Class Meeting #10
Exams 1 Debriefing

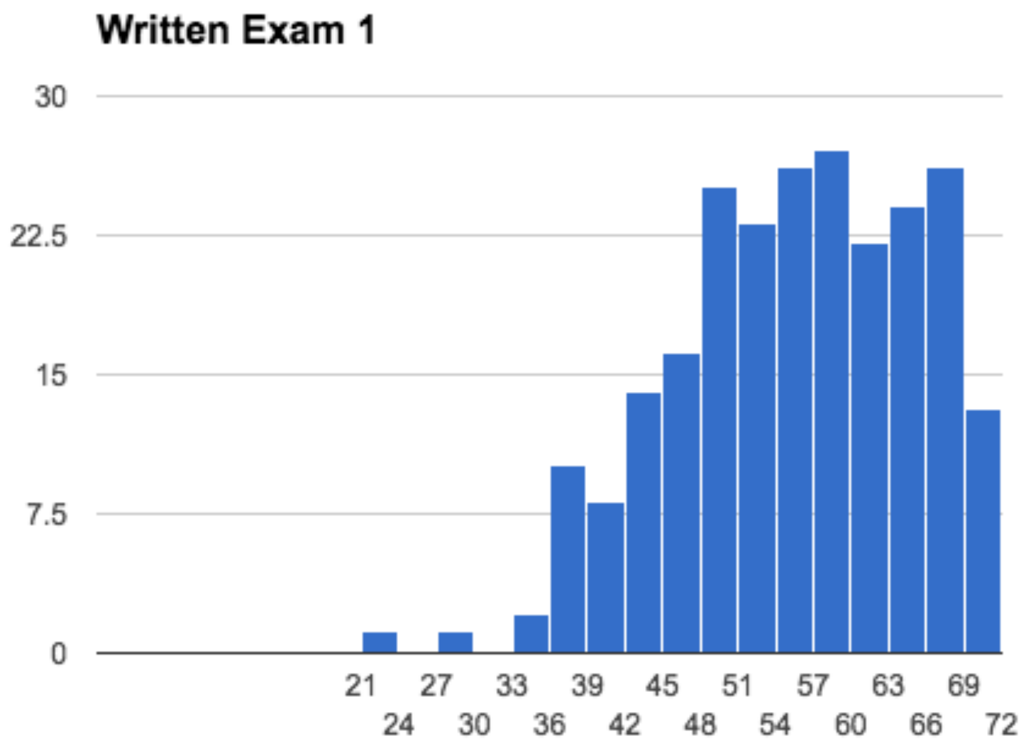
COS 226 — Spring 2017

Jérémie Lumbroso

lumbroso@cs.princeton.edu

WRITTEN EXAM 1

Stats from WE1

