# Suggested Solutions to Midterm Problems

1. The following is a recursive variant of Euclid's algorithm (using the simpler - rather than %) for computing the greatest common divisor of two positive integers.

```
int gcd(int a, int b)
{
  if (a == b)
    return b;
  if (a > b)
    return gcd(b, a - b);
  else
    return gcd(a, b - a);
}
```

   (a) (4 points) What are the base cases? Why are they appropriate?

   *Solution.* The base case is when $a = b$ (for $a, b > 0$). It is appropriate, as when $a = b$ the computation may stop with the correct greatest common divisor (which is either of the two equal positive integers). This means that the algorithm terminates correctly if it ever will.

   Further details: when $a > b > 0$, $\gcd(a, b) = \gcd(b, a - b)$; and when $b > a > 0$, $\gcd(a, b) = \gcd(a, b - a)$. Reasoning inductively, the final returned result equals the gcd of the original two input numbers. □

   (b) (6 points) Will the base cases always be reached? Why? What precondition should be stated?

   *Solution.* Yes, the base case will eventually be reached, assuming that both of the original input numbers are positive. This means that the algorithm will terminate.

   When $a > b > 0$, `gcd` is invoked with $b$ and $a - b$ , and when $b > a > 0$, `gcd` is invoked with $a$ and $b - a$. In either case, one of the two numbers and hence their sum become strictly smaller, yet both still greater than 0. As long as the two positive integers are *not* equal, the above recursive invocation will continue and their sum will decrease in each invocation. The finite sum may decrement for only a finite number of times, and eventually the two numbers will become equal.

   Precondition: $a > 0$ and $b > 0$. □

2. The Fibonacci sequence (starting with 1) may be defined by the following recurrence relation:

$$\begin{cases} F_1 = 1 \\ F_2 = 1 \\ F_n = F_{n-2} + F_{n-1} \quad \text{for } n > 2 \end{cases}$$

   Below is a closed-form representation of the same sequence:

$$F_n = \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^n - \frac{1}{\sqrt{5}}(\frac{1-\sqrt{5}}{2})^n, \text{ for } n \geq 1.$$

Prove by *induction* that the closed-form representation is indeed correct.

*Solution.* The proof is by induction on $n$.

Base cases ($n = 1$ or $n = 2$): When $n = 1$, $\frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2}) - \frac{1}{\sqrt{5}}(\frac{1-\sqrt{5}}{2}) = (\frac{1+\sqrt{5}}{2\sqrt{5}}) - (\frac{1-\sqrt{5}}{2\sqrt{5}}) = \frac{\sqrt{5}}{2\sqrt{5}} + \frac{\sqrt{5}}{2\sqrt{5}} = 1 = F_1$, and when $n = 2$, $\frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^2 - \frac{1}{\sqrt{5}}(\frac{1-\sqrt{5}}{2})^2 = \frac{1}{\sqrt{5}}(\frac{1+2\sqrt{5}+5}{4}) - \frac{1}{\sqrt{5}}(\frac{1-2\sqrt{5}+5}{4}) = \frac{2\sqrt{5}}{4\sqrt{5}} + \frac{2\sqrt{5}}{4\sqrt{5}} = 1 = F_2$.

Inductive step ($n > 2$):

