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`	$O(1)$	$O(n)$
Quicksort that always picks the first element as the pivot	$O(1)$	$O(n^2)$
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)	$O(1)$	$O(\log n)$

(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 $O(n^7)$.

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

→ constant

→ loop runs $n^2 - 1$ times

– constant

– constant

$O(1)$ + $O(1)$

$O(n^2)$

– constant

$T(n) = O(n^2)$

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

$O(1)$

halved

$O(\log n)$

$O(1)$

$O(n)$

$(O(1))$

$T(n) = O(n \log n)$

(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--;
        }
    }
}
```

$O(1)$
$O(1)$
$O(n)$
$O(n)$
$O(n)$
$O(1)$

$T(n) = O(n^2)$

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;
}
```

$O(1)$
$O(1)$
$O(1)$
$O(1)$
$T(n/2)$
$T(n/2)$

$T(n) = 2T(n/2) + O(1)$

$= O(\log n)$

(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;
}
```

$O(1)$
$O(1)$
$T(n/2)$
$T(n/2) + O(n)$

$T(n) = T(n/2) + O(1)$

$= O(\log n)$

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 this *without* path compression.

1.2) $h(n) \in O(n^7)$ if $\lim_{n\to\infty} h(n) \leq C \cdot n^7$, where C is a constant.

Given $f(n) \in \Theta(n^2)$ and $g(n) \in \Theta(n^3)$, we have

$$\begin{array}{l} c_1 \cdot n^2 \leq f(n) \leq c_2 \cdot n^2 \\ c_3 \cdot n^3 \leq g(n) \leq c_4 \cdot n^3 , \end{array} \quad \text{for all } n \geq n_0 \text{, where}$$

$c_1, c_2, c_3,$ and $c_4 \geq 0$.

- $h(n) = (n^5 + n) \cdot f(n) + n^3 \cdot g(n)$

$$\leq (n^5 + n) \cdot c_2 \cdot n^2 + n^3 \cdot c_4 \cdot n^3$$

$$= (n^7 + \underbrace{\cancel{n^3}}_{\text{dominated by } n^7}) \cdot c_2 + n^6 \cdot c_4$$

- For $n \geq 1$, $c_2 \cdot n^7$ is the most dominant term as $c_2 \cdot n^7 \geq c_4 n^6$, regardless of constants as $n \to \infty$.

