CS 5633 Analysis of Algorithms – Fall 23

Exam 2

NAME: Md Sohanur Rahman

- This exam is closed-book and closed-notes, and electronic devices such as calculators or computers are not allowed. You are allowed to use a cheat sheet (half a single-sided letter paper).
- Please try to write legibly – if I cannot read it you may not get credit.
- **Do not waste time** – if you cannot solve a question immediately, skip it and return to it later.

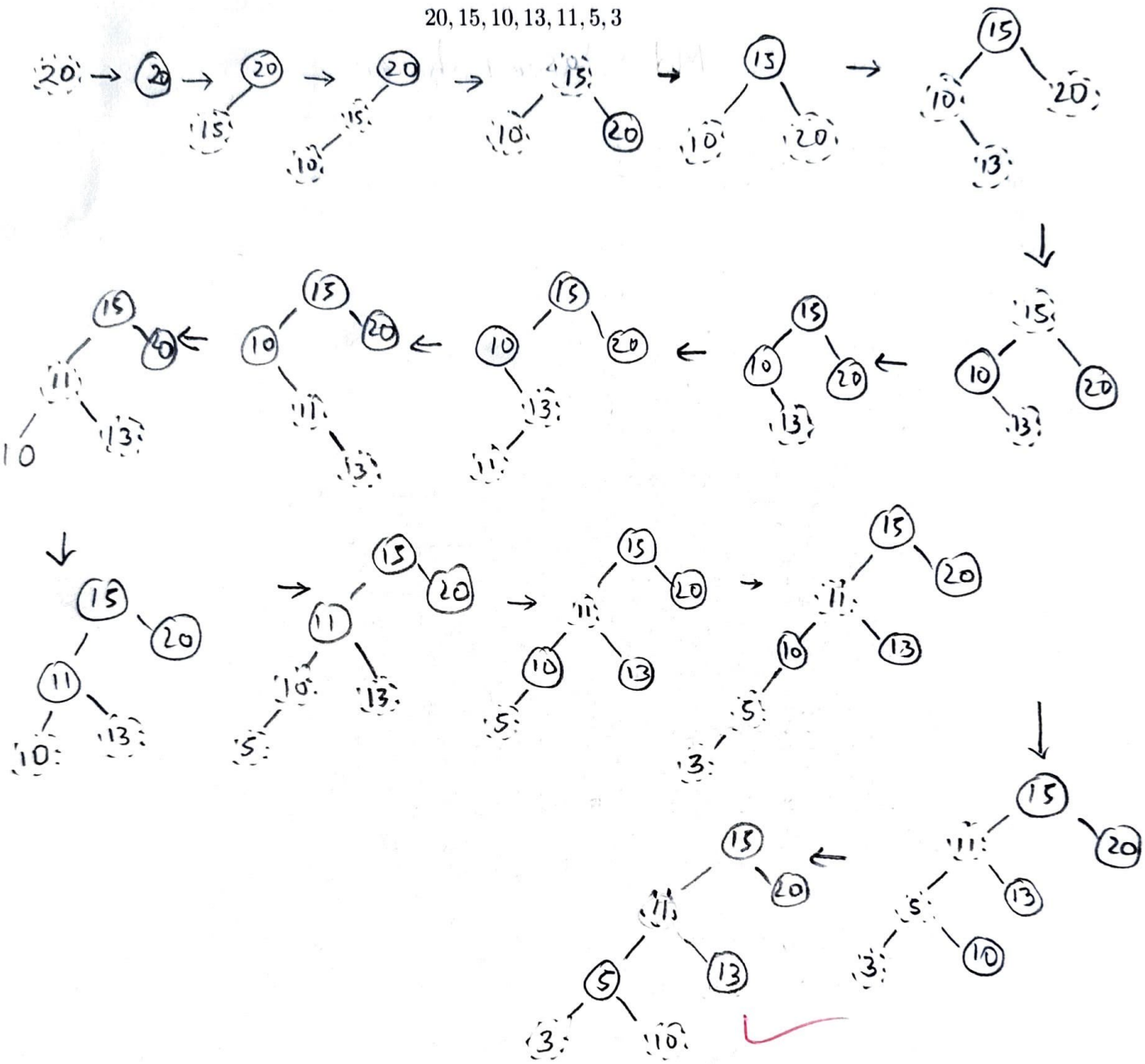
1) Red-Black Trees		24
2) B-Trees		23
3) Sorting Algorithms		18
4) Dynamic Programming		35
		100

