \Theta(g), r = \Theta(s) \) and all functions are positive, are the following true or false?

a) \( f + r = \Theta(g + s) \)
b) \( \max\{f, r\} = \Theta(g + s) \)
c) \( nf(n) = \Theta(n \cdot g(n)) \)
d) \( r \cdot f = \Theta(r \cdot f) \)
e) \( \min\{r, f\} = \Theta(g + s) \)
f) \( \min\{r, f\} = O(s) \)
g) \( f = O(g) \)

1.4. Find a function \( f \) that grows faster than any polynomial and slower than any exponential:
For all \( k, n^k = O(f(n)) \)
For all \( c > 1, f(n) = O(c^n) \)

NOTE: \( k \) and \( c \) must not appear in the expression for \( f \).
HINT: What is wrong with the answer \( n^{100} \), or \( 2^n \)?

1.5. Solve the following recurrence equations.
a. \( T(2) = 4 \)
   \( T(n) = T(n/2) + n \), \( n > 2 \) (assume \( n \) is a power of 2)
b. \( T(0) = 2 \)
   \( T(n) = T(n - 1) + 2n + 2 \), \( n > 0 \)
c. \( T(1) = 3 \)
   \( T(n) = 3T(n - 1) + 3^n \), \( n > 1 \)
d. \( T(1) = 1 \)
   \( T(n) = 2T(n/5) + n^{\log_5 2} \), \( n > 1 \) (assume \( n \) is a power of 5)
Remember to check that the solutions satisfy the given recurrence equations. Noting that an attempted solution does not satisfy the equations will receive more credit than failing to observe this.

1.6. Solve the following recurrence equations. In parts (a) and (b) write the solution as a sum.
   a. \( T(2) = 2 \)
      \[ T(n) = nT(n-1) + n, \quad n > 2 \]
   b. \( T(8) = 8 \)
      \[ T(n) = n^2T(n-1) + n, \quad n > 8 \]
   c. \( T(1) = 1 \)
      \[ \frac{T(n)}{T(n-1)} = 5n, \quad n > 1 \]
      Hint: this telescopes multiplicatively.

1.7. Derive the general solution to

   \[ T(1) = c_1 \]
   \[ a_nT(n) = b_nT(n-1) + c_n, \quad n > 1 \]

   Hints: You need a summing factor \( f_n \). Write down the requirement for \( f \) if it is to serve as a summing factor, and solve for \( f \). Notice that there is no initial condition for \( f \); you might as well let \( f_1 = 1 \).

1.8. Explain how to get, for fixed \( k \), a tight asymptotic estimate for \( \sum_{i=1}^{n} i^k \).

1.9. Characterize the asymptotic behavior of the solutions to
   a) \( T(n) = T(n/2) + n \log n, \quad n > 1 \) and \( T(1) = 1 \).
   b) \( T(n) = aT(n/5) + n^c, \quad n > 1 \) and an integer power of \( 5 \); \( T(1) = 1 \). (The solution has several cases depending on the values of \( a \) and \( c \).)

1.10. Solve the following recurrence equations.
   a. \( T(1) = 1 \)
        \[ \frac{T(n)}{T(n-1)} = 3n, \quad n > 1 \]
        Hint: this telescopes multiplicatively.
   b. \( T(1) = 4 \)
      \[ T(n) = 2^{n-1}T(n/2) + 2^{2n-1}, \quad n > 1 \]
      (Assume \( n \) is a power of \( 2 \))
   c. \( T(2) = 4 \)
      \[ T(n) = 4T(n/2) + n^2, \quad n > 3 \]
      (Assume \( n \) is a power of \( 2 \))
   d. \( T(4) = 4 \)
      \[ T(n) = T(n/2 + 1) + n - 2, \quad n > 4 \]
      (Assume \( n \) is a power of \( 2 \) plus \( 2 \))
Hint: You need a domain transformation \( d(k) = n \), and \( d(k - 1) = \frac{n}{2} + 1 \), so that the substitution \( S(k) = T(d(k)) \) yields \( S(k) = S(k-1) + \text{stuff} \). How do we solve two equations in the variables \( d \) and \( n \)? Answer: eliminate the \( n \) and get an equation for the function \( d \).