We can say:

$$h(n) \leq (c_2 + c_4) n^7.$$

Our constant $C = c_2 + c_4$. So $h(n) \leq C \cdot n^7$ for $n \geq 1$.

Thus, $h(n) \in O(n^7)$.

3) Predicate: $P(n) \equiv \text{puzzle4}(n,a) \rightarrow a + n^2$

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

Base case: $P(0) \equiv \text{puzzle4}(0,a) \rightarrow a + 0 \rightarrow a$ $\quad$ $P(0)$ is true.

Inductive step: Assume that $\underbrace{\text{puzzle4}(k,a) \rightarrow a + k^2}_{IH.}$ for $0 \le k \le n$.

Show that this works for $n+1$.

__Case 1: $n+1$ is even__

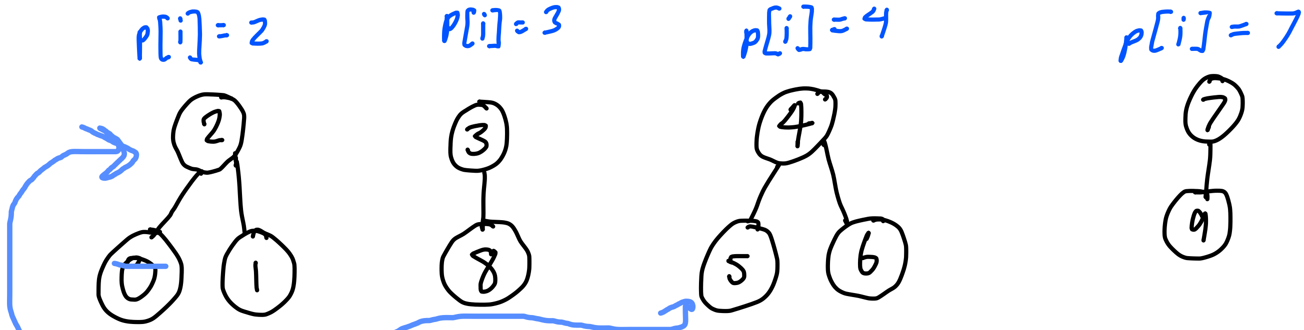
$$\text{puzzle4}(n+1,a) \rightarrow \text{puzzle4}\left(\frac{n+1}{2}, a + 3\left(\frac{n+1}{2}\right)^2\right)$$
$$\xrightarrow{IH} a + 3\left(\frac{n+1}{2}\right)^2 + \left(\frac{n+1}{2}\right)^2$$
$$\longrightarrow a + 4\left(\frac{n+1}{2}\right)^2$$
$$\longrightarrow a + (n+1)^2$$

__Case 2: $n+1$ is odd__ $\quad \left(\left\lfloor \frac{n+1}{2} \right\rfloor = \frac{n}{2}\right)$

$$\text{puzzle4}(n+1,a) \longrightarrow 2(n+1) + \text{puzzle4}\left(\frac{n}{2}, a + 3\left(\frac{n}{2}\right)^2\right) - 1$$
$$\xrightarrow{IH} 2(n+1) + \left(a + 3\left(\frac{n}{2}\right)^2 + \left(\frac{n}{2}\right)^2\right) - 1$$
$$\longrightarrow (a + n^2) + 2n + 1$$
$$\longrightarrow a + (n^2 + 2n + 1)$$
$$\longrightarrow a + (n+1)^2$$

By mathematical induction, $\text{puzzle4}(n,a) \longrightarrow a + n^2$

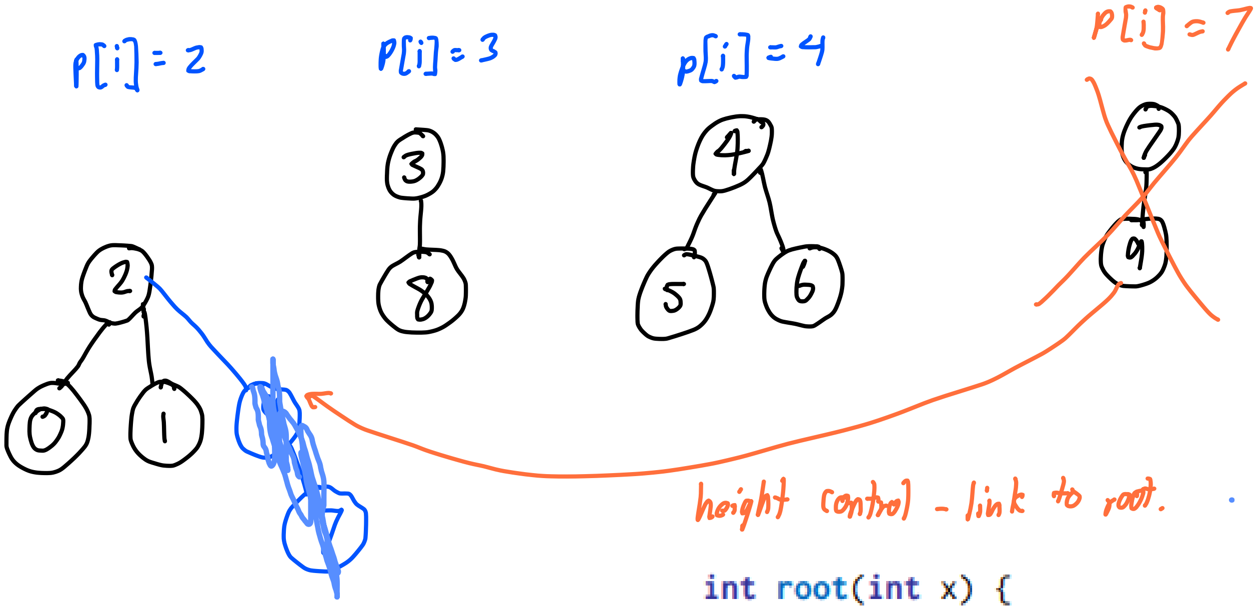
4.1)



(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

4.2 link (1,9)




```
boolean link(int x, int y) {
    p[root(x)] = root(y);
}
```

p[2] = 7

```
int root(int x) {
    while (p[x] != x) {
        x = p[x];
    }
    return x;
}
```

writing a method that identifies the root of a given element, like so:

```
int root(int x) {
```