24

1 Red-Black Trees (24 Points)

Insert the following numbers into an initially empty Red-Black tree, showing the tree after each insertion. You can use solid nodes for black and dotted nodes for red if you wish.

20, 15, 10, 13, 11, 5, 3

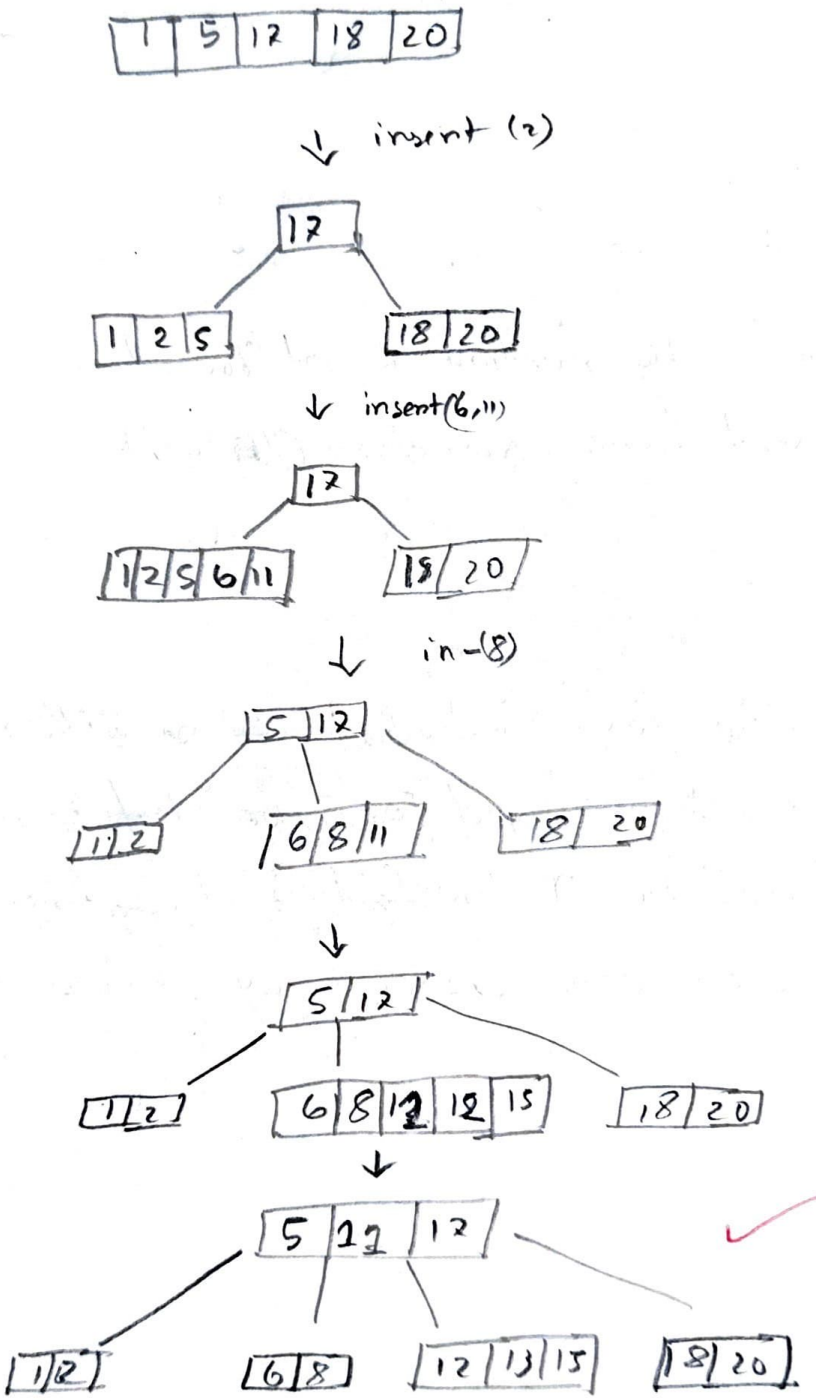


23

2 B-Trees (23 Points)

Insert the following numbers into an initially empty B-Tree with $t = 3$. Show the tree before/after every *split*. I.e., you do not need to show it after every insertion if it isn't about to create a split. I will accept preemptive splitting or no premptive splitting.

$17, 20, 1, 5, 18, 2, 6, 11, 8, 12, 15, 13$ $2t-1=5$



18

3 Sorting Algorithms (18 Points)

Suppose we want to choose a sorting algorithm with the best worst-case running time possible for the assumed input. What sorting algorithm would you use? What would the running time of the algorithm be? Justify your answer.

1. An array of n grade point averages of the form $X.XX$.

Radix sort can be used with $O(n)$ run time.

✓

2. An array of n numbers where each number is the running time of some program measured in milliseconds.

As the range of the runtime is not bound we should use merge sort here, Run time $O(n \log n)$

✓

3. An array of n *abc123* ids from UTSA.

Here we can use radix sort, ~~but the~~ with run time $O(n+k)$, where k is ranged $[0,9]$ for last 3 numbers. and k is ranged $[a \ldots z]$ for first 3 characters. As k is constant for each case we can say runtime $O(n)$

✓

33

4 Dynamic Programming (35 Points)

Suppose we consider a problem like the toll booth problem from the homework where we have a long road with various locations of where we can place a toll booth. We are given an array T of positive numbers such that $T[i]$ is the value we obtain from opening a toll booth at the ith possible location. In the homework we were given a distance array X such that $X[i]$ was how many miles from