CS2223 Algorithms **D Term 2018**
Exam 1 **April 03, 2018**

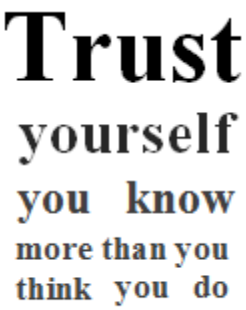
NAME: _RUBRIC FOR EXAM GRADERS_____

Instructions:

- Time allowed: 50 minutes
- Show your work and justify your answers
- Use the space provided to write your answers

Total	Q1	Q2	Q3	Q4	Q5

DO NOT OPEN EXAM UNTIL INSTRUCTED TO DO SO!!



Question 1. Short Answer Questions

For each of the following statements: (+3 correct answer; +2 provide explanation)

If the statement is *true*, circle **True** and explain why.
If the statement is *false*, circle **False** and explain *why the statement is false*.

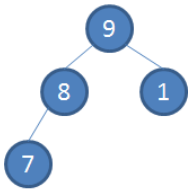
Your explanations should be ***brief*** (using about *one sentence*), but complete.

(a) [5 pts.] **True / False** : You are given an array containing 8 integers in ascending order and you want to check whether the array contains a specific value. You can do this using no more than four comparisons **in the worst case**.

TRUE. You need 1+Floor(log N) = 1+3 = 4.

(b) [5 pts.] **True / False** : In a max heap with N>3 values, the root contains the largest value of the heap, and the next two largest values in the heap are (in any order) the left and right child of the root.

FALSE: The top node contains the max, and the 2nd largest must be a child of the root; however the 3rd largest could be a child of the 2nd largest (show example).



(c) [5 pts.] **True / False** : You have an ordered linked list with N nodes storing integers in ascending order. You can use binary search to determine whether the list contains a value by inspecting no more than log (N) nodes **in the worst case**.

FALSE: A linked list doesn't offer the ability to perform binary search. You have to search each node in sequential order, and in the worst case you have to inspect all N nodes. You can't index into a linked list

(d) [5 pts.] **True / False** : In a max heap, you can delete the largest value in the heap in time that is independent of the number of values in the heap.

FALSE: You can find the largest value in constant time, but to delete it, you will need to swap with final value in the heap and then call sink() which has performance of ~log(N) where N is the number of elements in the heap.

Question 2. (20 pts.) Mathematical Analysis

You are given four unknown integers A, B, C and D. You are only allowed to call `equals(X, Y)` to see if X and Y are equal to each other. **You are told that there are only two distinct values in these four integers**. For example, it could be the case that [A=1, B=9, C=9, D=1] and the distinct values are 1 and 9.

[4 pts.] What is the least number of times you need to call `equals(X,Y)` to determine the two distinct values for any possible four values of A, B, C, and D? **[5 pts.]** Describe algorithm that exhibits behavior.

TWO DISTINCT WAYS OF READING THIS PROBLEM, AND I WAS FLEXIBLE WHILE GRADING

(1) If you thought it meant two pairs of distinct numbers (for a total of four) then you only need 1 comparison in the worst case to determine the distinct values: If (A==B) then return A,C else return A,B

(2) If you thought it meant that you could have 1 value repeated three times, then in worst case need 2 comparison. If (A != B) return A,B; else they are same, so check if B=C: if it does, then return A,D since D has to be different. If B != C then return B,C

(c) [5pts.] Let **S(N)** be the number of times `Math.sqrt()` is invoked in `process` with an array of length N which is a power of 2. Write a recurrence relationship that determines the upper bound (i.e., worst case) for the number of times `Math.sqrt()` is called.

```
static int process(int[] a, int lo, int hi) {
  if (lo == hi) { return (int) Math.sqrt(a[lo]); }
  int mid = lo + (hi-lo)/2;
  int x = process(a, lo, mid) + process(a, mid+1, hi);

  for (int i = lo; i <= hi; i += 2) {
    if (Math.sqrt(a[i]) == x) { x++; }
  }
  return x;
}
```

S(N) = 2*S(N/2) + N/2

(+2 point for S(n) = ...) (+1 point for 2*), (+1 for S(N/2)), (+1 for N/2)
Note that S(1) = 1 since there is a final Sqrt invocation for problems of size 1

(d) [6pts.] Derive an exact solution to the recurrence for S(N) when N is a power of 2.