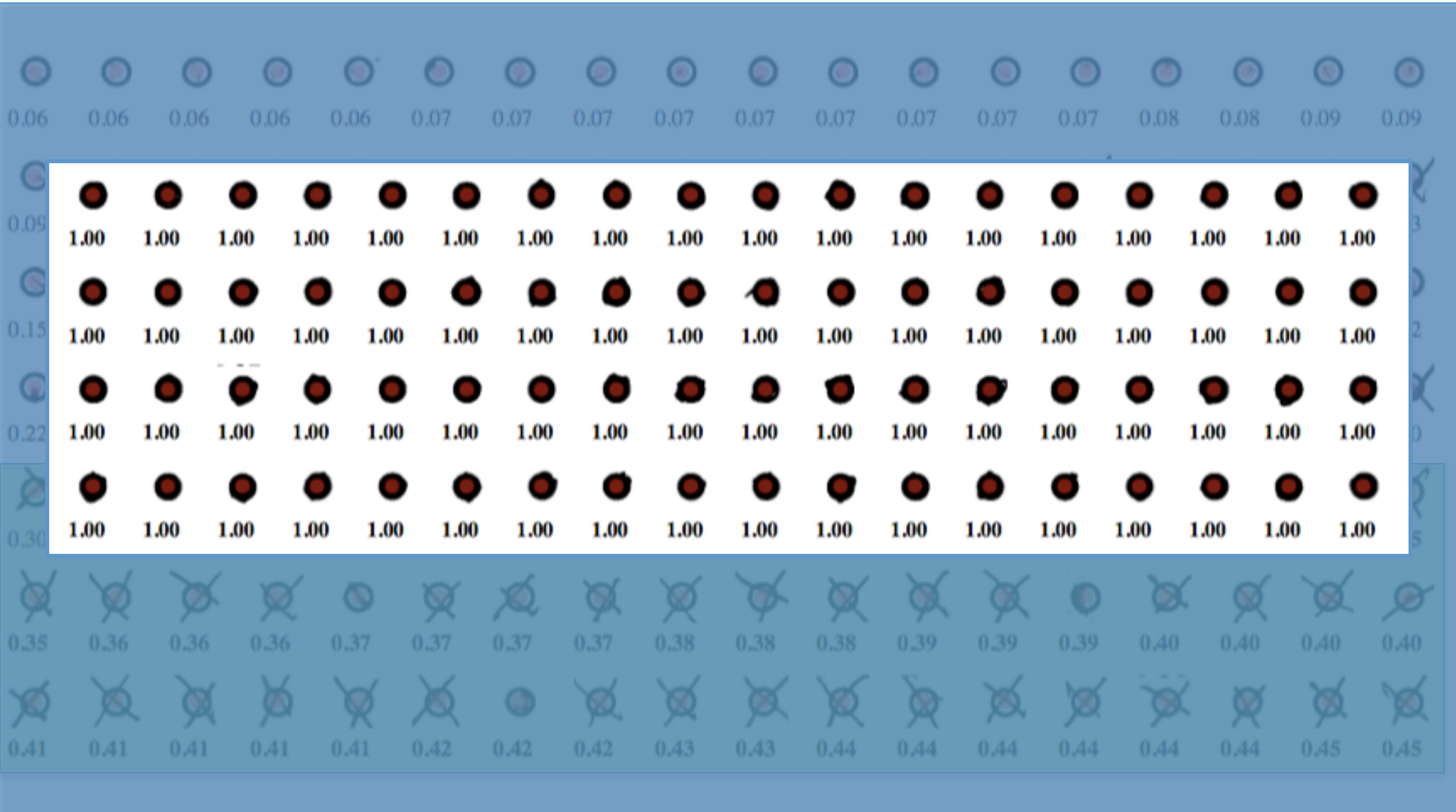


- Average: 55
- Easiest question (Q3, people got on average 97% pts)
- Hardest question (Q9, people got on average 53.4% pts)

