Class Meeting #10 Exams 1 Debriefing

COS 226 — Spring 2017 *Jérémie Lumbroso* lumbroso@cs.princeton.edu

WRITTEN EXAM 1

Stats from WE1



- Average: 55
- Easiest question (Q3, people got on average 97% pts)
- Hardest question (Q9, people got on average 53.4% pts)

Scanning: Pros and Cons

Pros

- Fast grading
- Reprocessable grading
- Easier to assign partial credit
- Safety for all!
- Statistics that help improve exams
- Cons
 - UNSTAPLING
 - Filling in the bubbles

Filling in a bubble

0	۲	0	0	Ő	۲	\bigcirc	۲	۲	0	0	0		۲	۲	۲	۲	۲
0.06	0.06	0.06	0.06	0.06	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.07	0.08	0.08	0.09	0.09
C	•	•	•	• •	•	•	٠	•	•	٠	•	٠	•	•	•	• •	
0.09	1.00	1.00	1.00 1	.00 1.0	0 1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00 1	.00 1.0	0 3
	•	•	•	• •	•	•	۲	٠		•	٠	۲	•	•	•	• •	
0.15	1.00	1.00	1.00 1	.00 1.0	0 1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00 1	.00 1.0	0 2
	•	•	•	• •	•	•	۲		٥	۲	٠	•	•	•	•	•	
0.22	1.00	1.00	1.00 1	.00 1.0	0 1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00 1	.00 1.0	o
è	۲	•	•	• •	•	•	•	•	۲	•	٠	٢	•	•	•	• •	5
0.30	1.00	1.00	1.00 1	.00 1.0	0 1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00 1	.00 1.0	0 5
×	ø		Ŕ		X	xá,	X	Ø		×	X	X		ð.	ø	X	ø
Ø				X			X		X	X		X	X			X	X
0.41	0.41	0.41	0.41	0.41	0.42	0.42	0.42		0.43	0.44	0.44	0.44	0.44	0.44	0.44	0.45	

Question-Level Statistics

0.8

0.8.1 \stepcounter {CounterSubQuestion}\textbf {{\printSubQ	Color flip 6\smallskip Color flip 7\smallskip	9.96 %								
\makebox [4em][c]{constant} \makebox [4em][c]{logarithmic} \makebox [4em][c]{linear} \makebox [4em][c]{linearithmic} \makebox [4em][c]{quadratic}\bigskip	88.38 % 0.00 % 2.90 % 0.00 % 10.79 %	Color flip 8\smallskip Color flip 10\smallskip Color flip 12\smallskip Color flip 14\smallskip Color flip 16\smallskip Color flip 16\smallskip Color flip 18\smallskip	87.97 % 7.05 % 7.05 % 1.24 % 0.41 % 2.90 %							
0.8.2 \stepcounter {CounterSubQuestion}\textbf {{\printSubQuestionNum }}~{Order of growth of best case of \textsc										
{Method II}}	1010	Rotate 6 left/smallskip	78.01 %							
\makebox [4em][c]{constant} \makebox [4em][c]{logarithmic}	1.24 %	Rotate 6 right/smallskip Rotate 7 left/smallskip	0.83 %							
\makebox [4em][c]{linear}	4.15 %	Rotate 7 right\smallskip	124 %							
\makebox [4em][c]{linearithmic} \makebox [4em][c]{quadratic}\bigskip	90.46 %	Rotate 8 right\smallskip	13.69 %							
0.8.3 \stepcounter {CounterSubQuestion}\textbf {\printSubQ	Rotate 10 left\smallskip Rotate 10 right\smallskip	2.49 % 10 83.40 %								
{Method III}}	and the second	Rotate 12 left\smallskip	0.83 %							
\makebox [4em][c]{constant} \makebox [4em][c]{logarithmic} \makebox [4em][c]{linear} \makebox [4em][c]{linearithmic} \makebox [4em][c]{quadratic}\bigskip	87.14 % 2.07 % 11.20 % 0.41 % 0.00 % 0.00 % 11.20 %	Rotate 12 right/smallskip Rotate 14 left/smallskip Rotate 14 right/smallskip Rotate 16 left/smallskip Rotate 16 right/smallskip Rotate 18 left/smallskip Rotate 18 right/smallskip	76.35 % 0.00 % 0.41 % 0.41 % 0.00 % 0.00 % 0.41 %							
• A lot of statistics	inform up on whatha	r quaatia								

- A lot of statistics inform us on whether questions were:
 - hard,
 - or misunderstood,
 - or poorly designed

Q1. (i) Union-Find



- Friends: 1st degree; Friendly: any degree.
- Bad question:

Q5. (ii) Mergesort (1)

(ii) [3 pts] We now assume it is possible to merge two *sorted* sublists (of any size) in constant time¹. Express the average running time, as a function of N, of the standard textbook 2-way mergesort, modified only to use this new (imaginary) constant-time merging algorithm.

- do you understand the analysis of mergesort's runtime well enough to be able to rebuild analysis with a different runtime for merge?
- do you understand what run time measures?

Q5. (ii) Mergesort (2)

- Partial: ~log N
- Full: ~N



N



Q8. Binary Trees (1)

Average success rate: 60.08%

Q8. Binary Trees and BSTs (5 points).

If the in-order traversal of that binary tree prints nodes labeled C D E N P X Y, and the post-order traversal of a binary tree prints nodes labeled D C E P Y X N, then what sequences of labels does the pre-order traversal print?

- do you know to distinguish between binary tree and BST?
- do you know what is a tree traversal?
- do you know what is pre-order, in-order?
- do you know what is post-order?

