Quiz 3 — Data Struct. & More (T. I/21–22)

Directions:

- This exam is "paper-based." Answer all the questions in the on-screen editor provided.
- No consultation with other people is permitted. But feel free to use your notes, books, and the Internet. You can only use them in reading mode—do *not* ask for help, etc. You are also allowed to write code and run it.
- At all time, the proctor must be able to see you, your workspace, and your screen.
- You can chat with the instructors via the built-in chat.
- This quiz is worth a total of 35 points, but we'll grade out of 30. Anything above 30 is extra credit. You have 70 minutes. Good luck!

Problem 1: Basic Facts & Techniques (8 points)

(i) **(3 points)** For each of the following algorithms from lecture, indicate its best-case running time and worst-case running time for input of size n in terms of the *tightest* big-O.

	Best Case	Worst Case
`isSorted`	O(n)	O(n)
Quicksort that always picks the first element as the pivot	O(n)	O(n^2)
One `link` operation in the disjoint set data structure that uses lazy linking with height control (i.e., joining the smaller group into the larger one)	O(n)	O(n)

(ii) **(5 points)** Suppose $f(n)$ is $\Theta(n^2)$ and $g(n)$ is $\Theta(n^3)$. Give a mathematical proof using either the limit definition or the for-all-there-exist definition that $h(n) = (n^5 + n) \cdot f(n) + n^3 \cdot g(n)$ is $O(n^7)$.

$f(n) \in \Theta(n^2) \rightarrow \lim_{n\to\infty} \frac{f(n)}{n^2} > 0$

$h(n) \in O(n^7)$ if $\lim_{n\to\infty} \frac{h(n)}{n^7} < \infty$

$\lim_{n\to\infty} \frac{(n^5+n)f(n)+n^3 g(n)}{n^7} = \lim_{n\to\infty}$

$\lim_{n\to\infty} \frac{n^5+n}{n^7} \underbrace{\lim_{n\to\infty} \frac{f(n)}{n^2}}_{c} > 0 + \lim_{n\to\infty} \frac{n^3}{n^7} \underbrace{\lim_{n\to\infty} \frac{g(n)}{n^3}}_{c} > 0$

$\lim_{n\to\infty} \frac{1}{\infty} + \lim_{n\to\infty} 0$

$0 + \frac{0}{\infty} = 0$ O(

Problem 2: Running Time Analysis (15 points)

- **Carefully** analyze each of the following snippets and give the tightest possible big-O for its running time as a function of n.
- Optionally, justify your answer very briefly—no more than three short sentences.
- Partial credit will be given to correct answers that aren't tight but aren't outrageous.

(i)

```
int puzzle0(int[] data) {
    int n = data.length, answer = 0, unknow_val=0;   O(1) Constant
    for (int i=0;i<=(n*n)-1;i++) {
        if (i<n) {answer += data[i];}
        else { unknow_val+=1;}
    }
    return answer-unknow_val;
}
```

Iteration 1   i = 0
Item 2   i = 1
3   i = 2
⋮
k   i = k−1

$k-1 = (n*n)-1$
$k = n^2$
$O(n^2)$

$= O(1) + O(n^2)$
$= O(n^2)$ #

(ii)

```
int puzzle1(int n) {
    int acc = 0;   O(1)
    for (int i=n;i>0;i/=2) {
        int j = 0;   O(1)
        while (j < i) {
            acc++;
            j++;
        }
    }
    return acc;
}
```

I_1 $i = n$
I_2 $i = \frac{n}{2}$
I_3 $i = \frac{n}{4}$
⋮
I_k $i = \frac{n}{2^{k-1}}$

$\frac{n}{2^{k-1}} = 1$
$\log n = k-1$
$k = \log n + 1$
$O(\log n)$

I_1 $i=0$
I_2 $i=$

(iii)
```java
void puzzle2(int[] data) {
    int n = data.length, p = data[0];
    int i = 0, j = n-1;
    while (i <= j) {
        while (i < n && data[i] < p) { i++; }
        while (j >= 0 && data[j] > p) { j--; }
        if (i<=j) {
            swap(data, i, j); // O(1)-time swap data[i] and data[j]
            i++;  j--;
        }
    }
}
```

$O(1)$
$O(1)$
$O(n)$
$(O(n)+O(n))\ O(n) = O(n^2)$
I_1 $j=n-1$
I_2 $j=n-2$
I_3 $j=n-3$
$\vdots$
I_k $j=n-k$
$n-k=0$
$k=n$
$O(n)$

Further Directions: The snippets below are recursive. Write a recurrence and indicate the final big-O.