S(N) = 2*S(N) + N/2 and so write
S(N) = 2*(2*S(N/4)+N/4) + N/2, which equals 4*S(N/4) + N
S(N) = 4*(2*S(N/8)+N/8)+2*N = 8*S(N/8) + 3*(N/2)

so it looks like we can repeat K=log N times, to get

S(N) = 2^k*S(N/2^k) + k*(N/2). Now S(1) = 1, so eventually we get to
S(N) = N*1 + log N*(N/2) or N + (1/2)(N*log N)

(+2 point if they have telescoping logic). (+1 if N*Log N appears) (+1 if N*LOG N/2 appears). (+2 if they have N+ ... since that term doesn't reduce to 0).

Question 3. (20 pts.) Type Question

You are asked to write a method that returns a copy of a **StackOfInteger** object. I have given you an extra **StackOfInteger** object and an extra **QueueOfInteger** object that you can use for additional storage in your answer. For this question, there is no iterator for the **StackOfInteger**.

(a) **[14 pts.]** You can provide Java code or describe your answer in pseudocode.

```
/** Given a stack of integers, return a copy of the stack. When this method returns
the original stack must contain its original contents in their original positions. */
 public static StackOfInteger copy (StackOfInteger stack) {
    StackOfInteger extra = new StackOfInteger();
    QueueOfInteger queue = new QueueOfInteger();
```

Multiple ways to handle this. Note that to make a copy of 'stack' you have to disturb it. Let's get started

```
+1      while (!stack.isEmpty()) {
+2       extra.push(stack.pop());
        }                              // at this point, extra is a reverse copy of 'stack
+2      StackOfIntegers copy = new StackOfInteger();
+1      while (!extra.isEmpty())  {
+2       int x = extra.pop()          // note first one popped from extra was the last one pop'd from stack
+2       stack.push(x)                // by pushing both onto stack and copy, we recreate both at same time
+2       copy.push(x);
        }
+2      return copy                  // return the copy
```

b) **[6 pts.]** If `stack` has N elements, compute the total number of `push` OR `pop` calls needed in the worst case on the stack passed into your copy method.

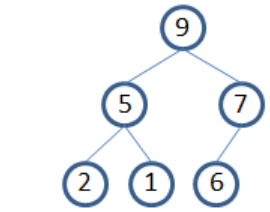
You will need N pop operations to be able to get the values. Then you will need to make N push operations to restore the stack to its original structure. This gives N+N = 2N

Note that I was only asking for the operations on the original stack. MANY, MANY students just counted up all push and pop operations, and I had graders try to figure out if you had individual separated out the calls for stack.

Question 4. (20 pts.) Heap (you don't really need the code, but I'm providing just in case)

```
public class Heap {
  int[] pq; // Store in pq[1..N]
  int N;    // number of items in Heap

  public Heap (int initCapacity) {
    pq = new int[initCapacity + 1];
  }

  public void insert (int x) {
    pq[++N] = x;
    swim(N);
  }

  public int delMax() {
    int max = pq[1];
    exch(1, N--);
    pq[N+1] = null;
    sink(1);
    return max;
  }

  public boolean isEmpty() { return N == 0;  }
  public int size()        { return N; }

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

  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;
      exch(k, j);
      k = j;
    }
  }
```

(a) **[12 pts.]** Assume you create **heap=new Heap(8)** and call **insert** with the following values in this order: **6**, **2**, **9**, **5**, **1** and **7**. Once all values are inserted, draw the tree representation of the final heap. Also show the array representation of **pq** that stores its values.



