COMP 250 – Midterm
October 17th 2014, 18:10 – 19:55

- This exam has 7 questions.
- This is an open book and open notes exam. No electronic equipment is allowed.

Question 1 (15 points). Java programming

What will the following Java program print when executed?

class question1 {
    static public void questionA(int x) {
        x = x + 2;
    }

    static public int questionB(int x) {
        x = x + 3;
        return x;
    }

    static public void questionC(int array[]) {
        array[0] = array[0] + 4;
    }

    static public int questionD(int n) {
        if (n<=1) return 1;
        return questionD(n-1)+questionD(n-2);
    }

    public static void main(String args[]) {
        int x, y, z;
        int a[] = new int[10];
        x = 1;
        y = 1;
        a[0] = 1;
        questionA(x);
        y = questionB(y);
        questionC(a);
        z = questionD(6);
        System.out.println("x = " + x);
        System.out.println("y = " + y);
        System.out.println("a[0] = " + a[0]);
        System.out.println("z = " + z);
    }
}

Answer:

x = 1
y = 4
a[0] = 5
z = 13
Question 2 (20 points). Stacks and recursion

Professor Stackbottom proposes the following recursive algorithm that is using a stack as argument.

```
Algorithm mistery(Stack S)
Input: Stack S
Output: Modifies the stack S and returns a number

value = S.pop()
if (S is empty) then return value
else {
    result = mistery(S)
    S.push(value)
    return result
}
```

The objective of this question is to discover the purpose of this algorithm. We start by executing the following commands.

```
S = new Stack();
S.push('1');
S.push('2');
S.push('3');
```

a) (4 points) Draw the content of the stack at this point.

```
Stack S:
3
2
1
```

b) (8 points) If we now execute

```
int x = mistery(S);
```

What is the value of x, and what is the content of the stack after the execution of the algorithm?

```
x = 1
Stack S:
3
2
```

c) (4 points) In one sentence, explain what is this algorithm doing when given a stack S as input.

```
It removes the object at the bottom of the stack and returns it.
```

d) (4 points) Using the big-Oh notation, give the running time of the mistery algorithm if it is executed on a stack of n elements. No justification is needed.

```
O(n)
```
Question 3 (15 points). Proofs by induction

Prove by induction on \( n \) that for every integer \( n \geq 0 \) and any real number \( a > 0 \), we have

\[
a^0 + a^1 + a^2 + \ldots + a^n = \frac{a^{n+1} - 1}{a - 1}.
\]

Base case: WE’VE DONE THIS EXAMPLE IN CLASS

Induction hypothesis:

Inductive step:
Question 4 (15 points). Recursive algorithms

Complete the pseudocode of the RecursiveSum algorithm below to obtain a recursive algorithm such that given a positive integer $n$, it prints all the ways of expressing $n$ as sums of positive integers. For example, given $n=4$, the output should looks like this:

1+1+1+1=4
1+1+2=4
1+2+1=4
1+3=4
2+1+1=4
2+2=4
3+1=4
4=4

Note: This will be easier to do if we add, in addition to $n$ itself, two additional arguments to the RecursiveSum algorithm:
- an array $A$ large enough to store up to $n$ elements, which will be used to accumulate partial sums through recursive calls.
- an integer $soFar$ that keeps track of how many elements of $A$ have been filled already.

Then, the result shown above would be obtained by calling $\text{RecursiveSum}(A[ ], 0, 4)$.

**Algorithm** $\text{RecursiveSum}(A[ ], soFar, n)$

**Inputs:** $A[ ]$ is an array of integers, where elements $A[0,..., soFar-1]$ are already filled
$n$ is an integer

**Output:** The algorithm prints out every possible ways to complete the partial sum already stored in $A[0,...,soFar-1]$ so that the numbers add up to $n$.

\[
\]

```plaintext
if ( sumSoFar = n ) then print A[0] "+" A[1] "+" ... "+" A[soFar-1] "=" n
else { /* WRITE YOUR PSEUDOCODE HERE */
    for i = 1 to n - sumSoFar do
        A[soFar] = i
        RecursiveSum(A, soFar+1, n)
}
```
Question 5 (10 points). Big-Oh notation

Prove, using only the definition of the big-Oh notation, that \( \log(n^2 + 1) + n + 1 \) is \( O(n) \).

