CS 5633 Analysis of Algorithms – Fall 18

Exam 2

NAME: Solution

- 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.

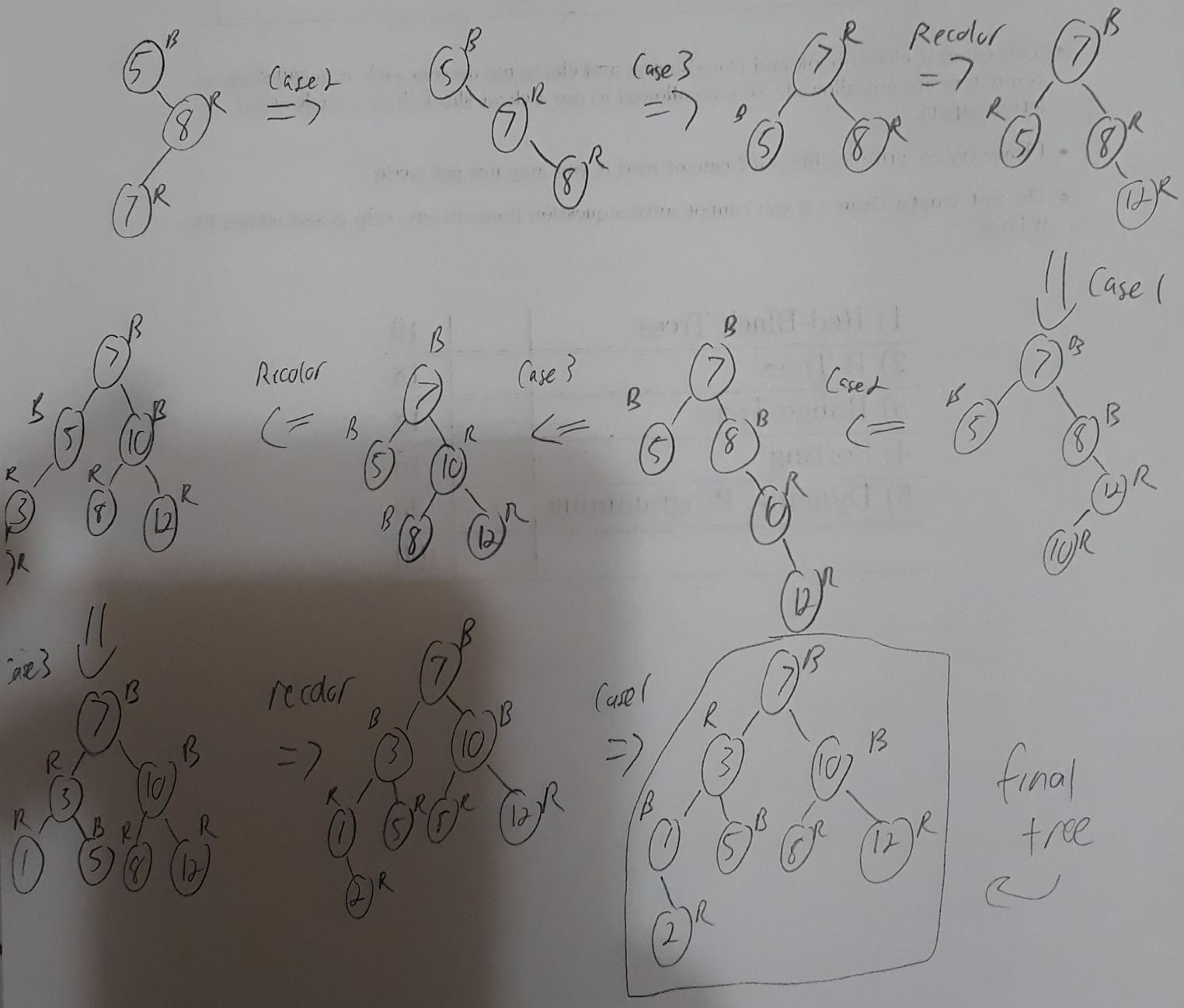
1) Red-Black Trees		19
2) B-Trees		18
3) Range Trees		18
4) Sorting		15
5) Dynamic Programming		30
		100

1 Red-Black Trees (19 Points)

Consider making the following sequence of inserts into an initially-empty red-black tree.

$$5, 8, 7, 12, 10, 3, 1, 2$$

Show the tree before and after each rotation or recoloring (i.e. you do not need to draw it after each insertion if the insertion does not force a rotation or recoloring).



