First Exam
CS 1102 Computer Science 2

Fall 2018

Thursday October 4, 2018
Instructor Muller

KEY

Before reading further, please arrange to have an empty seat on either side of you. Now that you are seated, please write your name **on the back** of this exam.

This is a closed-notes and closed-book exam. Computers, calculators, and books are prohibited.

- Partial credit will be given so be sure to show your work.
- Feel free to write helper functions if you need them.
- **Please write neatly.**

Problem	Points	Out Of
1		7
2		4
3		5
4		4
Total		20

1. Basic Java [7 Points]

1. (2 Point) Write a function `int add(int m, int n)` such that a call `add(m, n)` returns the integer sum of `m` and `n`.

   Answer:

   ```
   int add(int m, int n) { return m + n; }
   ```

2. (2 Points) Write a function `boolean containsDuplicates(int[] a)`. A call `containsDuplicates(a)` should return `true` if `a` contains at least one duplicate integer. Otherwise the call should return `false`. Assuming that `a` contains N integers, how many comparisons might your algorithm perform?

   Answer:

   ```
   boolean containsDuplicates(int[] a) {
     if (a.length < 2) return false;
     for (int i = 0; i < a.length - 1; i++)
       for (int j = i + 1; j < a.length; j++)
         if (a[i] == a[j]) return true;
     return false;
   }
   ```

   This function could compare $(N-1)+(N-2)+\ldots+1$, roughly $N(N-1)/2$ items.

3. (1 Point) In Java, two-dimensional arrays are effectively layed out in memory in *row-major order*. That is, the elements of row 0 are allocated in consecutive memory words. The elements of row 1 are allocated somewhere else consecutively and so forth. In a sentence or two, is it possible to observe a difference in performance if 2D array elements were processed row by row versus column by column? Why?

Answer:

Column by column processing has poor locality.

4. (2 Points) Write a function `boolean contains(int n, Node node)` such that a call `contains(n, node)` returns `true` if the linked list of `Node`s starting with `node` contains `n`. Otherwise it should return `false`. A `Node` is defined as follows

```
class Node {
  int info;
  Node next;

  Node(int info, Node next) { this.info = info; this.next = next; }
}
```

Answer:

```
boolean contains(int n, Node node) {
  if (node == null)
    return false;
  else
    return (n == node.info) || contains(n, node.next);
}
```

```
boolean contains(int n, Node node) {
  while (node != null) {
    if (n == node.info) return true;
    node = node.next;
  }
  return false;
}
```

2. Storage Diagrams [4 Points]

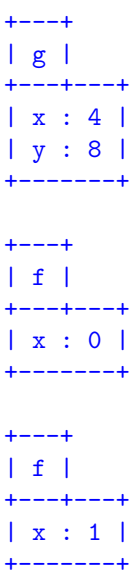
1. (2 Points) Consider the code below and draw the state of the call stack and the heap if execution were started with a call `f(1)` and stopped immediately after line (1).

```
int g(int x) {
  int y = x * 2;           (1)
  return y;
}

int f(int x) {
  if (x == 0)
    return g(4);
  else
    return f(x - 1);
}
```

Stack Heap

Answer:



2. (2 Points) Consider the code below and draw the state of the call stack and the heap if execution were started with a call `f(2, null)` and stopped immediately *before* line (1) was executed.

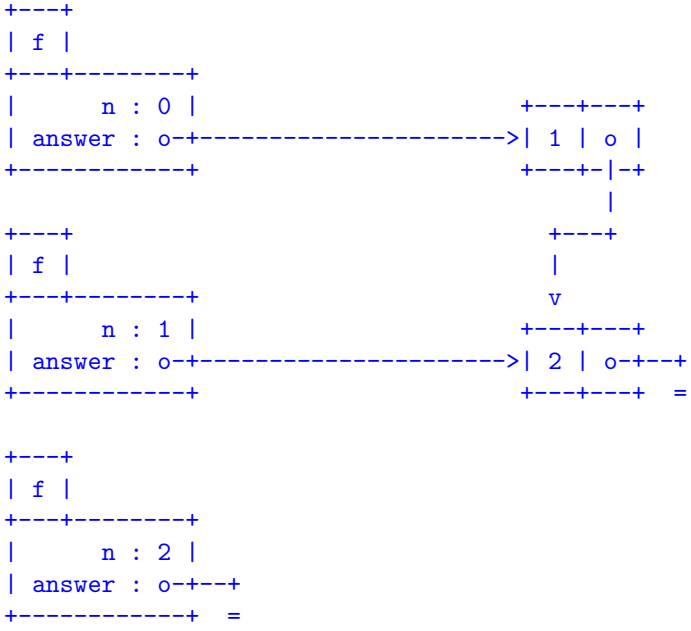
```
class Node {
  int info;
  Node next;

  Node(int info, Node next) { this.info = info; this.next = next; }
}

Node f(int n, Node answer) {
  if (n == 0)
    return answer;                                (1)
  else
    return f(n - 1, new Node(n, answer));
}
```

Stack Heap

Answer:



3. Stacks, Queues, Deques and PQs [5 Points]

1. (1 Point) Priority queues are different from ordinary queues in that they are sensitive to an *ordering* of the items housed in the queue. Order-sensitive data structures like PQs come with warnings that one has to be careful about housing mutable keys in the data structure. In a sentence or two, why does one have to be careful about housing mutable keys in an order-sensitive data structure? Why are immutable keys safer?

   Answer: If a key housed in the data structure is mutated after it has been inserted in the data structure, the entire data structure is corrupted.

