UNIVERSITY of GUELPH

CIS2520 Data Structures

Fall 2011, Answers to Midterm

---

1) 3+2 MARKS

(a)

By hypothesis, there exist two values $m_1 \in \mathbb{Z}_+$ and $\lambda_1 \in \mathbb{R}_+$ such that:

$$\forall n \in m_1..+\infty,\ f_1(n) \leq \lambda_1\, g_1(n)$$

Also, there exist two values $m_2 \in \mathbb{Z}_+$ and $\lambda_2 \in \mathbb{R}_+$ such that:

$$\forall n \in m_2..+\infty,\ f_2(n) \leq \lambda_2\, g_2(n)$$

Therefore: $\forall n \in \max\{m_1,m_2\}..+\infty,\ f_1(n) \leq \lambda_1\, g_1(n) \ \wedge\ f_2(n) \leq \lambda_2\, g_2(n)$

Which implies: $\forall n \in \max\{m_1,m_2\}..+\infty,\ f_1(n)f_2(n) \leq (\lambda_1 g_1(n))(\lambda_2 g_2(n))$

In other words: $\forall n \in \max\{m_1,m_2\}..+\infty,\ (f_1 f_2)(n) \leq (\lambda_1\lambda_2)\ (g_1 g_2)(n)$

We have found two values $m=\max\{m_1,m_2\} \in \mathbb{Z}_+$ and $\lambda=\lambda_1\lambda_2 \in \mathbb{R}_+$ such that:

$$\forall n \in m..+\infty,\ (f_1 f_2)(n) \leq \lambda\ (g_1 g_2)(n)$$

This means that $f_1 f_2$ is $O(g_1 g_2)$.

(b)

Consider the functions defined by $f_1(n)=g_1(n)=g_2(n)=n^2$ and $f_2(n)=n$.

We have $(f_1/f_2)(n)=f_1(n)/f_2(n)=n$ and $(g_1/g_2)(n)=g_1(n)/g_2(n)=1$.

f_1 is $O(g_1)$ and f_2 is $O(g_2)$, but f_1/f_2 is not $O(g_1/g_2)$.

2) 2+3 MARKS

(a)

Let n be a positive integer. The function f is defined at n iff
$2n^4+9n^3\sqrt{n}-17$ belongs to the codomain $\mathbb{R}_+$, i.e., iff $2n^4+9n^3\sqrt{n}-17>0$.

If $n \geq 2$, then $n^4 \geq 16$ and $2n^4 \geq 32$. Moreover, $9n^3\sqrt{n} \geq 0$.

Therefore, if $n \geq 2$, then $2n^4+9n^3\sqrt{n}-17 \geq 32+0-17 = 15 > 0$.

In other words, f is defined on the neighbourhood of infinity $2..+\infty$.

(b)

Let n be an element of 2..+∞. The function f is then defined at n and:
$2n^4+9n^3\sqrt{n}-17 \le 2n^4+9n^3\sqrt{n} \le 2n^4+9n^3n \le 2n^4+9n^4 \le 11n^4$
We have found two values $m=2\in\mathbb{Z}_+$ and $\lambda=11\in\mathbb{R}_+$ such that:

$$\forall n\in m..+\infty,\ f(n) \le \lambda n^4$$

This means that f(n) is $O(n^4)$.

3) 2+0.5+0.5 MARKS

(a) Line 1: 2n+5 primitive operations
Line 2: $n(2n+5) = 2n^2+5n$
Line 3: $8n^2$
Line 4: 1
TOTAL = $10n^2+7n+6$
(b) $(10n^2+7n+6)$ *tmax*
(c) $O(n^2)$

4) 4+0.5+0.5 MARKS

(a) Line 1: $2(n-1)+5 = 2n+3$ primitive operations

Line 2: $[2(n-1)+5]+[2(n-2)+5]+\ldots+[2\times1+5]$
$= \sum_{k=1}^{n-1}[2k+5] = 2\sum_{k=1}^{n-1}k + 5(n-1) = n(n-1)+5(n-1) = n^2+4n-5$