1.11. Consider the code fragment

```plaintext
function Rand(N);
    begin
        if \( N = 1 \) then return(1)
        else begin
            assign \( x \) one of the values 0, 1, 2, each with probability \( \frac{1}{3} \);
            if \( x = 0 \) then return(Rand(N))
            else if \( x = 1 \) then return(Rand(N - 1) + 1)
            else if \( x = 2 \) then return(Rand(N - 1) + Rand(N - 1) + 1)
        end;
    end;
```

Write down the recurrence equation for the expected running time of \( \text{Rand}(N) \). DO NOT solve it.

1.12. Consider the code fragment

```plaintext
function Prop(N)
    begin
        if \( N = 1 \) then return(1)
        else return(Prop(N - 1) + Prop(N - 1) + Prop(N - 1))
    end;
```

(a) What does \( \text{Prop}(N) \) compute?
(b) What is the running time of \( \text{Prop}(N) \)?
(c) What would be the running time if the second return statement in \( \text{Prop} \) were

```plaintext
return(3\text{Prop}(N - 1))?
```

1.13. Consider the following code fragment which returns \( \text{Prop}(N) \). Notice that the routine, for a given \( N \), saves the value of \( \text{Prop}(N) \) in table \( X[\ast] \), once it has been computed, and subsequently uses \( X \) to look up \( \text{Prop}(N) \), should the value be needed later.

Global

\( X[1 \ldots \infty] \) of integer,
\( \text{Index: integer;} \)
\( X[1] \leftarrow 1; \text{Index} \leftarrow 1; \quad /\!\!\!\!\!/ \text{ Initialize*/}

