CS 341, Fall 2020 A. Lubiw, A. Storjohan

ASSIGNMENT 1

DUE: Wednesday September 23, 5 PM. DO NOT COPY. ACKNOWLEDGE YOUR SOURCES.

Please read `http://www.student.cs.uwaterloo.ca/~cs341` for general instructions and policies.
Note: All logarithms are base 2 (i.e., $\log x$ is de ned as $\log_2 x$).

Problems. To be handed in.

1. [12 marks] **Order notation.** For each of the following pairs of functions $f(n)$ and $g(n)$ de ned on positive integers, determine the \most appropriate" symbol in the set $\{O, o, \quad\}$ to ll in the blank in the statement $f(n) \in$ ____$(g(n))$, if one of these symbols applies. (It is possible that none of the choices applies.) \Most appropriate" means that you should not answer \O" if you could answer \o" or \ ". *Justify your answers.*

   You may use the following facts (based on Skiena, p. 56), where $f(n) \ll g(n)$ is shorthand for $f(n) \in o(g(n))$:

   $$1 \ll \log\log n \ll \log n \ll \log^2 n \ll \sqrt{n} \ll n \ll n\log n \ll n^2 \ll 2^n \ll n!$$

   Furthermore, $n^a \in o(n^b)$ for $0 < a < b$, and $\log^a n \in o(n^b)$ for any $b > 0$, and $n^a \in o(2^n)$.

   (a) $f(n) = 3.14159n^3 + 4000n$, $g(n) = .0001n^3 - 2^{2^{10}}n^2$;
   Solution
   Looking at $f(n)$

   $$3.14159n^3 + 4000n \leq 3.14159n^3 + 4000n^3 \leq 5000n^3, \forall n \geq 0$$

   This implies that $f(n) \in O(n^3)$

   $$3.14159n^3 + 4000n \geq 3n^3, \forall n \geq 0$$

   This implies that $f(n) \in \quad (n^3)$
   Since $f(n) \in O(n^3)$ and $f(n) \in \quad (n^3)$, $f(n) \in \quad (n^3)$
   Now looking at $g(n)$
   The same can be done, but we know that if we ignore the constant terms and the lower order values we get that $g(n) \in \quad (n^3)$
   Thus we can conclude that $f(n) \in \quad (g(n))$

   (b) $f(n) = 3^n$, $g(n) = \sum_{i=0}^{n} 2^i$;
   Solution
   Looking at $g(n)$

   $$g(n) = 2^n + 2^{n-1} + 2^{n-2} + \ldots$$

where each exponential not 2^n are simply lower order values that when going to ∞, are negligible

Thus, $g(n) \in \quad (2^n)$

Then by solving the following limit:

$$lim_{n\to\infty} \frac{3^n}{2^n} = (\frac{3}{2})^n = \infty$$

This means that $f(n) \in \omega(g(n))$, however ω is not an option

Thus none of the symbols in the set $\{O, o, \quad\}$ satisfy the condition described above (as would be the correct answer)

(c) $f(n) = \sqrt{\log n}$, $g(n) = \log(\sqrt{n})$;

Solution

Since all logs are base 2, we have that $\log_2 1 = 0$ and $\log_2 2 = 1$

For $n = 1$, $\sqrt{\log 1} == \log 1$

However for $n > 1$ we have that

$\sqrt{\log n} \ll log(n) \ll 2log(n) \ll log(n^2)$

Therefore $f(n) \in o(g(n))$

(d) $f(n) = (\frac{n}{2} - \lfloor\frac{n}{2}\rfloor)n^2$, $g(n) = n^2$;

Solution

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{(\frac{n}{2} - \lfloor\frac{n}{2}\rfloor)n^2}{n^2} = \lim_{n\to\infty} (\frac{n}{2} - \lfloor\frac{n}{2}\rfloor)$$

This limit does not exist thus, there is not a suitable symbol from the set $\{O, o, \quad\}$

2. [10 marks] **Analyzing Algorithms.** Suppose you are given numbers $a_0, a_1, \ldots, a_n$ and you want to evaluate the polynomial

$$P(x) = \sum_{i=0}^{n} a_i x^i$$

at a given value of x. Here is one way to do this:

Algorithm 1: Algorithm for Polynomial Evaluation

```
t=0;
for i = n...0 do
  t = a_i + x·t
```

(a) [6 marks] Prove by induction that this is correct.

Solution

Let p_k be the value of t when the degree of the highest polynomial is k

Let $P_k(x)$ be the value of $P(x)$ when $n = k$

Base Cases

When k = 0, we have that

$$P_0(x) = \sum_{i=0}^{0} a_i x^i = a_0$$

Tracing the value of t when k = 0 gives that

$$t = 0 \rightarrow t = a_0 \Rightarrow p_0 = P_0(x) = a_0$$

When k = 1

$$P_1(x) = \sum_{i=0}^{1} a_i x^i = a_0 + a_1 x$$

$$t = 0 \rightarrow t = a_1 \rightarrow t = a_0 + x(a_1)$$

$$P_1(x) = p_1$$

When k = 2

$$P_2(x) = \sum_{i=0}^{2} a_i x^i = a_0 + a_1 x + a_2 x^2$$

$$t = 0 \rightarrow t = a_2 \rightarrow t = a_1 + x(a_2) \rightarrow t = a_0 + x(a_1 + x(a_2))$$

$$P_2(x) = p_2$$

Inductive Hypothesis

Suppose that this equality holds true for some $0 \leq i \leq 2$ that is $P_i(x) = p_i$

First o , we know that $P_i(x) = P_{i-1} + a_i x^i$ and $p_i = p_{i-1} + a_i x^i$ for $0 \leq i \leq 2$

Inductive Step

We wish to prove that this also holds for p_{i+1}

If we iterate the function twice, we get a t value of $t = a_{i+1}$

The third step would yield $t = a_i + x(a_{i+1})$

However, this is simply the second step of p_i if $t = a_{i+1}$ instead of 0

That is we can also write p_i as $p_i = p_{i-1} + a_i x^i + 0x^{i+1}$

If we replace 0 with a_{i+1} we see that $p_{i+1} = p_i + a_{i+1}x^{i+1}$

Thus we have show that for $i + 1$, $P_{i+1}(x) = p_{i+1}$ and that Algorithm 1 is correct.