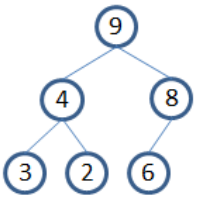
-	9	5	7	2	1	6	-	-

(+2 points for Array with 9 cells.)
(+1 points if a[0] is ignored)

(+1 point for each of 9..5..7..2 in left to right)

(+1 point for binary heap tree with 9,5,7,2 properly placed)

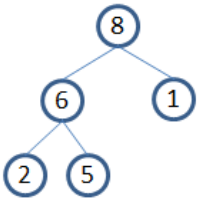
(+1 point for everything right)



-	9	4	8	3	2	6	-	-	-

Alternate version asked to insert 6, 3, 9, 4, 2 and 8 into heap of size 9

(b) **[8 pts.]** With this same heap, call **delMax** twice in a row and **insert** the value **8**. Now draw the tree representation of the final heap. Also show the array representation of **pq** that stores its values.

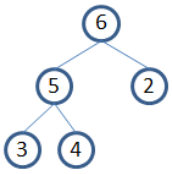


-	8	6	1	2	5	-	-	-

(+3 point for final heap)

(+3 point for binary heap tree with 5 nodes properly placed)

(+1 if a[0] is ignored)
(+1 if last three cells are empty "-")



-	6	5	2	3	4	-	-	-	-

Alternate version asked to delmax twice and insert 5.

If Student makes a single mistake with insertion on part (a), try to recreate correct answer for part (b). That is, if (a) has mistakes, assume (b) takes (a) as input.

Question 5. (20 pts.) Algorithm Question

The following is an example of a ***Gap*** array of size N=6, which contains N distinct integers in ascending order from the range of N+1 integers [0, N]. As you can see, the integer "4" is the only value missing.

0	1	2	3	5	6
a[0]					**a[5]**

To summarize, a ***Gap*** array of size N has the following properties:

- **N ≥ 3** and all values in the array appear in strictly ascending order
- **a[0]=0** and **a[N-1]=N**
- The array contains <u>all but one</u> of the integers from 0 to N.

(a) **[15 pts.]** Design an algorithm that computes **gap**, the missing integer from the range [0, N], which does not appear in the ***Gap*** array. The order of growth (as a function of N) of the running time of your algorithm must be **log N**. **<u>(To receive 7/15 pts on this part, the performance can be N).</u>** Write your answer in pseudocode or Java.

(b) **[5 pts.]** **<u>In the worst case</u>**, what is the most number of array inspections your algorithm makes?

```
/** Return missing number, gap, in a[0, N-1]. */
public static int findGap (int[] a) {
```

There many ways to solve this problem, many of them discovered by students. Here is my implementation with points:

```
+1    int lo = 1                 // can avoid 0th and last one since they are known in advance
+1    int hi = a.length-2
+2    while lo <= hi             // must be <= otherwise strange case happens
+1      mid = lo + (hi-lo)/2     // or any suitable midpoint computation
+3      if a[mid] > mid          // gap is on the left….
+2        hi = mid-1             // so close in on it
        else
+2        lo = mid + 1           // gap is on the right
+3    return lo                  // could also be high + 1
```

Now how to compute number of comparisons? Each pass through, as with binary array search, there is a comparison, thus:

(1 + Floor(log (N - 2)) – and you can review the posted code solution to see that this is confirmed. If you start with lo=0 and hi=a.length-1, then (1 +Floor(log N))

Note: FOR 7/15 STUDENT COULD HAVE DONE SIMPLE LINEAR SEARCH UNTIL YOU SEE GAP, WHICH WOULD REQUIRE N-2 COMPARISONS IN WORST CASE, or N-1 if you double check 1st and last.