

Final Exam: May 19**Name: Key**

You will have 3 hours for this exam, although you should not need that much. This exam is closed-book and closed-note. Please take some time to check your work. If you need extra space, write on the back. There are a total of 70 points on this exam.

1. (12 points) Consider the following Haskell function:

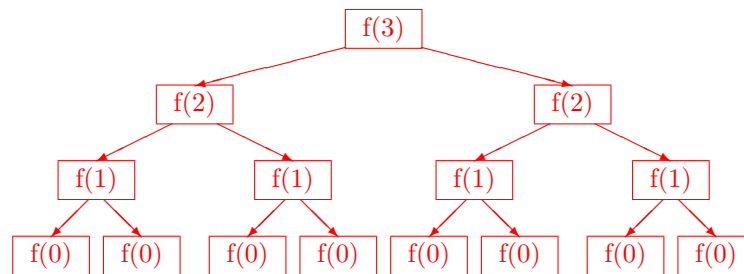
$$f(0) = 1$$

$$f(n) = f(n-1) - n + f(n-1) + 2$$

- (a) Complete the following table of values of $f(\)$:

n	0	1	2	3	4	5	6	...	10
f(n)	1	3	6	11	20	37	70	...	1034

- (b) Show the recursion tree (*i.e.*, draw a tree with one node for each function invocation, where the children of a node are the function calls that it makes) for evaluating $f(3)$:



- (c) Write a Java version of $f(\)$ which will produce the same results *and have the same recursion tree*:

```

public int f(int n) {
    if (n == 0) return 1;
    return f(n-1) - n + f(n-1) + 2;
}
  
```

- (d) Estimate the big-O running time of your function in part (c)

It will take exponential time, because of the doubling of the nodes in each level of the recursion tree, so it will be $O(2^n)$.

- (e) What is an easy way to make this function much more efficient, and what does the big-O running time become?

By replacing the two calls to $f(n-1)$ with a single call times 2, the running time will become $O(n)$.

2. (6 points) What will be the output of the following C++ program?

```
void printem(int n, int a, int &b)
{
    cout << n << ": " << a << " " << b << endl;
    a = 0;
    b = 0;
}

int main()
{
    int x, y;
    int *p, *q;

    x = 4; y = 2; printem(1, x, y);

    x += 3; y += 3; printem(2, x, y);

    p = new int;
    *p = 6; q = &x; printem(3, *p, *q);

    *q = *p; x += 3; printem(4, *p, *q);

    q = p; x += 3; printem(5, *p, *q);

    y = x; x = *p; printem(6, x, y);
}
```

(Recall that `cout << a << b << endl;` is the C++ equivalent of `System.out.print(a); System.out.print(b); System.out.println();`)

```
1: 4 2
2: 7 3
3: 6 7
4: 6 9
5: 6 6
6: 0 3
```

3. (2 points) In the above program, what statement needs to be added (and where) to properly deallocate the dynamically allocated memory?

Add the statement `delete p;` after the last line of `main`.

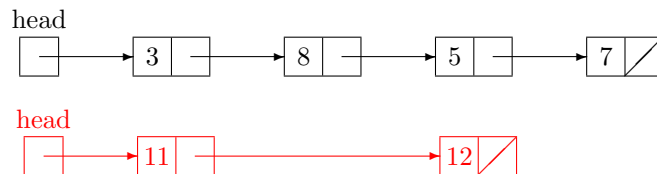
4. (10 points) For these questions, use the following definition for a `Node`:

```
class Node {
    int item;
    Node next;
}
```

- (a) Consider the following Java method (in some class that has access to the definition of `Node`):

```
void someFunc(Node head)
{
    Node p = head;
    Node q;
    while (p != null && p.next != null) {
        q = p.next;
        p.item = p.item + q.item;
        p.next = q.next;
        p = p.next;
    }
}
```

Show the result of executing `someFunc(head)` for the following list:



- (b) If `someFunc` were written in C++, what additional statement would need to be added (and where) to properly deallocate `Nodes`?
 At the end of the `while` loop, add the statement `delete q;`.
- (c) Using the above definition of `Node`, write a Java method `countOddPairs(Node head)` which returns a count of the number of pairs of *adjacent* nodes in the list whose sum is odd. For example, calling `countOddPairs(head)` for the above list should return 2, since $3 + 8$ and $8 + 5$ are both odd, while $5 + 7$ is even. Be sure to handle corner cases correctly.

```
public int countOddPairs(Node head)
{
    int count = 0;
    Node p = head;
    while (p != null && p.next != null) {
        if ((p.item + p.next.item) % 2 == 1) count++;
        p = p.next;
    }
    return count;
}
```

5. (10 points) Here is Java code for one version of the insertion sort algorithm:

```
public void insertionSort(int[] a, int n)
// sort a[0 .. n-1] into ascending order
{
    for (int k = 1; k < n; k++) {
        // insert a[k] into the sorted region a[0 .. k-1]
        for (int i = k; i > 0 && a[i - 1] > a[i]; i--) {
            int temp = a[i - 1];
            a[i - 1] = a[i];
            a[i] = temp;
        }
    }
}
```

- (a) Trace the operation of insertion sort by showing the contents of $a[]$ at the end of each pass through the outer loop; use the array $\{5, 1, 2, 3, 4\}$, where $n = 5$:

k	a[0]	a[1]	a[2]	a[3]	a[4]
<i>initial</i>	5	1	2	3	4
1	1	5	2	3	4
2	1	2	5	3	4
3	1	2	3	5	4
4	1	2	3	4	5

- (b) How many item comparisons ($a[i - 1] > a[i]$) are performed in part (a)?