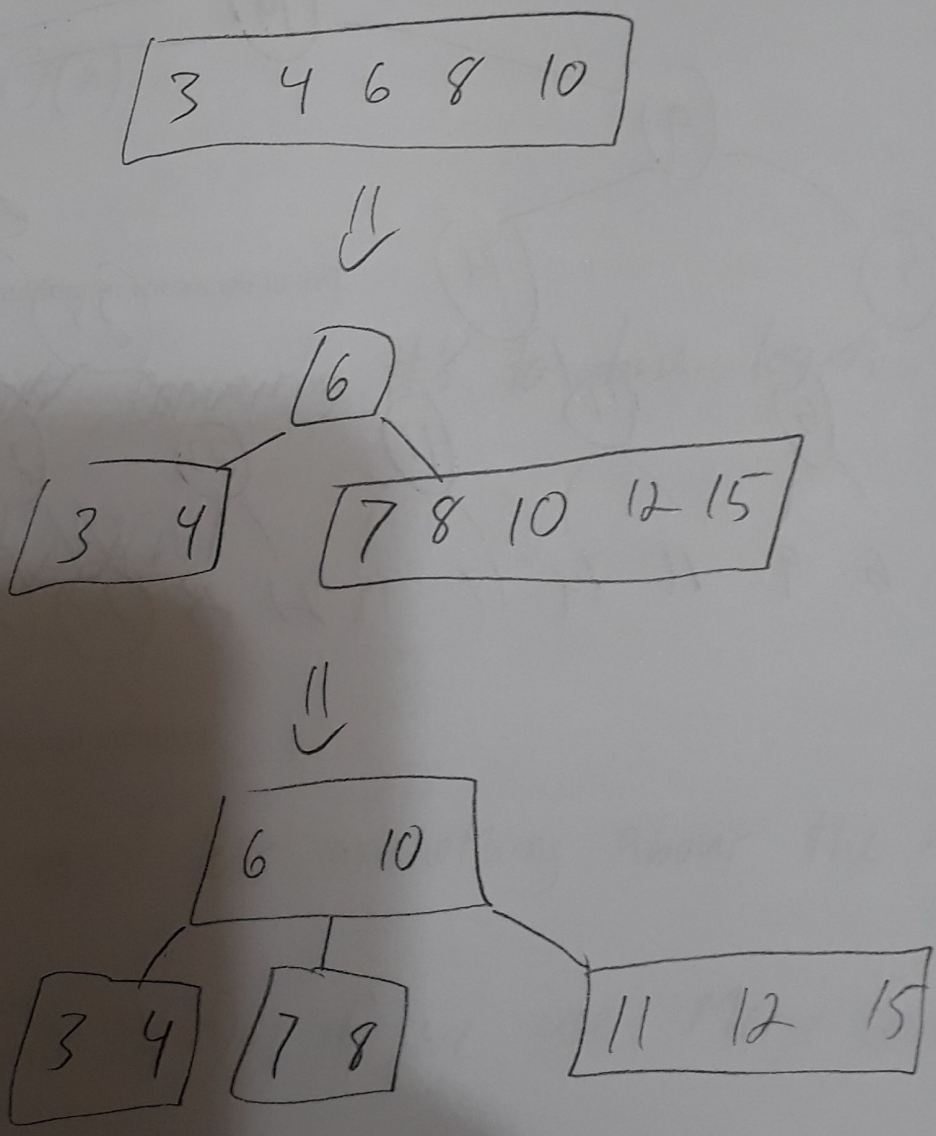
2 B-Trees (18 Points)

Consider making the following sequence of inserts into an initially-empty B-tree with $t = 3$.

$$3, 6, 10, 4, 8, 12, 15, 7, 11$$

Show the tree before and after each split (i.e. you do not need to draw it after each insertion if the insertion does not force a split).