function Prop(N);
    begin
        if \( N \leq \text{Index} \) then return(\( X[N] \)) \quad /* Use table lookup*/
        /* Hint: what would be output if \text{print(Index)} \) were right here? */
        else begin
            Temp \( \leftarrow \) Prop(N - 1);
            /* Hint: what would be output if \text{print(Index)} \) were right here? */
            \( X[N] \leftarrow 2 \cdot \text{Temp} + \text{Prop}(N - 1) \);
            \( \text{Index} \leftarrow \max\{\text{Index}, N\} \);
            return(\( X[N] \))
        end;
    end;
```
end
end;

a. What does $Prop(N)$ compute?
b. What is the running time of $Prop(N)$?
c. Can the statement $Index \leftarrow \max\{Index, N\}$ be replaced by $Index \leftarrow N$?
1.14. Write and solve the recurrence equations for the expected running times of the following functions.

a. function $A(N)$
   
   begin
   if $N \leq 1$ then return(1)
   else begin
     let $x = \text{TRUE or FALSE}$ with probability $\frac{1}{2}$;
     if $x$ then return($A(N)$)
     else return($A(N - 1)$)
   end
   end;

b. function $B(N)$
   
   begin
   if $N \leq 1$ then return(1)
   else begin
     let $x = \text{TRUE or FALSE}$ with probability $\frac{1}{2}$;
     if $x$ then return($B(N)$)
     else return($B(N/2)$)
   end
   end;

c. function $C(N)$
   
   begin
   if $N \leq 1$ then return(1)
   else begin
     perform $N$ steps of work;
     let $x = \text{TRUE or FALSE}$ with probability $\frac{1}{2}$;
     if $x$ then return($C(N)$)
     else return($B(C(N/2))$) /* $B$ is defined in part b. */
   end
   end;

d. function $D(N)$
   
   begin
   if $N \leq 1$ then return(1)
   else begin
     perform $N$ steps of work;
     let $x = \text{TRUE with probability } p, \text{FALSE with probability } 1 - p$;
     if $x$ then return($D(N)$) else return($D(N - 1)$)
   end
   end;

1.15. Josephus's problem is as follows: Imagine $n$ individuals arranged in a circle. They are numbered from 1 to $n$ in clockwise (increasing) order. Person 1 starts out with the one and only token. The rule is: the person with the token removes his (clockwise) next neighbor from the circle and passes the token to his new (clockwise) neighbor. The game terminates when only one person
is left. Thus, if \( n = 10 \), people 1, 3, 5, 7, and 9 will remain after 5 passes, and the final winner turns out to be 5. Write a recurrence equation for the winning number NAME as a function of \( n \). (Note: you are to write an equation for the name, NOT the running time.) Do NOT solve it. Comments. The nicest formulation comes from considering what happens when \( n/2 \) deletions occur. Notice that for this problem, Big-Oh notation is useless; exact answers are required. There are 2 cases: A different recurrence is required for \( NAME(2n) \), and for \( NAME(2n + 1) \).

1.16 In this problem, you are to write (but not solve) the recurrence equation to compute the exact length of the side of an \( H \)-tree at level \( k \).

\( H_0 \), which is an \( H \)-tree at level zero, is defined to have a single vertex and be a square with a sidelength of 2 inches. The recursive way an \( H \)-tree is built is as follows: To build \( H_{k+1} \), an \( H \)-tree with level \( k + 1 \), take four \( H \)-trees of level \( k \), and three additional vertices and organize them into a square as shown: there is only one additional rule:

- The level-\( k \) trees must be placed two inches apart (horizontally and vertically).

Thus, an \( H \) tree at level 1 has a side length of 6 inches. The root of an \( H_k \) tree is always located right in the middle of the structure.

![Diagram of H-trees](image)

a) Write the recurrence equation for the length of a side of an \( H \) tree of level \( k \).

Be sure to include an initial condition.

b) Now write a recurrence equation for the number of nodes in an \( H \)-tree at level \( k \).

Be sure to include an initial condition.

c) Now solve the recurrence equations. You should see that \( H_k \) has \( \Theta(4^k) \) vertices and a side length of \( \Theta(2^k) \). Thus an \( H \)-tree of \( n \) vertices has an area of \( \Theta(n) \): this physical layout of binary trees uses an area that is linear in the number of vertices. Such a physical layout keeps nodes and edges separated by unit spacing, and only uses a constant amount of area per node, which is as good as it gets.

1.17 Let \( t \) be an arithmetic expression parsed as a binary tree. The leaves are numerical values, whereas each internal node is an arithmetic operator. You can assume that a leaf \( x \) has its value in \( x.val \) and an internal node \( x \) has the name of its operator stored as \( x.op \). You can also assume that each operator is binary, so that each internal node has two children. Assume that you can issue calls to the function \( eval(op, x, y) \), which returns the numerical value of the operator \( op \) as applied
to the numerical values $x$ and $y$.

Write a recursive function to evaluate any such tree $T$. You are not asked to give the code for $eval$, or do not worry about whether $3 - 2$ is $eval(-, 2, 3)$ or $eval(-, 3, 2)$. Just assume that your interpretation and the actual coding are consistent.

1.18 Let $L$ be a stack that contains an arithmetic expression in postorder order (see problem 1.17), so that the root of the binary expression tree is at the top of the stack. In the previous problem, for example, the stack would read, from top to bottom: $+ * + - 263 - * 9743$. Obviously, the $+$ is the first object to be removed.

We assume that each element of the stack is either a number or a binary operator. Assume that you can issue calls to the function $eval(op, x, y)$, which returns the numerical value of the operator $op$ as applied to the numerical values $x$ and $y$.

Complete the code below to be a recursive function that processes $L$.

```plaintext
function StackVal(var L:Stack):number;  { Please note that StackVal modifies stack L. }
    x ← PopFrom(L);
    if $x$ is a number then return($x$)
    else
        return(eval($x$, __________________________, __________________________))
    endif
end StackVal.
```

Please note that all arguments of a function such as $eval$ are evaluated left to right. Also, this code has no checks to see if the arithmetic expression in $L$ has a correct syntax.

1.19) Let $S$ be the list of nodes of an arbitrary tree, where the nodes are printed in postorder. Assume that the children of each node are accessed from left to right. Let $T$ be the list of nodes of the same tree, but this time the nodes are printed in preorder, and the children of each node are accessed from right to left.

a) what is the relationship between $S$ and $T$?

b) Can you explain, in a sentence or two of very high-level reasoning, why your answer in a) is correct?

c) Write an iterative solution for a modification of problem 1.18, where $L$ is assumed to contain the same postorder listing, but in reverse order, so that the operator at the root of the binary parse tree is at the bottom of the stack. Use a stack for temporary storage.

d) What are the efficiency advantages between the solutions to a) and c)?

A preorder listing of an arithmetic parse tree such as $S$ in part a) is called Polish Normal Form, and that where the sequence is reversed is called, naturally enough, Reverse Polish Normal Form. Unlike our regular infix-notation, Polish and Reverse Polish do not need parentheses.