Exercise Set #1

CMPUT 204, Fall 2019
Department of Computing Science
University of Alberta

Problem 1. Exercise 2.1-3 from the book (page 22).

Problem 2. Consider the following recursive version of `InsertionSort`:

procedure InsertionSort(A, n)
  `**Sorts array` A `of size` n
  if $n > 1$ **then**
    InsertionSort($A, n-1$)
    $x \leftarrow A[n]$
    PutInPlace($A, n-1, x$)
  end if

procedure PutInPlace(A, j, x)
  if ($j = 0$) **then**
    $A[1] \leftarrow x$
  else if $x > A[j]$ **then**
    $A[j+1] \leftarrow x$
  else $**$ i.e., $x \leq A[j]$
    $A[j+1] \leftarrow A[j]$
    PutInPlace($A, j-1, x$)
  end if

Input: An array A s.t. $A[1, \cdots, j]$ is sorted and $|A| \geq j+1$
- an index x

output: A st. $A[1, \cdots, j+1]$ is sorted.
2. $A[1, \cdots, j+1]$ contains $A[1, \cdots, j]$ and contains x.

a) First, prove (using induction) the correctness of `PutInPlace` by showing that:

> For any array A, and natural number j such that (i) A has (at least) $j+1$ cells, and (ii) the subarray $A[1, ..., j]$ is sorted, when `PutInPlace(`A, j, x`)` terminates, the first $j+1$ cells of A contain all the elements that were originally in $A[1, .., j]$ plus x in sorted order.

b) Use induction and the correctness of `PutInPlace()` (proved in part a) to prove correctness of `InsertionSort(`A, n`)`.

Problem 3.

Prove the correctness of the following program that is supposed to return the largest number in a given array A.

<u>`procedure FindMax(`A, n`)`</u> `**Returns the largest element in` $A[1..n]$
$index \leftarrow 1$
`for (`j `from 2 to` n`)`
  `if (`$A[j] > A[index]$`)`
    $index \leftarrow j$
`return` $A[index]$

Problem 4. Suppose you are given a set of small boxes, numbered 1 to n, identical in every aspect except that each of the first i contains a pearl whereas the remaining $n - i$ are empty. You can have two magic wands that can each test if a box is empty or not in a single touch, except that a wand disappears if you test it on a box that is empty. Show that, without knowing the value of i, you can use the two wands to determine all the boxes containing pearls using at most no more than $2\sqrt{n}$ wand touches.

Problem 5. Prove the following pseudocode is correct in that it returns, $\max_{i \in 1,...,n-1} |A[i] - A[i+1]|$:
DIFF (A)

** Given array A of $n \geq 2$ integer, finds the largest difference of two consecutive elements of array
$d = |A[1] - A[2]|$
for $i = 2$ **to** $n - 1$ **do**
  if $|A[i] - A[i+1]| > d$ **then**
    $d = |A[i] - A[i+1]|$
  end if
end for
return d

Problem 6. True or False (no proof needed)?

a. $n^{1.001} + n \log n \in \Theta(n^{1.001})$. True

b. $4^{\sqrt{\log n}} = O(\sqrt{n})$. True

True **c.** $2n^2 2^n \in o(2^{1.5n})$.

d. $(\log n)^n \in \Theta(n^{\log n})$. False

e. $n^2 \log^3 n + 10\sqrt{n} \in \Omega(n^{\frac{5}{4}})$. True

f. $2^{3n} \in \Theta(6^n)$. False

g. $(\log \log n)^{\log n} \in \Omega(2.5^n)$. False

h. $2^{\sqrt{\log n} \log n} \in \Omega(n \log n)$.

i. $n^{\frac{\log \log n}{\log n}} \in O(n^{0.01})$.

j. $n! \in O(2^{n^2})$.

b. $f(n) = O(g(n))$ if $g(n)$ grows at least as fast as $f(n)$ for large n.

$\sqrt{n} = 2^{\log \sqrt{n}} = 2^{\frac{1}{2}\log n}$

$4^{\sqrt{\log n}} = 2^{2\sqrt{\log n}}$

since $2\sqrt{\log n} \in O(\frac{1}{2}\log n)$

$4^{\sqrt{\log n}} \in O(\sqrt{n})$

h. $2^{\sqrt{\log n}\log n} \in \Omega(n\log n)$

$\log(2^{\sqrt{\log n}\log n}) = \sqrt{\log n}\log n$ → greater

$\log(n\log n) = \log n + \log\log n$

j. $n! \in 2^{n^2}$

$\log(n!) \leq \log(n^n) = n\log n$

and $n\log n \in O(n^2)$, T

Problem 7. Order the following list of functions by increasing big-Oh growth rate. Group together (for example, by underlining) those functions that are big-Theta of one another.