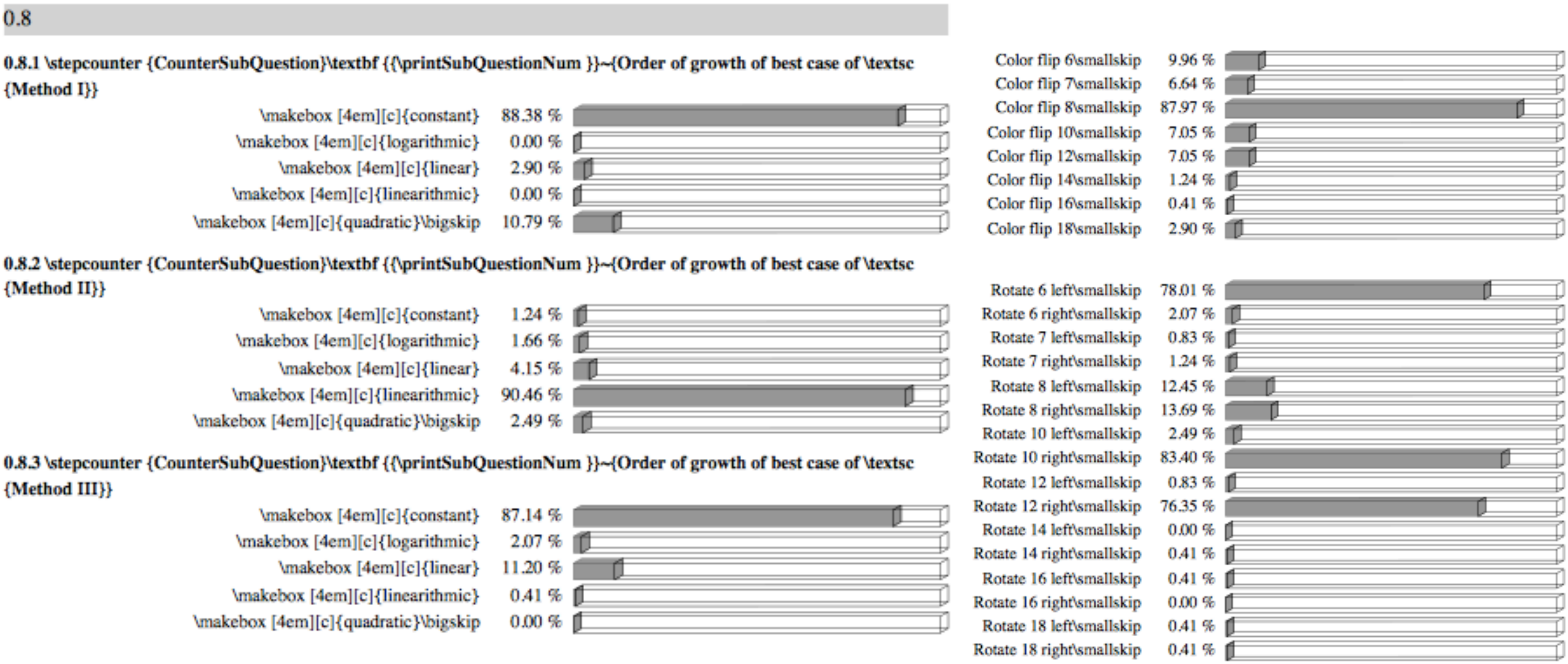
Scanning: Pros and Cons

- 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

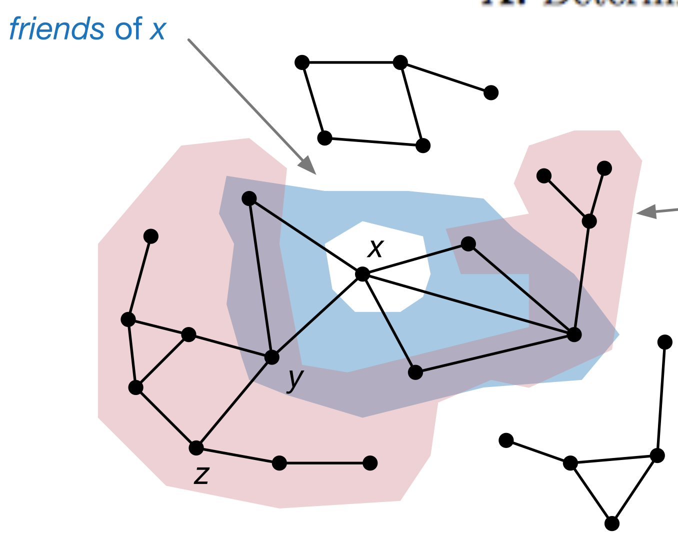


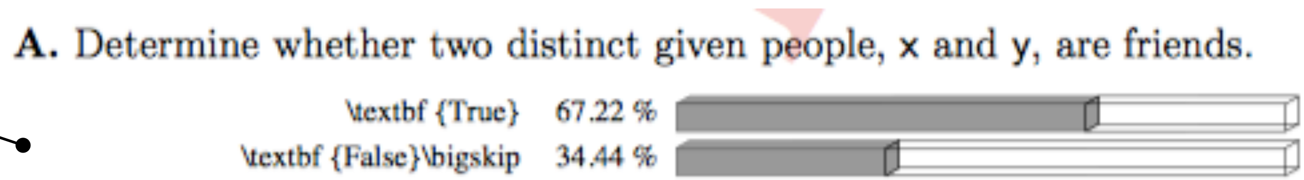
Question-Level Statistics



- 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)

