HW#2 Solution, CSCI 323, Spring 2003
Department of Computer Science
Queens College
Dr. Song

Problem 1 Solution

Let S_i be a statement before the loop $i+1$.

- S_0: currentMax is the maximum of the first element of A, since currentMax = $A[0]$;
- S_{i-1}: assume currentMax is the maximum of the first i elements of A;
- In the i-th loop, currentMax is compared with $A[i]$, it will be replaced by $A[i]$ if less than $A[i]$. Therefore after the i-th loop, we have S_i: currentMax is the maximum of the first $i+1$ elements of A;
- When the entire for-loop finishes, currentMax will be the max of the first n elements of A, which is the maximum element of A.

Problem 2 Solution

The inner for-loop uses the following number of primitive operations:

$$1+(i+1)+2i+2i = 5i+2$$

The outer for-loop uses the following:

$$1+1+\sum_{i=1}^{n}[1+(5i+2)+2+1+2] = 2+8n+5\frac{n(n+1)}{2}$$

The total number of primitive operations is therefore:

$$\frac{5}{2}n^2+\frac{21}{2}n+4$$

Problem 3 Solution

See Algorithm 1.

- In the worst case, the exact number of primitive operations are:

$$T(n) = \begin{cases} 4, & n=1 \\ T(n/2)+12, & n>1 \end{cases}$$

$$\begin{aligned} T(n) &= T(n/2)+12 && (1)\\ T(n/2) &= T(n/4)+12 && (2)\\ T(n/4) &= T(n/8)+12 && (3)\\ &\cdots && (4)\\ T(n/2^{\log n}) &= 4 && (5) \end{aligned}$$

  Sum both sides up, $T(n) = 4+12\log n$.

- Since $T(n)$ is the worst case number of primitive operations, $T(n)$ is $O(\log n)$.

Algorithm 1 Binary-Search(x, A)

INPUT: a number x, a sorted sequence A of $n > 0$ numbers in non-decreasing order
OUTPUT: an index $0 \leq i \leq n-1$ or -1
if $n == 1$ **then**
  if $x == A[0]$ **then**
    return 0
  else
    return -1
  end if
end if
$i \leftarrow \lfloor \frac{n}{2} \rfloor$
if $x == A[i]$ **then**
  return i
else if $x < A[i]$ **then**
  return Binary-Search(x, $A[0...(i-1)]$)
else
  if $i < n-1$ **then**
    $k \leftarrow$ Binary-Search(x, $A[(i+1)...(n-1)]$)
    if $k \geq 0$ **then**
      return $i+1+k$
    end if
  end if
  return -1
end if

Problem 4 Solution

(G&T R-1.15)

Since

$$f(n) \leq c_1 g(n) \text{ for } n > n_1$$

and

$$d(n) \leq c_2 h(n) \text{ for } n > n_2$$

Take $c = \max\{c_1, c_2\}$ and $n_0 = \max\{n_1, n_2\}$, we have:

$$f(n) + d(n) \leq c(g(n) + h(n)) \text{ for } n > n_0$$

Therefore, $f(n) + d(n)$ is $O(g(n) + h(n))$.

Problem 5 Solution

(G&T R-1.15)

$$(n+1)^5 \leq (n+n)^5 = 32n^5$$

We can take a c such that

$$32n^5 \leq cn^5$$

For example, $c = 43$. For n_0, it can be any value greater or equal to 1.

Therefore $(n+1)^5$ is $O(n^5)$.

Problem 6 Solution

(G&T R-1.22)

To show the little-omega bound is true, we need to find an n_0 for any given $c > 0$.

$$n^2 \geq cn$$

$n > c$ will work in the above case. So we can choose any n_0 which is greater than c. Then we have shown that n^2 is $\omega(n)$.

Problem 7 Solution

(G&T C-1.10)

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + 3^2 + \cdots + n^2 \leq n^2 + n^2 + n^2 + \cdots + n^2 = n^3$$

Therefore, we can take $c = 1$ and $n_0 = 1$. Then the following satisfies:

$$\sum_{i=1}^{n} i^2 \leq cn^3 \text{ for } n \geq n_0$$

That is $\sum_{i=1}^{n} i^2$ is $O(n^3)$.

Problem 8 Solution

(G&T C-1.27)

Intuition: the algorithm works row by row but it doesn't inspect all the elements in a row. Once a zero is came across, the algorithm jumps to the next row, but stays at the same column and continues to scan the current row until a zero is found. It is like walking down stairs, except that the stairs may have different width.

In this way, the algorithm inspects at most $2n-1$ elements in A. Therefore, the complexity is $O(n)$. The following is the pseudo-code of the algorithm.

Algorithm 2 Find the row with the maximum number of 1's

```
INPUT n × n matrix A containing only 0's or 1's. In each row all 1's come before 0's.
OUTPUT the row index that contains the most number of 1's.
j ← 0
i_max ← 0
while i < n do
  if A[i, j] = 1 then
    i_max ← i
    while j < n and A[i, j] = 1 do
      j ← j + 1
    end while
    if j = n then
      return i_max
    end if
  end if
  i ← i + 1
end while
return i_max
```