$$
\begin{aligned}
& \tfrac{1}{\sqrt{5}}(\tfrac{1+\sqrt{5}}{2})^n - \tfrac{1}{\sqrt{5}}(\tfrac{1-\sqrt{5}}{2})^n \\
= \;& \tfrac{1}{\sqrt{5}}(\tfrac{1+\sqrt{5}}{2})(\tfrac{1+\sqrt{5}}{2})^{n-1} - \tfrac{1}{\sqrt{5}}(\tfrac{1-\sqrt{5}}{2})(\tfrac{1-\sqrt{5}}{2})^{n-1} \\
= \;& [\tfrac{1}{\sqrt{5}}(\tfrac{1+\sqrt{5}}{2})^{n-1} + \tfrac{1}{\sqrt{5}}(\tfrac{-1+\sqrt{5}}{2})(\tfrac{1+\sqrt{5}}{2})^{n-1}] - [\tfrac{1}{\sqrt{5}}(\tfrac{1-\sqrt{5}}{2})^{n-1} + \tfrac{1}{\sqrt{5}}(\tfrac{-1-\sqrt{5}}{2})(\tfrac{1-\sqrt{5}}{2})^{n-1}] \\
= \;& [\tfrac{1}{\sqrt{5}}(\tfrac{1+\sqrt{5}}{2})^{n-1} - \tfrac{1}{\sqrt{5}}(\tfrac{1-\sqrt{5}}{2})^{n-1}] + [\tfrac{1}{\sqrt{5}}(\tfrac{-1+\sqrt{5}}{2})(\tfrac{1+\sqrt{5}}{2})^{n-1} - \tfrac{1}{\sqrt{5}}(\tfrac{-1-\sqrt{5}}{2})(\tfrac{1-\sqrt{5}}{2})^{n-1}] \\
= \;& \{\text{from the induction hypothesis}\} \\
& F_{n-1} + [\tfrac{1}{\sqrt{5}}(\tfrac{-1+\sqrt{5}}{2})(\tfrac{1+\sqrt{5}}{2})(\tfrac{1+\sqrt{5}}{2})^{n-2} - \tfrac{1}{\sqrt{5}}(\tfrac{-1-\sqrt{5}}{2})(\tfrac{1-\sqrt{5}}{2})(\tfrac{1-\sqrt{5}}{2})^{n-2}] \\
= \;& F_{n-1} + [\tfrac{1}{\sqrt{5}}(\tfrac{-1+5}{4})(\tfrac{1+\sqrt{5}}{2})^{n-2} - \tfrac{1}{\sqrt{5}}(\tfrac{-1+5}{4})(\tfrac{1-\sqrt{5}}{2})^{n-2}] \\
= \;& F_{n-1} + [\tfrac{1}{\sqrt{5}}(\tfrac{1+\sqrt{5}}{2})^{n-2} - \tfrac{1}{\sqrt{5}}(\tfrac{1-\sqrt{5}}{2})^{n-2}] \\
= \;& \{\text{from the induction hypothesis}\} \\
& F_{n-1} + F_{n-2} \\
= \;& \{\text{by the recursive definition}\} \\
& F_n
\end{aligned}
$$

□

3. Consider the array-based implementation of the ADT bag, the C++ class `ArrayBag`. We have implemented the function `remove()` with a static array as follows:

```
template<typename ItemType>
bool ArrayBag::remove(const typename& anEntry)
{
  int locatedIndex = getIndexOf(anEntry);
  bool canRemoveItem = (locatedIndex > -1);
  if(canRemoveItem)
  {
    itemCount--;
    items[locatedIndex] = items[itemCount];
  }
  return canRemoveItem;
}
```

Suppose that we now want to implement `ArrayBag` with dynamic memory allocation. The class definition has been changed to

```
template<typename ItemType>
class ArrayBag: public BagInterface
{
private:
```

```
  static const int DEFAULT_CAPACITY = 6;
  ItemType* items;
  int itemCount;
  int maxItems;
  int getIndexOf(const ItemType& target) const;
public:
  // public member functions are omitted
};
```

where `items` is a pointer pointing to the dynamic array. We now want to modify `remove()` and add the following feature: *When the number of items is less than half of the current bag capacity, cut the bag capacity by a half.* Rewrite `remove()` to add this feature.

(Note: you may decide to round up or down when the capacity happens to be odd. The function `getIndexOf()` returns the array index of a given item if it exists or $-1$ otherwise.)

*Solution.* See the slides for TA Session #3 on 11/20. □

4. Convert the infix expression `a * (b - c / d * e) - f` to the postfix form. Please follow the conversion algorithm discussed in class (see the appendix), and show the status of the stack and the current postfix expression after each character of the infix expression is processed. (Note: you may ignore the blank spaces.)