7

- (c) How many swaps are performed in part (a)?

4

- (d) What would be the numbers of comparisons and swaps for the array $\{50, 1, 2, \dots, 49\}$, where $n = 50$?

97 comparisons, and 49 swaps.

6. (10 points) Here is Java code for the selection sort algorithm:

```
public void selectionSort(int[] a, int n)
// sort a[0 .. n-1] into ascending order
{
    for (int k = n - 1; k > 0; k--) {
        // find index of largest item in a[0 .. k]
        int largest = 0;
        for (int i = 1; i <= k; i++) {
            if (a[i] > a[largest]) largest = i;
        }

        // move largest item to position k
        int temp = a[largest];
        a[largest] = a[k];
        a[k] = temp;
    }
}
```

- (a) Trace the operation of selection sort by showing the contents of `a[]` at the end of each pass through the outer loop; use the array {5, 1, 2, 3, 4}, where `n = 5`:

k	a[0]	a[1]	a[2]	a[3]	a[4]
<i>initial</i>	5	1	2	3	4
4	4	1	2	3	5
3	3	1	2	4	5
2	2	1	3	4	5
1	1	2	3	4	5

- (b) How many item comparisons (`a[i] > a[largest]`) are performed in part (a)?

10

- (c) How many swaps are performed in part (a)?

4

- (d) What would be the numbers of comparisons and swaps for the array {50, 1, 2, ..., 49}, where `n = 50`?

$49 + 48 + \dots + 2 + 1 = 1225$ comparisons, and 49 swaps.

7. (12 points) Consider the following Haskell code defining a datatype of binary trees, a binary search tree insertion function, and a preorder tree traversal:

```
data Tree = Node(Int, Tree, Tree) | Null

bstInsert(x, Null) = Node(x, Null, Null)
bstInsert(x, Node(y, left, right)) =
  if x == y then Node(y, left, right)
  else if x < y then Node(y, bstInsert(x, left), right)
  else Node(y, left, bstInsert(x, right))

preOrder(Null) = [ ]
preOrder(Node(x, left, right)) = [x] ++ preOrder(left) ++ preOrder(right)
```

(Recall that ++ is the list concatenation operator: [1, 2] ++ [3, 4] yields [1, 2, 3, 4].)

- (a) Write a Haskell function `find`, with type `(Int, Tree) -> Bool`, such that `find(x, t)` returns `true` if `x` is in the binary search tree `t`, and `false` otherwise:

```
find(x, Null) = false
find(x, Node(y, left, right)) =
  if x == y then true
  else if x < y then find(x, left)
  else find(x, right)
```

- (b) Complete the following definition of a Haskell (actually `HasCl`) function `bstInsertAll`, with type `([Int], Tree) -> Tree`, such that `bstInsertAll(xs, t)` is the result of inserting all of the values from the list `xs` into the binary search tree `t`:

```
bstInsertAll([ ], t) = t

bstInsertAll([x : xs], t) = bstInsert(x, bstInsertAll(xs, t))
```

- (c) Give a Haskell definition for an inorder traversal function:

```
inOrder(Null) = [ ]
inOrder(Node(x, left, right)) = inOrder(left) ++ [x] ++ inOrder(right)
```

8. (8 points) What is the output when the main method of the Mystery class is executed?

```

public interface LazyList {
    public int next();
}

public class Ints implements LazyList {
    private int n;

    public Ints(int n)
    {
        this.n = n;
    }

    public int next()
    {
        n++;
        return n;
    }
}

public class Filter implements LazyList {
    private LazyList list;
    private int p;

    public Filter(LazyList list, int p)
    {
        this.list = list;
        this.p = p;
    }

    public int next()
    {
        int n = list.next();
        while (n % p == 0) {
            n = list.next();
        }
        return n;
    }
}

public class Mystery implements LazyList {
    private LazyList list;

    public Mystery()
    {
        this.list = new Ints(1);
    }

    public int next()
    {
        int p = list.next();
        list = new Filter(list, p);
        return p;
    }

    private static void test(LazyList x)
    {
        for (int i = 0; i < 5; i++) {
            System.out.print(x.next() + " ");
        }
        System.out.println();
    }

    public static void main(String[] args)
    {
        System.out.println("Ints");
        test(new Ints(1));

        System.out.println("Filter");
        test(new Filter(new Ints(1), 2));

        System.out.println("Mystery");
        test(new Mystery());
    }
}

```

```

Ints
2 3 4 5 6
Filter
3 5 7 9 11
Mystery
2 3 5 7 11

```