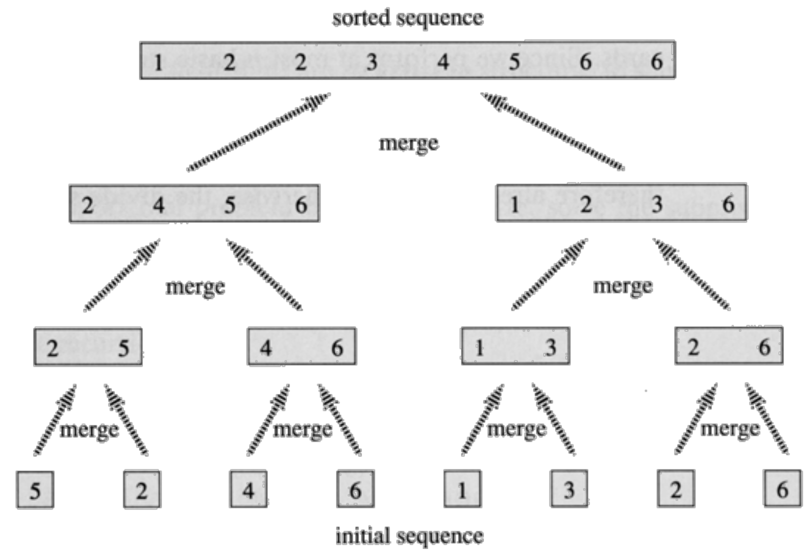
(ii) [3 pts] We now assume it is possible to merge two *sorted* sublists (of any size) in constant time[1].

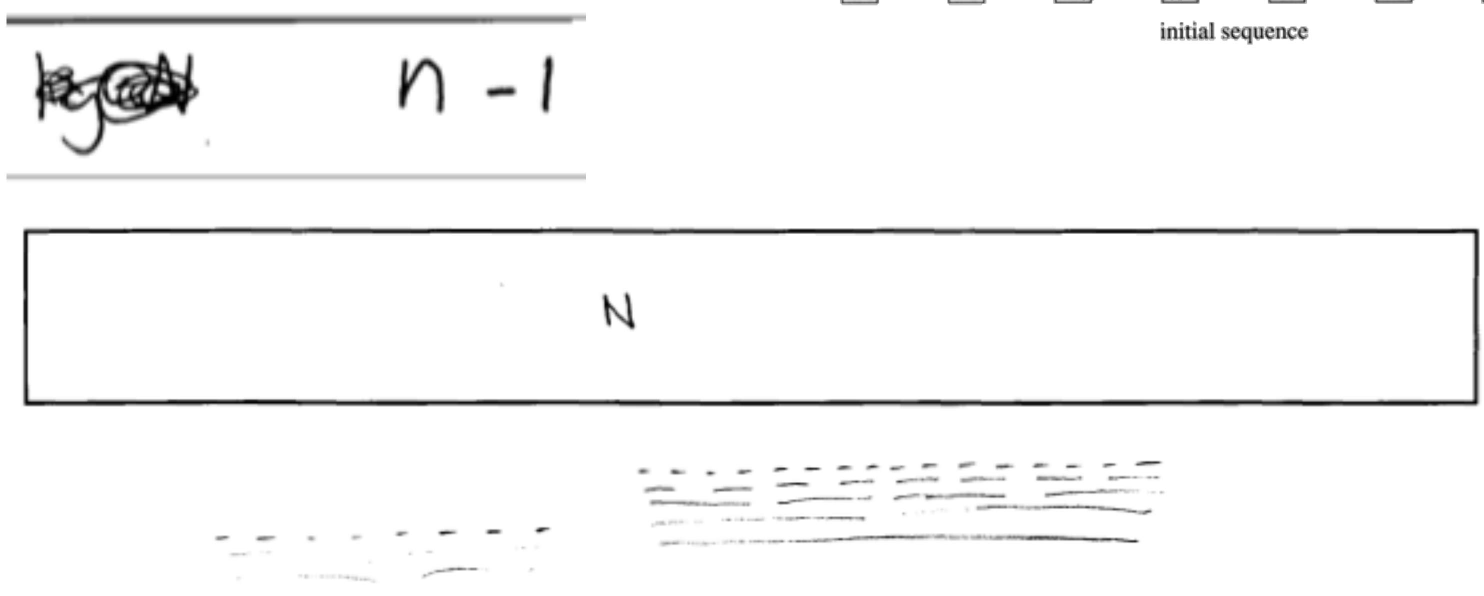
Express the average running time, as a function of N, of the standard textbook 2-way mergesort, modified only to use this new (imaginary) constant-time merging algorithm.