To prove this, we need to find constants \( c \) and \( n_0 \) such that \( \log(n^2+1) + n + 1 \leq c \cdot n \) for all \( n \geq n_0 \).

We note that:
\[
\log(n^2 + 1) + n + 1 \leq \log(n^2 + n^2) + n + 1 \quad \text{(if } n \geq 1) \\
= \log(2n^2) + n + 1 \\
= \log(2) + 2 \log(n) + n + 1 \\
= 2 \log(n) + n + 2 \\
\leq 2n + n + 2n \quad \text{(if } n \geq 1) \\
= 5n
\]

So, if we choose \( n_0 = 1 \) and \( c = 5 \), we get that \( \log(n^2 + 1) + n + 1 \leq c \cdot n \) for all \( n \geq n_0 \). Thus, \( \log(n^2+1) + n + 1 \) is \( O(n) \).
Question 6 (10 points). Solving recurrences

Using the substitution method, obtain an explicit formula for the following recurrence:

\[ T(n) = T(n-1) + 2n + 1 \quad \text{if } n > 0 \]
\[ 0 \quad \text{if } n = 0 \]

Let's first obtain the first few values of \( T(n) \), for verification purposes.

\[ T(0) = 0; \quad T(1) = 0 + 2*1 + 1 = 3; \quad T(2) = 3 + 2*2+1 = 8; \quad T(3) = 8 + 2*3 + 1 = 15; \quad T(4) = 15 + 2*4 +1 = 24 \]

Now, we use the substitution method to obtain an explicit formula for \( T(n) \).

\[ T(n) = T(n-1) + 2n + 1 \]
\[ = (T(n-2) + 2(n-1)+1) + 2n + 1 = T(n-2) + 4n + 2 - 2 \] \( (2) \)
\[ = (T(n-3) + 2(n-2) + 1) + 4n + 2 - 2= T(n-3) + 6n + 3 -2-4 \] \( (3) \)
\[ = (T(n-4) + 2(n-3) + 1) + 6n +3 -2-4= T(n-4) + 8n + 4 -2-4-6 \] \( (4) \)
\[ \ldots \]
\[ = T(n-k) + 2k n + k - 2 \sum_{i=0}^{k} i \]

We hit the base case when \( n-k = 0 \), i.e. \( k=n \). We then get

\[ T(n) = T(0) + 2n^2 + n - 2 \sum_{i=0}^{n-1} i \]
\[ = 0 + 2n^2 + n - 2(n-1)*n/2 = 2n^2 + n - n^2 + n = n^2 + 2n \]

Verification: From the explicit formula, we get \( T(0) = 0, T(1) = 1+2 = 3, T(2) = 4+4 = 9, T(3) = 9 + 6 = 15, T(4) = 16 + 8 = 24 \). So all looks good.
**Question 7 (15 points). Running time of algorithms**

Give the worst-case running time of the following algorithms, using the simplest $\Theta()$ notation (big-Theta notation) possible. No justification needed.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$\Theta()$ Running time</th>
</tr>
</thead>
</table>
| **Algorithm1** $(\text{int } n)$ | $i \leftarrow 2 \cdot 2^n$  
while $(i > 1)$ do {
  $i \leftarrow i / 2$
} | $\Theta(n)$ |
| **Algorithm2** $(\text{int } n)$ | $\text{for } i = 1 \text{ to } n \text{ do }$
  $\text{for } j = 1 \text{ to } 999 \text{ do }$
  $\text{print ``Bazinga!''}$ | $\Theta(n)$ |
| **Algorithm3** $(A[], \text{int } n)$ | $\text{for } i = 0 \text{ to } n-1 \text{ do }$
  $A[i] = i$
merge(A, 0, n/2, n-1)
pivot = partition(A, 0, n-1) | $\Theta(n)$ |

**Note:** merge and partition refer to the algorithms seen in class.
This page is left intentionally empty. You can use it for drafting your solutions.