(iv)
```java
double puzzle3(double[] a, int b, int c){
    if(b >= c) return a[b];
    int d = (b+c)/2;
    double m1 = puzzle3(a,b,d);
    double m2 = puzzle3(a,d+1, c);
    if(m1>m2) return m1;
    else return m2;
}
```

$\to O(1)$
$\to O(1)$
$\to T(\frac{n}{2})$
$\to T(\frac{n}{2})$
$\to O(1)$
$\to O(1)$

$2T(\frac{n}{2})+O(1)=O(n)$

Common Recurrences memorize

We list a couple common recurrences, assuming T(0) and T(1) are constant:

- $T(n) = T(n/2) + O(1)$ solves to $O(\log n)$.
- $T(n) = T(n/2) + O(n)$ solves to $O(n)$.
- $T(n) = 2T(n/2) + O(1)$ solves to $O(n)$.
- $T(n) = 2T(n/2) + O(\log n)$ solves to $O(n)$.
- $T(n) = 2T(n/2) + O(n)$ solves to $O(n \log n)$.
- $T(n) = 2T(n/2) + O(n^2)$ solves to $O(n^2)$.
- $T(n) = T(n-1) + O(1)$ solves to $O(n)$.
- $T(n) = T(n-1) + O(n)$ solves to $O(n^2)$.

$n\log n$ $2T(n/2)+O(n)$ ✓
$\log n$ $T(n/2)+O(1)$ ✓
n $T(n/2)+O(n)$ ✓
n^2 $2T(n/2)+O(n^2)$ ✓
n $T(n-1)+O(1)$ ✓
n^2 $T(n-1)+O(n)$ ✓
n $2T(n/2)+O(\log n)$ ✓
n $2T(n/2)+O(1)$ ✓

(v)
```java
int puzzle4(int n, int a) {
    if (n==0) return a;
    int m = n/2;
    int t = puzzle4(n/2, a + m*m*3);
    if (n%2==0) return t;
    else return 2*n + t - 1;
}
```

$\to O(1)$
$\to O(1)$
$\to T(\frac{n}{2})$

$T(\frac{n}{2}) + O(1) = O(\log n)$

Predicate puzzle4 (n,a)
Base Case puzzle4 $(0,2) = 2 = 2+0^2$ TRUE
Inductive Step puzzle4 (n,a) is true $= a+n^2$
WTS puzzle4 $(n+1,a+1) = (a+1)+(n+1)^2$
$= (a+1)+(n^2+2n+1) = a+2+n^2+2n$
$= a+n^2+2n+2$
$=$
$=$

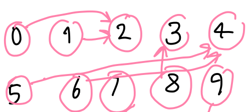
Problem 3: Correctness (5 points)

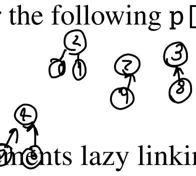
The function `puzzle4` above does compute something interesting. Prove using (strong) induction that for $n, a \in \mathbb{Z}$ with $n \geqslant 0$, `puzzle4`(n, a) returns $a + n^2$. You must clearly write down the predicate you are proving and show the steps.
(*Hint:* The identity $(x + y)^2 = x^2 + 2xy + y^2$ will be useful. Also, remember that in Java if n is odd, `n/2` is equal to $(n - 1)/2$.)

Problem 4: Disjoint Sets (7 points)

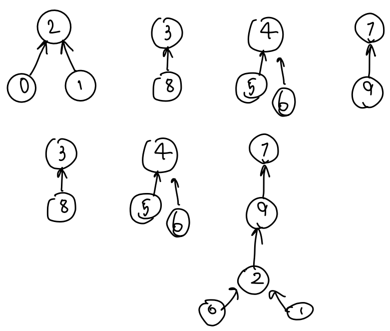
(i) **(4 points)** Draw a visualization of the disjoint-set structure as we did in class for the following `p[]` array.

i	0	1	2	3	4	5	6	7	8	9
p[i]	2	2	2	3	4	4	4	7	3	7





(ii) **(3 points)** Suppose `link(i, j)` is the method as discussed in class that implements lazy linking with height (depth) control (i.e., point small into large). Draw a visualization after `link(1,9)` is called on the disjoint-sets data structure with the `p[]` array above. If you have heard of path compression, note that it does *without* path compression.



link (1,9)