- 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





Q8. Binary Trees (1)

Average success rate: 60.08%

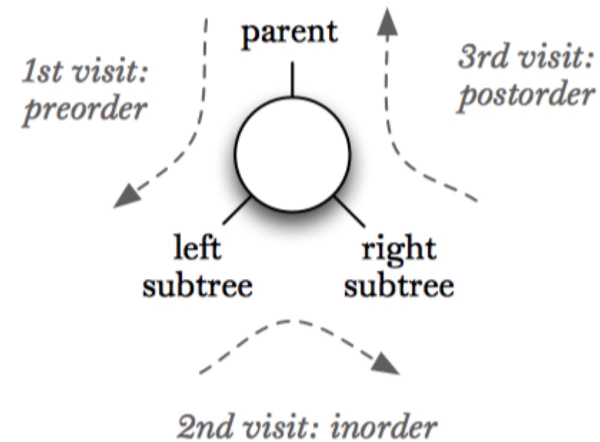
Q8. Binary Trees and BSTs (5 points).

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?

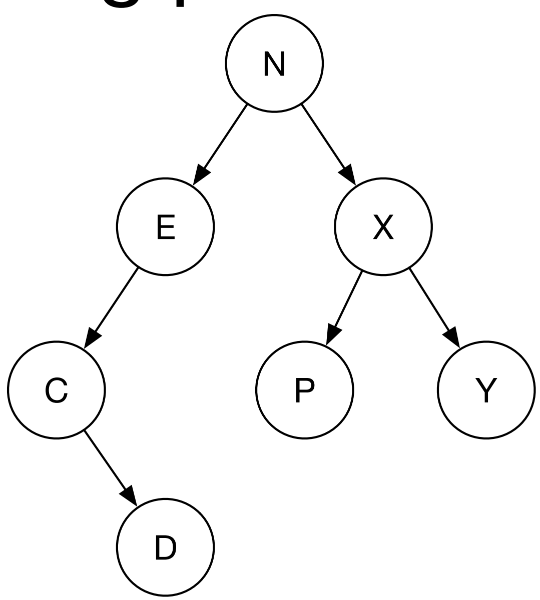
- do you know to distinguish between binary tree and BST?
- do you know what is a tree traversal?
- do you know what is pre-order, in-order?
- do you know what is post-order?

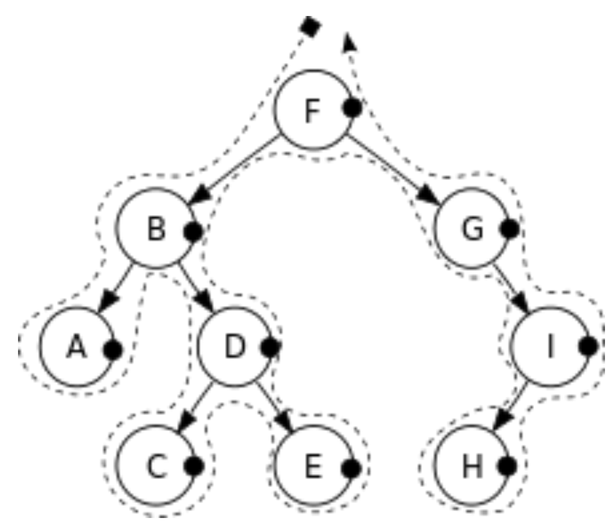
Q8. Binary Trees (2)