(b) [1 mark] How many multiplications does the algorithm perform? Express it exactly and then in notation.

Solution

Each loop of the algorithm performs exactly one multiplication $x \cdot t$

The loop runs from $1 \dots n$, n times, thus the number of multiplications would be

$$(n(1)) = \quad (n)$$

(c) [3 marks] A really straight-forward method of evaluating the polynomial is to compute each term $a_i x^i$ completely separately and then add them all up. Write this as pseudo-code. How many multiplications does this algorithm perform? Express it exactly and then in notation.

Algorithm 2: 2c) Algorithm for solving polynomial

sum=0;
for $i = 0 \dots n$ **do**
  tmp=1;
  for $j = 0 \dots i$ **do**
    tmp = tmp · x;
  $sum = sum + a_i \cdot tmp$

Each iteration of the loop above will perform exacty $i + 1$ multiplications. One for the multiplication between a_i and x^i and the remaining i to compute x^i

This means the multiplications in is:

$$(\sum_{i=1}^{n+1} i) = \quad (\frac{(n+1)(n+2)}{2}) = \quad (n^2)$$

3. [10 marks] **Reductions.** For a set S of n numbers, the element of *rank* k, for $k = 1..n$ is the element that would be at index k if the elements were placed in an array $A[1..n]$ sorted in non-decreasing order. The *median* of S is the element of rank $\lfloor \frac{n+1}{2} \rfloor$. For example: if $S = \{4, 6, 1, 7, 2\}$, then the median is 4 and the element of rank 5 is 7; and if $S = \{9, 7, 5, 2\}$, then the median is 5.

   There is a linear-time median nding algorithm, which we will call **Median**. Given a set S of n numbers, **Median**(S) nds the median value in worst case time $O(n)$. (In CS 240, you might have seen Quickselect, which is a randomized algorithm that runs in linear *expected* time, but quadratic worst case time.)

   Now suppose you want to solve the general selection problem, i.e., to write a program **Select**(S, k) that returns the element of rank k in S. You may assume that the numbers in S are natural numbers in the range $[1..M]$ and are all distinct. Observe that **Select**($S, \lfloor \frac{n+1}{2} \rfloor$) will return the median, so **Select** is more general.

   Give a linear time algorithm for **Select** by reducing the problem to **Median**. Recall that reductions were described in class; your algorithm should be really short, and it should make a call to **Median**. Argue that your algorithm is correct. Analyze its worst case run time. You will need to use the fact that **Median** runs in time $O(n)$ when the input set has size n.

Solution

ASKED TA FOR CLARIFICATION DURING OFFICE HOURS

Algorithm 3: Select Algorithm

```
Function Select(S, k) is
    r = ⌊(len(S)+1)/2⌋;
    m = Median(S);
    d = |k − r|;
    T = copy(S);
    max_v = max(S);
    for i = 1...d do
        if k > r then
            T.append(max_v);
        end
        else if k < r then
            T.append(0);
        end
    end
    return Median(T)
end
```

Correctness Proof

First we assume that $max(S)$ obtains the maximum value of the array S

The rst thing to note is that d is the distance that the element with rank k is from the median if the array were sorted

From the question statement, we know that all elements of the S are unique, so we don't need to worry about returning a duplicate

Consider the following scenario when adding elements to S:

1. Adding a 0 ($k < r$)

When adding a 0 to S, since all numbers are in range [1..M], adding a zero would not only make the array longer, but the median value is now at rank $r - 1$ (in relation to the original S) That is when we compute $Median(T)$ we would get the element at $r - 1$ if 1 zero is added, and the element of $r - l$ if l zeros are added.

Thus if $k < r$, adding $|k - r|$ zeros will return the correct value at that rank

2. Adding max_V ($k > r$)

When adding max_V to S, it is equivalent to adding the largest value of S to the end of the array (assuming it is sorted)

This results in the call to $Median(T)$ having to return the element at r + 1 (in relation to S), which is equivalent to calling $Select(S, \lfloor \frac{n+1}{2} \rfloor + 1)$

Thus for some call to $Select(S, k = \lfloor \frac{n+1}{2} \rfloor + i)$, by adding i instances to array S, we would get the element at k

3. If $k = r$

This would add nothing ($T = S$) and return the median

We have now covered all cases and this proves that Algorithm 3 should indeed return the element at rank k

Time Analysis

Let n be the length of S

Obtaining the max value of an array can be done in a linear scan which takes $O(n)$ time

The calculation of the median takes $O(n)$ in the worst case

Copy an array of size n takes $O(n)$ time

The for loop from $1 \ldots d$ will happen at most $n - r$ times or $\frac{n}{2}$ times

Appending into the end of an array takes $O(1)$ time

Thus the for loop will run at worst $O(\frac{n}{2})$

Finding the median of T will take $O(n + k - r)$, where $k - r \leq \frac{n}{2}$

Thus the worst case on the last Median calculation will take $O(\frac{3n}{2})$

The total runtime would then be

$$O(n + n + n + \frac{n}{2} + \frac{3n}{2}) = O(6n) = O(n)$$