25

43/100

EC504 A1 Advanced Data Structures, Fall 2015
Midterm Exam

Name Nicholas Mosella

1. True False **(20 Points)** (8)
Answer True or False to each of the questions below. Answer true only if it is always true as stated, with no additional assumptions. No explanation is needed.

F **a.** An algorithm that runs in worst-case $\Theta(n^3)$ time takes longer, on the same input, than an algorithm with worst-case $\Theta(\log n)$ time.
True

b. $\sqrt[3]{n}$ is o(n). True $n > n^{1/3}$

c. $2^{\log n}$ is $\Omega(\log n)$. True $\log n < 2^{\log n} = n$
n

d. $2^{\log n}$ is $O(n^2)$. $n^2 > n$ True
n

F **e.** $\sum_{k=1}^{n} \frac{1}{2^k}$ is $\Theta(n)$. $\Rightarrow \sum_{k=1}^{n} \left(\frac{1}{2}\right)^k = \frac{1/2^{n+1} - 1}{1/2 - 1} = \frac{}{-1/2}$ True

f. Any comparison-based sorting algorithm will take $\Theta(n \log n)$ on a sorted array of size n.
False

T **g.** It is possible to find the smallest log(n) elements of an unsorted array of size n in O(n) time.
False

F **h.** Quicksort of a sorted list of n elements has a worst-case run time of O(n log n).
True

F **i.** A binary search tree of height lg(n) will always have at least n elements.
True

T **j.** There are 14 different binary search trees that contain only the numbers 4, 5, 6, 7.
False

Scanned by CamScanner

19

2. Recurrences **(24 Points)**
For each of the following functions, provide:
- A recurrence T(n) describing the worst-case running time of the function with respect to n as provided (i.e. without optimizations).
- The tightest asymptotic upper and lower bound you can for T(n). Explain your work.

a.
```
static public int A(int n) {
        if (n == 0) return 1;
        else if (n >= A(n/2)) return A(n/2);
        else return 1;
}
```

$\Theta(1)$

$T(n) = 2T(n/2) + \Theta(1)$

Master:

$n^{\log_b a} = n^{\log_2 2} = n > cf(n)$

$n > c$

$\Rightarrow f(n) = \Theta(n)$

$\Rightarrow n^{\log_2 2 - \varepsilon} > c \quad c > 1$

$\Rightarrow$ True for $c \neq \log_2 n - \varepsilon$

* Using master method we get a tight bound of $\Theta(n)$

6

b.
```
static public int B(int n) {
        int sum;
        if (n <= 2) return 1;
        else {
                for (int ii=0; ii<4; ii++)
                        sum += B(n-1);
        }
        return sum;
}
```

$\sum_{0}^{4} 4 \cdot 10$

$0 \to 4$

$4(T_{n-1})$

$T(n) = 4T(n-1) + \cancel{c}\,\Theta(1)$

3

$\Theta(n \log n)$

c.
```
static public C(int n){
        if (n == 0) return 1;
        else {
                int sum = C(n/2) + C(n/2);
                for (int ii=1; ii != n; ii++)
                        for (int jj=1; jj != n; jj++)
                                sum += ii + jj
                return sum * C(n/2) * C(n/2);
        }
}
```

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

$\to$

$\to + 2T(\frac{n}{2})$

$T(n) = T(\frac{n}{2}) + T(\frac{n}{2}) + 2T(\frac{n}{2}) + c\,\frac{n}{n!}$?

4

$\Rightarrow \Theta(n^2 \log n)$

d.
```
static public int D(int n) {
        if (n == 0) return 1;
        if (D(n/2) <= D(n/4))
                return D(Math.sqrt(n))+ D(Math.sqrt(n));
        else
                return D(n/2);
}
```

$\Rightarrow T(n/2) + T(\frac{n}{4})$

$\to T(\sqrt{n}) + T(\sqrt{n})$

$\Rightarrow T(\frac{n}{2})$

6

$T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + T(2\sqrt{n}) - \Theta(n^2 \log n)$