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

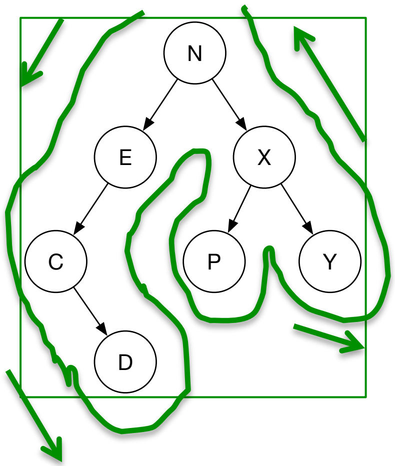


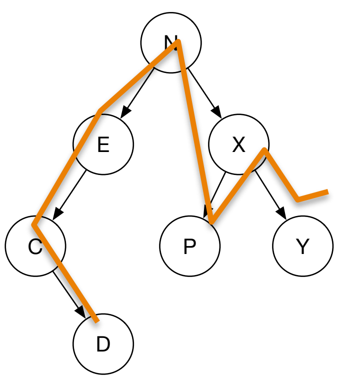
- Starting point is last of post-order: *root*

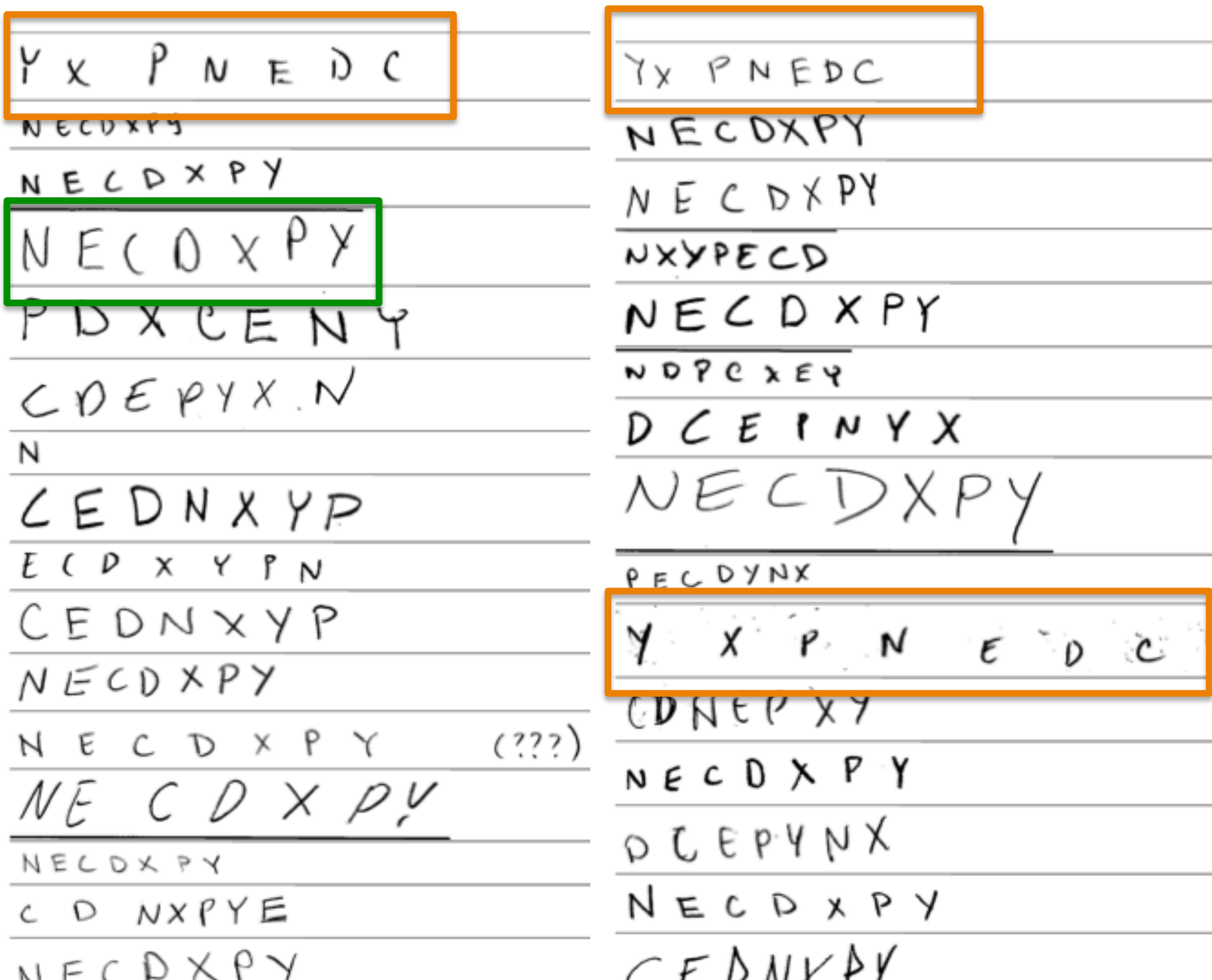




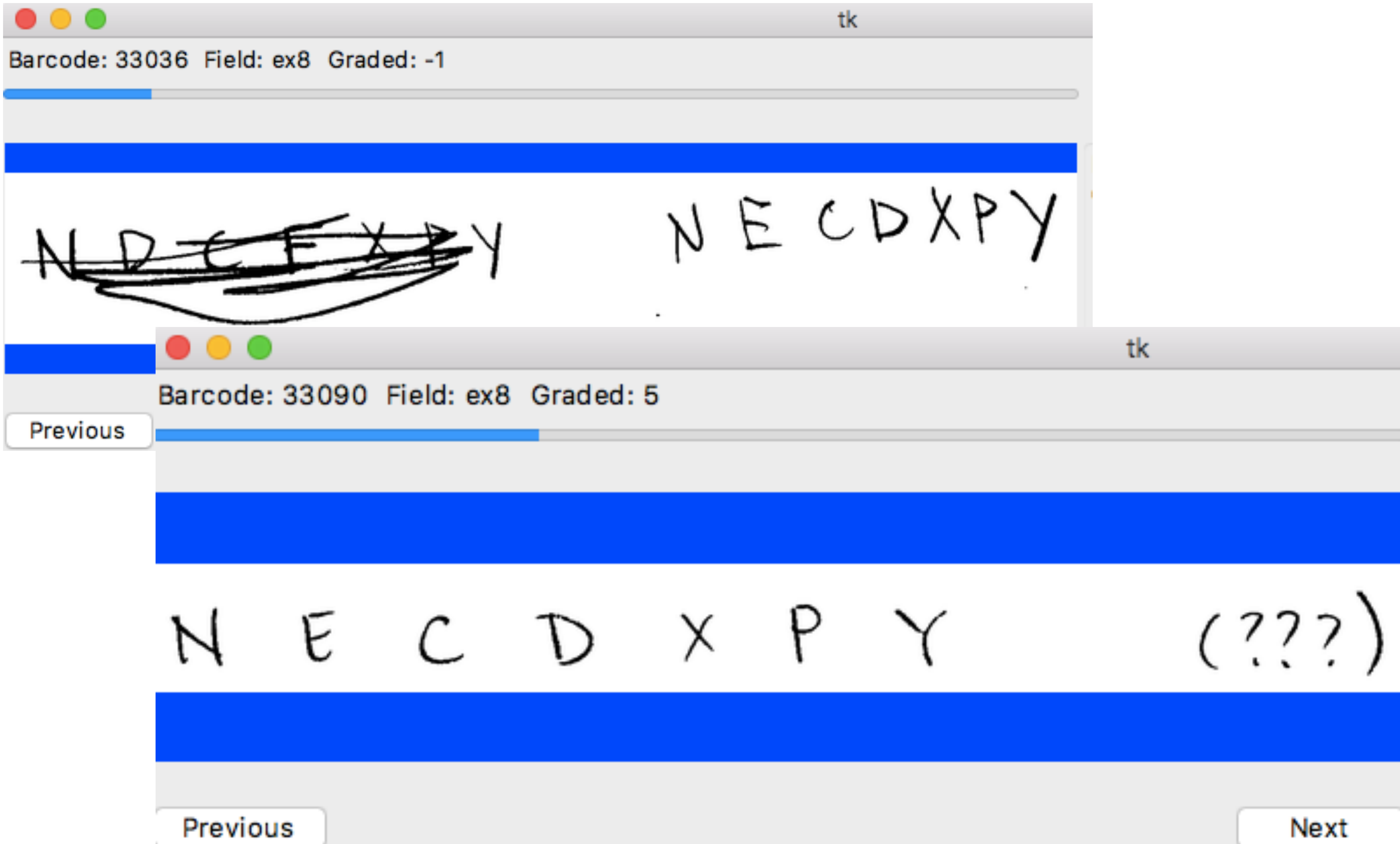
Q8. Binary Trees (3)



