Lecture 0

Thursday, 24 August 2023 3:52 PM

Problem 1: Given a +ve integer x as input, determine if x is prime or not.

Algorithm:

```
IsPrime (int x) :
    y ← ⌊√x⌋
    flag ← 1          \\ 1 if prime, else 0
    for i = 2 ... y
        if (x % i) == 0
            flag ← 0
        end
    end
    return flag
```

Proof of Correctness:

Claim: IsPrime is correct ✗

Claim: IsPrime returns 1 if x is prime, 0 o.w. ✗

Claim: Flag = 1 iff x is prime

Proof: ($\Leftarrow$) Suppose x is prime. Then only divisors are 1 & x. I.e., no divisor in $\{2, \ldots, \lfloor \sqrt{x} \rfloor\}$. Hence flag = 1

($\Rightarrow$) We prove the contrapositive. Suppose x is composite. Let q be smallest prime factor. Then $x = qr$, for $r \geq q$, & hence $q \leq \lfloor \sqrt{x} \rfloor$. When $i = q$ in for loop, it must set flag to 1.

Runtime:

- what is length of input?
- is the algorithm a polynomial time algorithm?

Typically, when we talk about runtime, we ignore a $\log(|I|)$ factor, since this is always present.

Problem 2: Given an array $S = \{a_1, \ldots, a_n\}$ of n +ve integers, sort them.

... using merge sort.

Example:

6 24 19 4 12 | 13 18 1 10 7

4 6 12 19 24 | 1 7 10 13 18

merge ↘ ↙

1 4 6 7 10 12 13 18 19 24

```
Merge Sort (array S)
    n ← length(S)
    if (n ≤ 100)
        sort & return S
    S1 ← S(1 ... ⌊n/2⌋), S2 ← S(⌊n/2⌋+1 ... n)
    SS1 ← Merge Sort (S1), SS2 ← Merge Sort (S2)
    n1 ← length(SS1)    n2 ← length(SS2)
    initialize (T, n)
    p1 ← 1, p2 ← 1, pT ← 1
    while (pT ≤ n)
        if (SS1(p1) < SS2(p2))
            T(pT) ← SS1(p1), p1++, pT++
        else
            T(pT) ← SS2(p2), p2++, pT++
    return T
```

- What is run time?

  What if we initially had a statement:

  if (S is sorted)
  return S

  ?

- We are interested in the running time as a function of n,
  - for large n
  - for worst-case inputs.
    (not best-case, or average-case, or normal-case ...)

Proof of Correctness:

Claim 1: Array S returned by Merge Sort (S) is sorted in non decreasing order.

Proof: By induction on size of S.

Base case: $n \leq 100$. Clearly the claim holds.

Inductive step: given $n > 100$, suppose algo works correctly for inputs of size $< n$.

SS_1, SS_2 are sorted halves of S.

Claim 2: At the end of each iteration of the while loop, $T(1 \ldots pT-1)$ consists of elts. from $SS_1(1 \ldots p_1 - 1) \cup SS_2(1 \ldots p_2 - 1)$ in sorted order

Proof: By induction on pT

Base case: when $pT = p_1 = p_2 = 1$. Clearly this holds

Inductive step: Suppose true for smaller pT.

Let pT', p_1', p_2' be values of pT, p_1, p_2 at end of previous while loop, note $pT' = pT - 1$.

By induction, $T(1 \ldots pT-2)$ consists of elts. from $SS_1(1 \ldots p_1' - 1) \cup SS_2(1 \ldots p_2' - 1)$ in sorted order

If $SS_1(p_1' - 1) < SS_2(p_2' - 1)$,

then $SS_1(p_1' - 1)$ is the next elt. added to array T, and this is the largest elt in T so far.

Then $p_1 = p_1' + 1$, $p_2 = p_2'$, $pT = pT' + 1$, and $T(1 \ldots pT-1)$ consists of the elts. from $SS_1(1 \ldots p_1 - 1) \cup SS_2(1 \ldots p_2 - 1)$ in sorted order.

The case if $SS_1(p_1' - 1) \geq SS_2(p_2' - 1)$ is similar

This proves the claim ∎

To prove Claim 1, note that the loop terminates when $pT = n+1$. In this case by Claim 2, $T(1 \ldots n)$ consists of the elts. from $SS_1(1 \ldots n_1) \cup SS_2(1 \ldots n_2)$ in sorted order, as required ∎