2. (4 Points) Let's say that we have generic ADTs `Stack<T>`, `Queue<T>`, `Deque<T>` and `MaxPQ<T>` all as specified on the attached sheet and with implementations `StackC<T>`, `QueueC<T>`, `DequeC<T>` and `MaxBinaryHeap<T>` (resp). Implement a variation of `MaxPQ<T>` called `NthMaxPQ<T>`. The new ADT should be just like `MaxPQ<T>` but instead of the operation `T removeMax()`, the new ADT has an operation `T removeNthMax(int n)` such that if `pq` is an `NthMax` priority queue, a call `pq.removeNthMax(n)` will return the `nth` largest element of `pq`. A call `pq.removeNthMax(1)` will return the maximum in the usual way. A `NoSuchElementException` should be thrown if `n` is an invalid integer. The new ADT has the following API. *NB: the code for* `MaxBinaryHeap<T>` *is attached but one doesn't need to know the details of that code to solve this problem.*

```
public interface NthMaxPQ<T extends Comparable<T>> {
  void insert(T item);
  T removeNthMax(int n);
  boolean isEmpty();
  int size();
  String toString();
}
```

```
public class NthMaxPQC<T extends Comparable<T>> implements NthMaxPQ<T> {
```

Answer:

```
  private MaxPQ<T> pq;

  public NthMaxPQC() { pq = new MaxBinaryHeap<T>(); }

  public T removeNthMax(int n) {
    if (n < 1 || n > pq.size()) throw new NoSuchElementException("Bad input");
    Queue<T> q = new QueueC<T>();
    for (int i = 1; i < n; i++)
      q.enqueue(pq.removeMax());
    T answer = pq.removeMax();
    while(!q.isEmpty())
      pq.insert(q.dequeue());
    return answer;
  }
  public void insert(T item) { pq.insert(item); }
  public boolean isEmpty() { return pq.isEmpty(); }
  public String toString() { return pq.toString(); }
  public int size() { return pq.size(); }
}
```

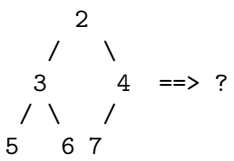
4. Binary Heaps [4 Points]

1. (2 Points) A perfect binary tree of height k has exactly 2^k leaves. How many leaves might a complete binary tree of height k have?

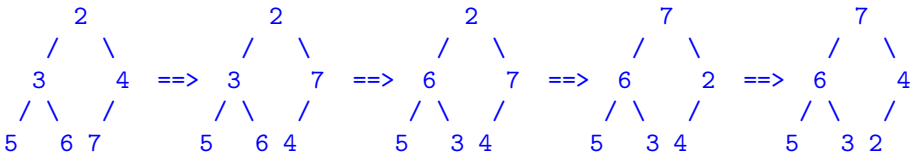
   Answer: A complete binary tree will have between 2^{k-1} and 2^k leaves.

2. (2 Points) Show the sequence of complete binary trees arising from the process of making a max binary heap from the elements of the following integer array.

```
int[] a = {0, 2, 3, 4, 5, 6, 7}
```



Answer:



Code for Section 3

```
public interface Stack<T> {
  void push(T item);
  T pop();
  int size();
  boolean isEmpty();
  String toString();
}
```

```
public interface Queue<T> {
  void enqueue(T item);
  T dequeue();
  int size();
  boolean isEmpty();
  String toString();
}
```

```
public interface Deque<T> {
  void pushLeft(T item);
  T popLeft();
  void pushRight(T item);
  T popRight();
  boolean isEmpty();
  int size();
  String toString();
}
```

```
public interface MaxPQ<T extends Comparable<T> {
  void insert(T item);
  T removeMax();
  boolean isEmpty();
  int size();
  String toString();
}
```

```
public class MaxBinaryHeap<T extends Comparable<T>> implements MaxPQ<T> {

  private T[] pq; // 1 .. N
  private int N;

  public MaxBinaryHeap() {
    pq = (T[]) new Object[2];
    N = 0;
  }

  private void resize(int capacity) {
    T[] temp = (T[]) new Object[capacity];
    for (int i = 1; i <= N; i++)
      temp[i] = pq[i];
    pq = temp;
  }

  private boolean less(int i, int j) {
    return pq[i].compareTo(pq[j]) < 0;
  }

  private void exchange(int i, int j) {
    T swap = pq[i];
    pq[i] = pq[j];
    pq[j] = swap;
  }

  private void swim(int k) {
    while(k > 1 && less(k / 2, k)) {
      exchange(k, k / 2);
      k = k / 2;
    }
  }

  private void sink(int k) {
    while (2 * k <= N) {
      int j = 2 * k;
      if (j < N && less(j, j+1)) j++;
      if (!less(k, j)) break;
      exchange(k, j);
      k = j;
    }
  }

  public void insert(T item) {

    // double size of array if necessary
    if (N == pq.length - 1) resize(2 * pq.length);

    // add item, and percolate it up to maintain heap invariant
    pq[++N] = item;
    swim(N);
  }

  public T removeMax() {
    if (isEmpty()) throw new NoSuchElementException("Priority queue underflow");
```

```
    T max = pq[1];
    exchange(1, N--);
    sink(1);
    pq[N+1] = null;     // to avoid loiteing and help with garbage collection
    if ((N > 0) && (N == (pq.length - 1) / 4)) resize(pq.length / 2);
    return max;
   }

  public T peek() {
    if (isEmpty()) throw new NoSuchElementException("Priority queue underflow");
    return pq[1];
  }

  public String toString() { return "Not implemented"; }
  public boolean isEmpty() { return N == 0; }
  public int size() { return N; }
}
```