Full at $2t - 1 = 5$

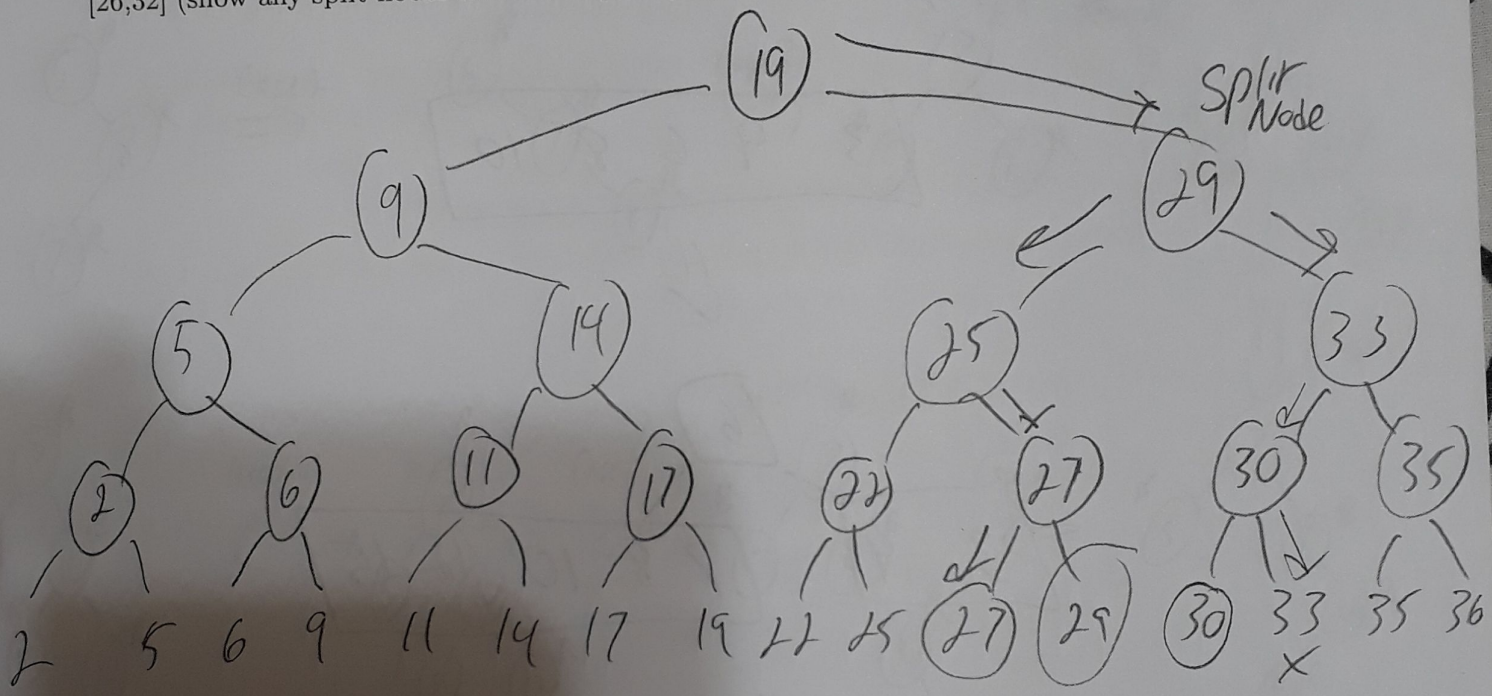


3 Range Trees (18 Points)

Consider a range tree on the 16 following 1D points:

$$2, 5, 6, 9, 11, 14, 17, 19, 22, 25, 27, 29, 30, 33, 35, 36$$

1. Draw the 1D range tree.
2. Show how the search works on your tree when searching for the points in the interval [26,32] (show any split nodes and search paths).



4 Sorting Runtimes (15 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 UTSA banner IDs.

Radix sort. Constant calls to Counting Sort with a $k=10$.

$\Rightarrow O(n)$

2. An array of n numbers in the range $[0..n^2]$.

Radix sort converting #'s to base $\log n$

$\Rightarrow O(n)$

3. An array of n rational numbers.

Rational #'s tells us nothing about the range.

$\Rightarrow O(n \log n)$ if we use Merge Sort

5 Dynamic Programming (30 Points)

In the United States, we have coins that are worth the following denominations: 1, 5, 10, and 25. Suppose we want to make change for n cents using the minimum number of coins. One can see that an optimal solution can be computed using the following simple greedy algorithm: repeatedly pick the largest coin that is less than or equal to the remaining change. For example, if I want to make change for 40 cents, then I first use a 25 cent coin (15 cents remain), then I use a 10 cent coin (5 cents remain), and finally I use a 5 cent coin (0 cents remain). This provides an optimal solution for the US coin denominations; however, it does **not** provide an optimal solution for different coin denominations. For example if we added a 20 cent coin to the US coin denominations, then the greedy algorithm would fail when making change for 40 cents. The greedy algorithm would compute the same solution as before, but an optimal solution uses two 20 cent coins.

In this problem we will consider giving a dynamic programming algorithm that will compute the optimal number of coins to use that should work regardless of the denominations. Formally, we have m coin denominations $d_1 < d_2 < d_3 < \cdots < d_m$ where the smallest coin d_1 is 1 cent. We are then given an integer n denoting the number of cents for which we want to make change, and we want to compute the minimum number of coins needed to make change for n cents.

1. Consider a brute force algorithm that considers every possible way of adding up coins to n cents and returns the best solution. What is the running time of this algorithm?

We can make multiple copies of each coin. Certainly we would not need more than $\frac{n}{d_i}$ copies of coin d_i. Considering every way of chosing those coins $\approx n^m$

↑ at most n copies of each coin over m different coins,

2. Let $a[i]$ denote the optimal number of coins to make i cents. Suppose we have 3 coins of value 1, 4, and 5. What is $a[1], a[2], a[3], a[4], a[5], a[6], a[7]$, and $a[8]$?

$a[1] = 1$

$a[2] = 2$

$a[3] = 3$

$a[4] = 1$

$a[5] = 1$

$a[6] = 2$

$a[7] = 3$

$a[8] = 2$ (two 4 cent coins).

3. Give a recursive definition for $a[i]$. Do not forget the base case.

$$a[i] = \begin{cases} 0 & \text{if } i = 0 \\ 1 & \text{if } i = d_j \text{ for some } j \\ \min_{j : d_j \le i} \{ a[i - d_j] + 1 \} & \text{o.w.} \end{cases}$$

4. Give a dynamic programming algorithm for this problem based on your recursive definition. You can use top-down or bottom-up.

```
a[0] = 0
for j = 1 to m
    a[d_j] = 1;

for (i = 2 to n) {
    if (a[i] hasn't been assigned) {
        min = n;
        for (j = 1 to m) {
            if (d_j ≤ i AND a[i - d_j] < min)
                min = a[i - d_j];
        }
        a[i] = min + 1;
    }
}
Return a[n];
```