5

3. More Recurrences **(16 Points)**
Give the tightest upper and lower bounds you can for the following recurrences.

a. $T(n) = 8T\left(\frac{n}{2}\right) + 3$

1

$n^{\log_b a} = n^{\log_8 2} = \frac{\log_2 2}{\log_2 8} =$

$\boxed{\Theta(n^{\log_2 8})}$

$T(n)$

$8 \Rightarrow T(n/2)$

8^1

height $= \log_2 n$ $\quad 8^2$

8^i

8^i = nodes at level i

$\sum_{i=0}^{\lg n} 8^i \left(\frac{n}{2^i}\right) = n \sum_{i=0}^{\lg n} \left(\frac{8}{2}\right)^i = n \sum_{i=0}^{\lg n} 4^i = \frac{4^{\lg n+1} - 1}{4-1} \quad \frac{4^{\lg n+1} - 1}{3}$

$\Rightarrow n / \frac{8^{\lg_2 n}}{2^{\lg n}} \quad 8^{\lg_2 n}$

$\Rightarrow n^{\log_2 8}$

b. $T(n) = T(n-1) + n$

0

$T(n+1) = T(n) + n + 1$

$T(n+1) - T(n) = n + 1$

$\Rightarrow (E-1)T(n) = E(n)$

$\Rightarrow (E-1)(E-1) \Rightarrow$ annilates function $\Rightarrow (E-1)^2 = (\alpha n + \beta) 1^n = \boxed{\Theta(n)}$

c. $T(n) = 2T\left(\frac{n}{2}\right) + \frac{1}{n}$ $\Rightarrow 2T\left(\frac{n}{2}\right) + n^{-1}$

4

$n^{\log_a 2-\varepsilon} = n > \frac{c1}{n} \Rightarrow n^{\log_2 2-\varepsilon}$

$\Rightarrow n^2 > C \Rightarrow \boxed{\Theta(n)}$ for $c > 0$

0 **d.** $T(n) = \frac{T(n-1)^2}{T(n-2)}, T(0) = 1, T(1) = \pi$ $\Rightarrow T(n-1)^2$

$T(n-2)\, T(n) = T(n-1)^2 \Rightarrow \boxed{\Theta(n^2)}$

$\frac{0}{15}$

4. Short Answer **(15 points)**
A cyclic permutation of an array involves moving the first element to the end of the array. A cyclically sorted array is one that can be cyclically permuted (possibly several times) to produce a sorted array. For example, <6, 8, 10, 1, 2, 3> is a cyclically sorted array because it can be cyclically permuted 3 times to produce a sorted array:
<6, 8, 10, 1, 2, 3> → <8, 10, 1, 2, 3, 6> → <10, 1, 2, 3, 4, 6, 8,> → <1, 2, 3, 4, 6, 8, 10>

For each of the following, provide an answer and a short explanation:

a. What is the fastest worst-case asymptotic run time needed to find the minimum element of an arbitrary cyclically sorted array?

$\frac{n!}{(n-k)!}$

- The fastest worst-case runtime would be $\Theta(n)$ since the worse case would need to go through the entire array & move the first element to the end of the array & shift the elements to the right

X

b. What is the fastest worst-case asymptotic run time needed to find the median element of an arbitrary cyclic sorted array?

- The runtime would be $\Theta(n^{\log_2 2})$ since it would need to sort the whole array & then find the median which would be $T(\frac{n}{2}) + \Theta(n)$,

$n^{\log_2 2} < n$

X

c. What is the fastest worst-case asymptotic run time needed to sort the elements of an arbitrary cyclically sorted array?

- The fastest worst case runtime would $\Theta(1)$ since it would be the case where the array is already sorted

X

11/25

5. Short answer **(25 points)**
For each of the following, provide an answer and a short explanation:

a. Determine the expected time to insert a uniformly random integer between 0 and n into a max heap containing the numbers 0 through n/2.

- The runtime would be $\Theta\left(\frac{\log(n)}{2}\right)$ since would would need to only bubble up and check half the tree if necessary

-5

b. Determine the worst case time required to sort a list of $\log n$ distinct elements in the range $\{1,\ldots,n^2\}$ with counting sort.

- It would be $\Theta(\log n + n^2)$, which would give a worst case runtime of $\Theta(n^2)$ since it would need to go through array & have an unique index of log n element & then go through the range $1 \to n^2$

c. Determine the worst case time required to sort a list of $\log n$ distinct elements in the range $\{1,\ldots,n^2\}$ with insertion sort.

- The worst case would be $\Theta((\log n)^2)$

d. Determine the best case time required to insert n^2 elements into an empty max heap.