   *Solution.* See the slides for TA Session #3 on 11/20. □

5. Consider again the infix to postfix conversion algorithm, which assumes all operators are associated to the left. Suppose we want to allow the unary `-` (negation), which has higher precedence than `*` and `/` and is associated to the right, i.e., `- - a` equals `- (- a)`. Please modify the conversion algorithm to allow the additional unary `-`. Please use a new symbol, say `~` (tilt), to represent the unary `-` in the postfix expression.

   *Solution.* Below is the modified algorithm that also allows the unary `-`. The algorithm uses `~` to represent the unary `-` internally and in the postfix expression, assuming that the precedence level of `~` is appropriately defined.

```
lastCh = undefined;
for (each character ch in the infix expression) {
  switch (ch) {
    case operand:  // Append operand to the end of postfixExp
      postfixExp = postfixExp + ch
      lastCh = operand;
      break
    case '(':      // Save '(' on the stack aStack
      aStack.push(ch);
      lastCh = '(';
      break
    case operator: // Process operators of higher precedence
      if (ch=='-' and ((lastCh == undefined) or (lastCh == '(') or
                       (lastCh is an operator)) {
        a.Stack.push('~');
        lastCh = operator;
        break
```

```
      }
      while (!aStack.isEmpty() and
             aStack.peek() is not a '(' and
             precedence(ch) <= precedence(aStack.peek())) {
        postfixExp = postfixExp + aStack.peek()
        aStack.pop()
      }
      aStack.push(ch); // Save the operator
      lastCh = operator;
      break
    case ')': // Pop the stack until matching '('
      while (aStack.peek() is not a '(') {
        postfixExp = postfixEP + aStack.peek()
        aStack.pop()
      }
      aStack.pop();  // Remove the matching '('
      lastCh = ‘)’;
      break
  }
}
// Append to postfixExp the operators remaining in the stack
while (!aStack.isEmpty()) {
  postfixExp = postfixExp + aStack.peek()
  aStack.pop()
}
```

□

6. Consider the link-based implementation of a stack. Below are relevant details of the files `StackInterface.h`, `Node.h`, `LinkedStack.h`, and `LinkedStack.cpp`.

```
/** @file StackInterface.h */
#ifndef _STACK_INTERFACE
#define _STACK_INTERFACE
template<class ItemType>
class StackInterface {
{
public:
...

/** @file Node.h */
...
template<class ItemType>
class Node
{
private:
  ItemType item;
  Node<ItemType>* next;
public:
```

```
  Node();
  Node(const ItemType& anItem);
  Node(const ItemType& anItem, Node<ItemType>* nextNodePtr);
  void setItem(const ItemType& anItem);
  void setNext(Node<ItemType>* nextNodePtr);
  ItemType getItem() const ;
  Node<ItemType>* getNext() const ;
};
...

/** ADT stack: Link-based implementation.
 @file LinkedStack.h */
...
#include "StackInterface.h"
#include "Node.h

template<class ItemType>
class LinkedStack: public StackInterface<ItemType>
{
private:
  Node<ItemType>* topPtr;  // Pointer to first node in the chain;
                           // this node contains the stacks top
public:
...
#include "LinkedStack.cpp"
#endif

/** @file LinkedStack.cpp */
#include <cassert>       // For assert
#include "LinkedStack.h" // Header file
...
```

Suppose we now want to add an additional operation called `reverse` that reverses the order of the entries in a stack. Please provide the necessary changes, including implementation, to the files. Try to make your code as efficient as possible.

*Solution.* To accommodate an additional `reverse`, we add a line in the `public` section of `LinkedStack.h` (extending the original template class `StackInterface`):

```
  void reverse();
```

We then provide an implementation for the function in `LinkedStack.cpp` as follows.

```
template<class ItemType>
void LinkedStack<ItemType>::reverse()
{
  if (topPtr != nullptr)
    if (topPtr->getNext() != nullptr) {
      Node<ItemType>* tailPtr = topPtr->getNext();
```

```
      topPtr->setNext(nullptr);
      // topPtr points to top of the reserved part of the stack
      // tailPrt points to start of the part yet to be reversed
      while (tailPtr != nullptr) {
        Node<ItemType>* prevTopPtr = topPtr;
        topPtr = tailPtr;
        tailPtr = tailPtr->getNext();
        topPtr->setNext(prevTopPtr);
      }
    }
}
```