$3^{\sqrt{n}}$	$2^{100+1/n}$	$\log n / \log\log n$	$\log^5 n$	$e^{\ln n}$
2^{2^n}	4^n	n^{100}	$1/(n \log n)$	$4n^{3/2}$
$3^{\sqrt{\log n}}$	$5(n + 1/n)$	$n \log^3 n$	$2^{n^2 \log n}$	$\log(n!)$
$n!$	n^3	$n^2 \log n$	$4^{\log n}$	$\sqrt{\log \log n}$

n

$\frac{1}{n\log n} > 2^{100+1/n}$

$\log(\frac{1}{n\log n}) = \log(1) - \log(n\log n)$
$= \log(\frac{1}{n}(\log n)^{-1}) = \log\frac{1}{n} + \log(\frac{1}{\log n})$

Problem 8. Selection Sort is another sorting algorithm that you might have seen. It first finds the smallest element and placing it in the first place, then finds the second smallest element and puts it in the second place and so on.

SelectionSort (A)
** Given array A of n integers, sorts the array
for $i = 1$ **to** $n - 1$ **do**
  $k = i$
  for $j = i + 1$ **to** n **do**
    if $A[j] < A[k]$ **then**
      $k = j$
    end if
  end for
  swap $A[i]$ and $A[k]$
end for

a. Prove the correctness of Selection Sort using loop invariant.

b. Find the worst-case running time of the algorithm and express it in $\Theta(.)$ form.

c. Find the best-case running time of the algorithm and express it in $\Theta(.)$ form.

d. What is the worst-case number of "Swap" operations? express this in $\Theta(.)$ form.

1. procedure linearSearch(A, V)

```
for i = 1 to i = A.length do
   if V = A[i] then
       return i
   else
       i <- i+1
return NIL
```

At the beginning of each loop iteration, the subarray A[1... i-1] consists the elements that are not equal to V.

initialization: before the loop begins, the array is empty so that none of its element equals to V.

maintenance: in the i-th iteration, the checking is done so that A[i] is neither equals to V or not.

if V = A[i], the loop terminates, else the iteration will be continuing further.

if V is not found in the subarray A[i... i-1], then before the next iteration it will not contain the value V.

Termination:

- The index of the element is being searched is found.
- reached at the end of the array and not found the index of the element is being searched.

P.2.

base case $(j=0)$
putInPlace$(A, 0, x)$

$A[1]=x$ $\boxed{x}$

IH: let $j \in \mathbb{Z}_{\geq 0}$

suppose putInPlace(A, j, x) is correct

want: putInPlace$(A, j+1, x)$ is correct

Case: $x > A[j]$

Case 2: $x \leq A[j]$

1 2 4 5 3

1 2 4 5 5

Problem 3.

Prove the correctness of the following program that is supposed to return the largest number in a given array A.

```
procedure FindMax(A, n) **Returns the largest element in A[1..n]
index ← 1
for (j from 2 to n)
    if (A[j] > A[index])
        index ← j
return A[index]
```

---

loop invariant: at the beginning of each loop iteration,
$A_{max} = \{A[1], A[2], \cdots, A[j-1]\}$.

initialization: before the loop begins, A[1] is the largest number in array.

maintenance: suppose the beginning of iteration j (j-1)th term holds,

(1) $A[j] > A[index]$, from loop invariant we get $A[j]$ is larger than the maximum of the numbers in $A[1 \cdots j]$, thus, $A[j]$ is the maximum of $A[1 \cdots j+1]$

case 2. $A[j] < A[index]$: in this case, $A[0 \cdots j]$ is smaller than. $A[j]$, thus the maximum $A[j+1]$ is the same as $A[j]$. The algorithm also does not change answer..

Termination: when the for loop terminates $j = (n-1)+1 = n$

Problem 5. Prove the following pseudocode is correct in that it returns, $\max_{i \in 1,\ldots,n-1} |A[i] - A[i+1]|$:
DIFF (A)

** Given array A of $n \geq 2$ integer, finds the largest difference of two consecutive elements of array

```
d = |A[1] − A[2]|
for i = 2 to n − 1 do
    if |A[i] − A[i + 1]| > d then
        d = |A[i] − A[i + 1]|
    end if
end for
return d
```

loop invariant:

a. $n^{1.001} + n \log n \in \Theta(n^{1.001})$

b. $4^{\sqrt{\log n}} = O(\sqrt{n})$. True

c. $2n^2 2^n \in o(2^{1.5n})$.

d. $(\log n)^n \in \Theta(n^{\log n})$. False

e. $n^2 \log^3 n + 10\sqrt{n} \in \Omega(n^{\frac{5}{4}})$.

f. $2^{3n} \in \Theta(6^n)$. False

g. $(\log \log n)^{\log n} \in \Omega(2.5^n)$. Fal

h. $2^{\sqrt{\log n} \log n} \in \Omega(n \log n)$.

i. $n^{\frac{\log \log n}{\log n}} \in O(n^{0.01})$.

j. $n! \in O(2^{n^2})$.

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

$\approx \log n \log 2^{\sqrt{\log n}}$

$\approx \log n \sqrt{\log n}$

$\log(n \log n) = \log n + \log \log n$

a - True

b. $4^{\sqrt{\log n}} = 2^{2\sqrt{\log n}}$ -

$\sqrt{n} = 2^{\log \sqrt{n}} = 2^{\log n^{\frac{1}{2}}} = 2^{\frac{1}{2}\log n}$

C. True

d. false

e. True

f. false

g. $\log(\log\log n)^{\log n}$

$= \log n \log(\log\log n)$