Line 3: $[(n-1)+(n-2)+\ldots+1]\times5 = 5\sum_{k=1}^{n-1}k = 5n(n-1)/2$

TOTAL = $(7n^2+7n-4)/2$

(b) $(7n^2+7n-4)$ *tmax* / 2

(c) $O(n^2)$

5) 3 MARKS

```
function binarySearch (A, v, first, last)
      if first>last return –1
      middle=(first+last)/2
      if A[middle]=v return middle
      elseif A[middle]>v return binarySearch(A,v,first,middle–1)
      else return binarySearch(A,v,middle+1,last)
```

6) 3+1 MARKS

(a)

The concrete data structure definition for students is probably:

```
typedef struct {
    char *name;
    int grade;
} Student;
```

`InitializeStudent("John",75,&S);`
allocates memory in the heap for the string “John”.
This memory is freed by the first
`FreeStudent(&S);`
and then reallocated for “Mary” by
`InitializeStudent("Mary",95,&S);`
“John” is therefore overwritten with “Mary”.

The problem comes from the fact that `Insert` does not make a deep copy of the `Student` item passed to it. It makes a shallow copy: `L->items[position]=X` copies the pointer to the memory allocated for the student’s name, but does not allocate new memory in the heap for a copy of the student’s name. `Insert` should therefore be modified as follows:

(b)

```
void Insert (Student X, int position, List *L) {
      int i;
      for (i=L->size; i>position; i--)
           L->items[i]=L->items[i-1];
      InitializeStudent(NameOfStudent(X),         // changes are here
                        GradeOfStudent(X),        // changes are here
                        &L->items[position]);     // changes are here
      L->size++;
}
```

7) 2 MARKS

```
void OutputFromLastToFirst (List *L) {
     if(L) {
          OutputFromLastToFirst (L->next);
          printf(“%d\n”,L->item);
     }
}
```

8) 2 MARKS

```
void Dequeue (Queue *Q) {
     Q->size--;
     Q->head=(Q->head+1)%100;
}
```

9) 2+3 MARKS

(a)

```
int Size (Stack *S) {
     if(!S) return 0;
     return 1+Size(S->next);
}
```

(b)

```
Stack *Push (SomeLargeStructure *X, Stack *S) {
      Stack *s;
      s=(Stack *)malloc(sizeof(Stack));
      copySomeLargeStructure(&s->item,X);
      s->next=S;
      return s;
}
```

10) 4 MARKS

```
function ReverseStack (S)
     Q=QCreate()
     while not SEmpty(S)
          QEnqueue(STop(S),Q);
          SPop(S)
     while not QEmpty(Q)
          SPush(QHead(Q),S)
          QDequeue(Q)
```

11) 2+2+2 MARKS

(a)

```
Create: ∅ → PQueue[T]
Insert: T x N x PQueue[T] → PQueue[T]
Remove: PQueue[T] → PQueue[T]
Full: PQueue[T] → Boolean
```

```
Empty: PQueue[T] → Boolean
Size: PQueue[T] → N
Priority: PQueue[T] → N
Item: PQueue[T] → T
```

(b)

```
   ¬Empty(Q)
∧  Size(Q)=Size(old Q)+1
∧  [ Empty(old Q) → (Priority(Q)=P ∧ Item(Q)=I) ]
∧  [ ¬Empty(old Q) → (Priority(Q)=max{P,Priority(old Q)}
                        ∧ (Item(Q)=P ∨ Item(Q)=Item(old Q))) ]
```

(c)

```
Remove(Insert(I,Priority(Q)+1,Q))=Q
```

12) 3 MARKS

(a) `assert()` is defined in the header file <`assert.h`>.
(b) It is used to test (pre/post)conditions: `assert(condition);`
(c) If the (pre/post)condition is false, `assert()` prints a diagnostic message (with the source filename, line number and function) and terminates the program.
(d) If the macro NDEBUG is defined before the inclusion of <`assert.h`>, `assert()` has no effect, and will not even evaluate its argument.
(e) Note that NDEBUG can be defined on the `gcc` command line instead.