□

7. Apply the merge sort algorithm to the following array. Show the contents of the array after each merge operation.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 3 | 11 | 6 | 12 | 7 | 5 | 4 | 2 | 9 | 1 | 8 |

*Solution.* See the slides for TA Session #3 on 11/20. □

8. Below is a partition procedure that we discussed in class as part of the quick sort algorithm.

```
partition(theArray: ItemArray, first: integer, last: integer): integer
  pivotIndex = last
  pivot = theArray[pivotIndex]

  indexFromLeft = first
  indexFromRight = last - 1
  done = false
  while (not done) {
    while (theArray[indexFromLeft] < pivot)
      indexFromLeft = indexFromLeft + 1

    while (theArray[indexFromRight] > pivot)
      indexFromRight = indexFromRight - 1

    if (indexFromLeft < indexFromFight) {
      Interchange theArray[indexFromLeft] and
                  theArray[indexFromFight]
      indexFromLeft = indexFromLeft + 1
      indexFromRight = indexFromRight - 1
    }
    else
      done = true
  }

  Interchange theArray[pivotIndex] and theArray[indexFromLeft]
```

```
  pivotIndex = indexFromLeft

  return pivotIndex
```

Suppose we now want to use as the pivot the first, instead of the last, entry of the array. More specifically, let us change the line "`pivotIndex = last`" into "`pivotIndex = first`". (The option of interchanging the first and the last entries and then following the same procedure is thus excluded.) What other changes should be made to the rest of the procedure?

*Solution.*

```
partition(theArray: ItemArray, first: integer, last: integer): integer
  pivotIndex = first
  pivot = theArray[pivotIndex]

  indexFromLeft = first + 1
  indexFromRight = last

  ...

  Interchange theArray[pivotIndex] and theArray[indexFromRight]
  pivotIndex = indexFromRight

  return pivotIndex
```

□

9. A sorting algorithm is *stable* if it does not exchange the items that have the same sort key. Among the four sorting algorithms: selection sort, bubble sort, merge sort, and quick sort, which are stable and which are not? Please give a brief explanation for each case. For those that are not stable, propose one single uniform adaptation to turn them into stable ones.

   *Solution.* See the slides for TA Session #3 on 11/20. □

10. Trace the execution of the bank-line simulation discussed in class, with the following data. Show the state of the queue and the event list at each relevant time point (starting from Time 0). Please use the forms (A,$at$,$tl$) and (D,$dt$,−), where $at$ is the arrival time, $tl$ is the transaction length, and $dt$ is the departure time, to represent an arrival event and a departure event respectively. Note that the customer being served is not part of the queue representing the waiting line.

| Arrival time | Transaction length |
|---|---|
| 4 | 8 |
| 9 | 6 |
| 13 | 9 |
| 20 | 5 |
| 26 | 6 |
| 40 | 4 |

*Solution.*

| Time | `bankQueue` | `eventListQueue` |
|---|---|---|
| 0 | | (A,4,8) (A,9,6) (A,13,9) (A,20,5) (A,26,6) (A,40,4) |
| 4 | | (A,9,6) (D,12,−) (A,13,9) (A,20,5) (A,26,6) (A,40,4) |
| 9 | (A,9,6) | (D,12,−) (A,13,9) (A,20,5) (A,26,6) (A,40,4) |
| 12 | | (A,13,9) (D,18,−) (A,20,5) (A,26,6) (A,40,4) |
| 13 | (A,13,9) | (D,18,−) (A,20,5) (A,26,6) (A,40,4) |
| 18 | | (A,20,5) (A,26,6) (D,27,−) (A,40,4) |
| 20 | (A,20,5) | (A,26,6) (D,27,−) (A,40,4) |
| 26 | (A,20,5) (A,26,6) | (D,27,−) (A,40,4) |
| 27 | (A,26,6) | (D,32,−) (A,40,4) |
| 32 | | (D,38,−) (A,40,4) |
| 38 | | (A,40,4) |
| 40 | | (D,44,−) |
| 44 | | |

□