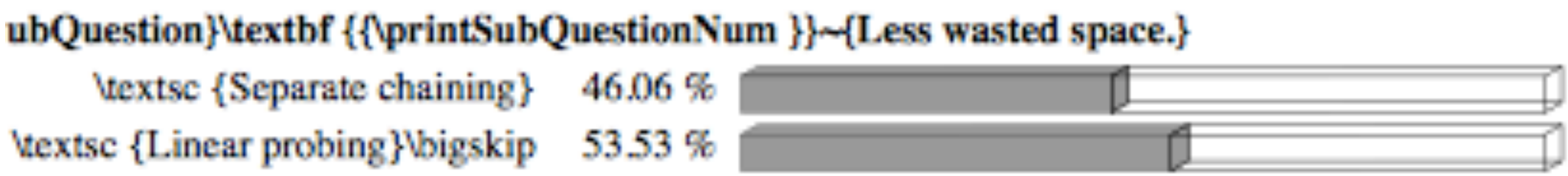




Q8. Binary Trees (4)



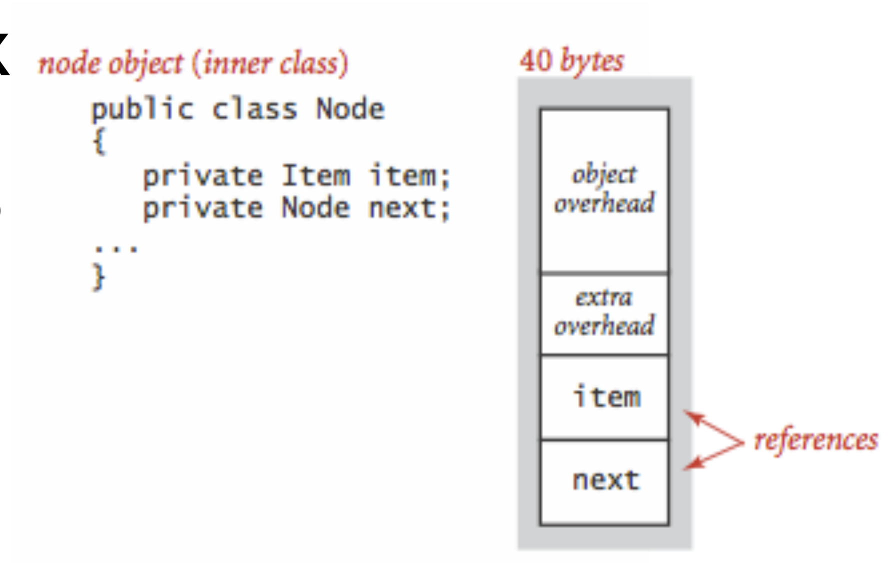
Q10. Hash Functions (1)



- 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 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;
}
```

ST.size() or ST.max()

```java
// 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);

    }
    Polynomial p = new Polynomial(c);
    return p;


}
```

```java
public Polynomial add(Polynomial that) {
    int max = Math.max(this.coefficients.max(), that.coefficients.max());
    double[] result = new double[max + 1];
    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));
        }
        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;
}
```