the beginning of the road the ith toll booth is which we needed to make sure we did not pick toll booths that were too close to each other. Suppose now instead of enforcing a 10 mile minimum between the toll booths we select, we instead enforce that we cannot pick three consecutive toll booth locations. So for example, if I pick toll booth 1 and 2, then I would not be able to pick toll booth 3. Subject to this constraint, we wish to maximize the sum of the values of the selected toll booths we pick.

1. What is the running time of a brute force strategy that considers every subset of toll booths, checks to see if it is feasible, and returns the best feasible solution?

To check every ~~feasible~~ possible combination takes $O(2^n)$ runtime
Checking feasible solution with that will take $O(n \cdot 2^n)$

2. Let $c[i]$ denote the optimal value when considering only the first i toll booth locations. Consider the following example with five possible locations: $T = [10, 25, 20, 5, 30]$. What is $c[1], c[2], c[3], c[4]$, and $c[5]$ for this example (I'm using 1-indexing here)? There is one more part on the next page.

$T = [10, 25, 20, 5, 30]$

$c[1] = 10$

$c[2] = 10 + 25 = 35$

$c[3] = 25 + 20 = 45$

$c[4] = 45$

$c[5] = 30 + c[3] = 75$
(45)

3. Give a recursive definition for $c[i]$. You do not need to give an algorithm.

$$c[i] = \begin{cases} T_i & \text{if } i = 1 \quad \text{// add the first one} \\ T_1 + T_2 & \text{if } i = 2 \quad \text{// add first two value} \\ \max\{\max(c[i-2]+T_{i-1},\ c[i-2]+T_i,\ c[i-3]+T_{i-2}+T_i),\ c[i-1]\} & \text{o/w} \end{cases}$$

First 3 max ... to find the maximum value, if we take the current value. Then check with the last optimal solution which is greater will be the optimal for current value.

$c[7] = ?$

$\max($ $35+5$ $\rightarrow c_{i-2}+T_{i-1}$,
$45+30$ $\rightarrow c_{i-2}+T_i$,
$35+5+30)$ $\rightarrow c_{i-3}+T_{i-2}+T_i$

$= 75$