Q8. Binary Trees (2)

- In-order: CDENPXY
- Post-order: D C E P Y X N



²nd visit: inorder

Starting point is last of post-order: root





Q8. Binary Trees (3)





YXPNEDC	YX PNEDC
NECUXYY	NECDXPY
NECDXPY	NECDXPY
ΝΕ(ΟχΡΥ	NXYPECD
PDXCENY	NECDXPY
INERVX N	NDPCXEY
CICEPIAN	DCEINYX
CEDNXYP	NECDXPY
E(PXYPN	PECDYNX
CEDNXYP	YXPNEDC
NECDXPY	CDREPYY
$N \in C \cup X \cap Y$ (???)	
NECDXPY	NECDXPY
NECDXPY	DCEPYNX
C D NXPYE	NECDXPY
NECDXPY	CEDNXAY

Q8. Binary Trees (4)

		•			<u> </u>			()		
						tk				
Barcode: 330)36 Field: e	ex8 Grade	ed: -1							
							n VI	>\/		
ND	-	FX	V		Nt	2C	DV	r y		
					•					
									ťK	
	Barcode:	33090	Field: ex8	Graded:	5					
Previous										
	NI	E	C	T	V	D	\sim			1777
	N	E	C	V	\wedge	1	((:::)
	Dreviou	•								Next

Q10. Hash Functions (1)

ubQuestion}\textbf {{\printSubQuestionNum }}~{Less wasted space.}

\textsc {Separate chaining} 46.06 % \textsc {Linear probing}\bigskip 53.53 %



- Class did not do well on hash questions
- But there are ways to think about these methodically (either while studying or during exam)

Q10. Hash Functions (2)

- Linear probing: need 2x not the space, so array of references 8 bytes, 2N*8 bytes
- Separate chaining: list nodes need a lot of overhead per reference (and still containing the same reference), so even if table is ~N or ~1/2N then it takes more space



PROGRAMMING EXAM 1

Some remarks

- Statistics to come
- Big criteria was performance
 - submissions comfortable with manipulating ST.java (great!)
 - submissions that used a temporary array but did not iterate over it (ok)
 - submissions that used a temporary array and iterated over it (meh)
 - submissions that used MAX_EXPONENT in a loop (that is a big performance hit unfortunately)

Big performance hit: MAX_EXP.

```
// instance polynomial to `that`
public Polynomial add(Polynomial that) {
    Polynomial add;
    double[] coeffs = new double[MAX_EXPONENT];
    for (int i = 0; i < MAX_EXPONENT; i++) {
        coeffs[i] = coefficients.get(i) + that.coefficients.get(i);
    }
    add = new Polynomial(coeffs);
    return add;
}</pre>
```

ST.size() or ST.max()

```
// Known bug: This method is not adding the highest t
public Polynomial add(Polynomial that) {
    int size;
    if (this.maxExp < that.maxExp) {
        size = that.maxExp;
    }
    else {
        size = this.maxExp;
    double [] c = new double [size];
    for (int i = 0; i < size; i++) {
        if (this.coefficients.contains(i))
            c[i] += this.coefficients.get(i);
        if (that.coefficients.contains(i))
            c[i] += that.coefficients.get(i);
                                         public Polynomial add(Polynomial that) {
    }
                                             int max = Math.max(this.coefficients.max(), that.coefficients.max());
    Polynomial p = new Polynomial(c);
                                             double[] result = new double[max + 1];
    return p;
                                             for (Integer i : this.coefficients) {
                                                  for (Integer j : that.coefficients) {
                                                      if (i == j) {
}
                                                          result[i] = this.coeff(i) + that.coeff(j);
                                                      }
                                                      else if (this.coeff(j) == 0)
                                                          result[j] = that.coeff(j);
                                                      else if (that.coeff(i) == 0) {
                                                       result[i] = this.coeff(i);
                                                  }
                                              return new Polynomial(result);
                                         }
```

Linear allocation/constructor (1)

```
public Polynomial add(Polynomial that) {
    double[] sum = new double[MAX_EXPONENT - 1];
    for (Integer a : this.coefficients) {
        sum[a] += this.coeff(a);
    }
    for (Integer b : that.coefficients) {
        sum[b] += that.coeff(b);
    }
    Polynomial add = new Polynomial(sum);
    return add;
}
```

Linear allocation/constructor (2)

```
// Create a new immutable polynomial resulting from the addition of the
// instance polynomial to 'that'
public Polynomial add(Polynomial that) {
   ST<Integer, Double> addand = new ST<Integer, Double>();
   int max exp = 0;
   for (Integer exp : this.coefficients.keys()) {
        // StdOut.println(" " + exp);
        addand.put(exp, this.coefficients.get(exp));
        if (exp > max exp) {
            max exp = exp;
        }
    }
   for (Integer exp : that.coefficients.keys()) {
        if (addand.contains(exp)) {
            addand.put(exp, addand.get(exp)+that.coefficients.get(exp));
        J.
        else {
            addand.put(exp, that.coefficients.get(exp));
        }
        if (exp > max exp) {
            max exp = exp;
        }
    }
   double[] c = new double[max exp+1];
   for (Integer exp : addand.keys()) {
        c[exp] = addand.get(exp);
    Polynomial p = new Polynomial(c);
    return p;
```

Creating Result Polynomial (1)

private Polynomial() { .. }

// ...

Polynomial p = Polynomial();
p.coefficients = new ST<>();

Creating Result Polynomial (2)

Polynomial p = Polynomial (

new double[0]);

p.coefficients //<-- brand new // ST

Creating Result Polynomial (3)

private Polynomial(ST<> st) {
 coefficients = st;

}