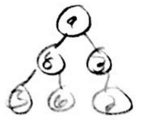
- The best case would $\Theta(1)$, since it is a lazy data structure and will just add all elements to heap w/o sorting

-4

2 3 5 6 8 9

e. If you insert the numbers 9, 8, 3, 5, 6, 2 into an initially empty min heap, what will the heap look like?

- Initially it will violate the min heap property, but would need to bubble & sort heap after



-5

— Midterm Rev —

- Part #1 T/F

d) $2^{\lg n}$ is $O(n^2)$ = $n^{\log_{10} 2}$ ⇒ True

e) $\sum_{k=1}^{n} \frac{1}{2^k}$ is $\Theta(n)$ ⇒ $\sum_{k=1}^{n} \frac{1}{2^k} = \sum_{k=1}^{n} \left(\frac{1}{2}\right)^k \le \sum_{k=0}^{\infty} \left(\frac{1}{2}\right)^k = \frac{1}{1-\frac{1}{2}} = 2$

f) False ⇒ Bubble sort etc ⇒ $\Theta(n)$ = not always case

g) True ⇒ Linear Time Select algo ⇒ can select it using tb

h) False ⇒ $\Theta(n^2)$

i) False ⇒ Bin Not Necessarily Balanced ⇒ so does not always apply

j) True ⇒ 14 diff ⇒ 4, 5, 6, 7 ⇒

⇒ 5 w/ 4 root
⇒ 5 w/ root 7
⇒ 2 w/ root 5
⇒ 2 w/ root 6
14

(2) a) $T(n) = 2T(\frac{n}{2}) + \Theta(1)$ ⇒ Master Case #1 ⇒ $\Theta(1)$? $\Theta(n)$

b) $T(n) = 4T(n-1) + \Theta(1)$ ⇒ Annihilators ⇒ $\Theta(4^n)$

c) $T(n) = 4T(\frac{n}{2}) + n^2$ ⇒ Master Case #2 ⇒ $\Theta(n^{\log_2 4} \log n)$, $\Theta(n^2 \log n)$

d) Hard to find bounds gave 6 points ⇒ Prof couldn't find bound

(3) a) $T(n) = 8T(\frac{n}{2}) + 3$ ⇒ Master Case #1 ⇒ $\Theta(n^3)$

b) $T(n) = T(n-1) + n$ ⇒ Iteration or Annihilators (not sure, ignore) harder ⇒ $T(n-2) + n - 1 + n \ldots$ ⇒ $T(1) + \sum_{i=1}^{n} i$ ⇒ $\Theta(n^2)$

c) Master Case #1 ⇒ $\Theta(n^{\log_2 2})$ ⇒ $\Theta(n)$

d) Take log of both sides, get: $\log_2 T(n) = 2\log_2 T(n-1) - \log_2 T(n-2)$

⇒ Substitute $s(n) = \log_2 T(n)$

$s(n) = 2s(n-1) - s(n-2)$ ⇒ Annihilators ⇒ $s(n) = (\alpha n + \beta) \cdot 1^n$

⇒ $T(n) = 2^{\alpha n} \cdot 2^{\beta}$ ⇒ use initial conditions: $\beta = 0$, $\alpha = \log_2 \pi$

⇒ $\Theta(\pi^n)$

4) Challenging Question
⇒ Can move in both or one direction, doesn't really matter

a) $\Theta(\log n)$ ⇒ could do a modified Binary Search ⇒ looking for breaking point of a sorted arrays
ie) 3rd array < 10, 1, 2, 3, 6, 8 >

b) $\Theta(\log n)$ ⇒ if find median know how many elements in array

c) $\Theta(n)$ ⇒ don't have to run sorting algo since already sorted

5) a) Heap has $\Theta(n/2)$ elements; having a 50% chance of going to insert as root, having 50% that it could go into diff levels of heap

b) Counting sort Runtime $\Theta(n+m)$ ⇒ $\Theta(\log n + n^2)$ ⇒ $\Theta(n^2)$
n ↑ = $\log n$, m ↑ = n^2

c) Insertion $\Theta(n^2)$ worst ⇒ if have $\log n$ elements then $\Theta((\log n)^2)$

d) Best case to insert one element ⇒ $\Theta(1)$ ⇒ n^2 times ⇒ $\Theta(n^2)$

e) Min Heap ⇒