CS 5633 Analysis of Algorithms – Spring 14

Exam 2

NAME: Solution

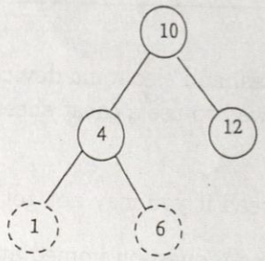
- This exam is closed-book and closed-notes, and electronic devices such as calculators or computers are not allowed. You are allowed to use a cheat sheet (half a single-sided letter paper).
- Please try to write legibly – if I cannot read it you may not get credit.
- **Do not waste time** – if you cannot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

1) Red-Black Trees		15
2) B-Trees		12
3) Range Trees		15
4) Dynamic Programming		18
5) Greedy Algorithms		18
6) Amortized Analysis		18
		96
Free Points	4	4
		100

1 Red-Black Trees (15 Points)

1. Is it possible to have a red-black tree that contains all black nodes. If so, how many vertices are in the tree as a function of its black height h? If a tree is not valid, then which property is violated?

2. Consider the following red-black tree where the solid nodes are black and the dotted nodes are red (nil leaf nodes not shown in this figure):



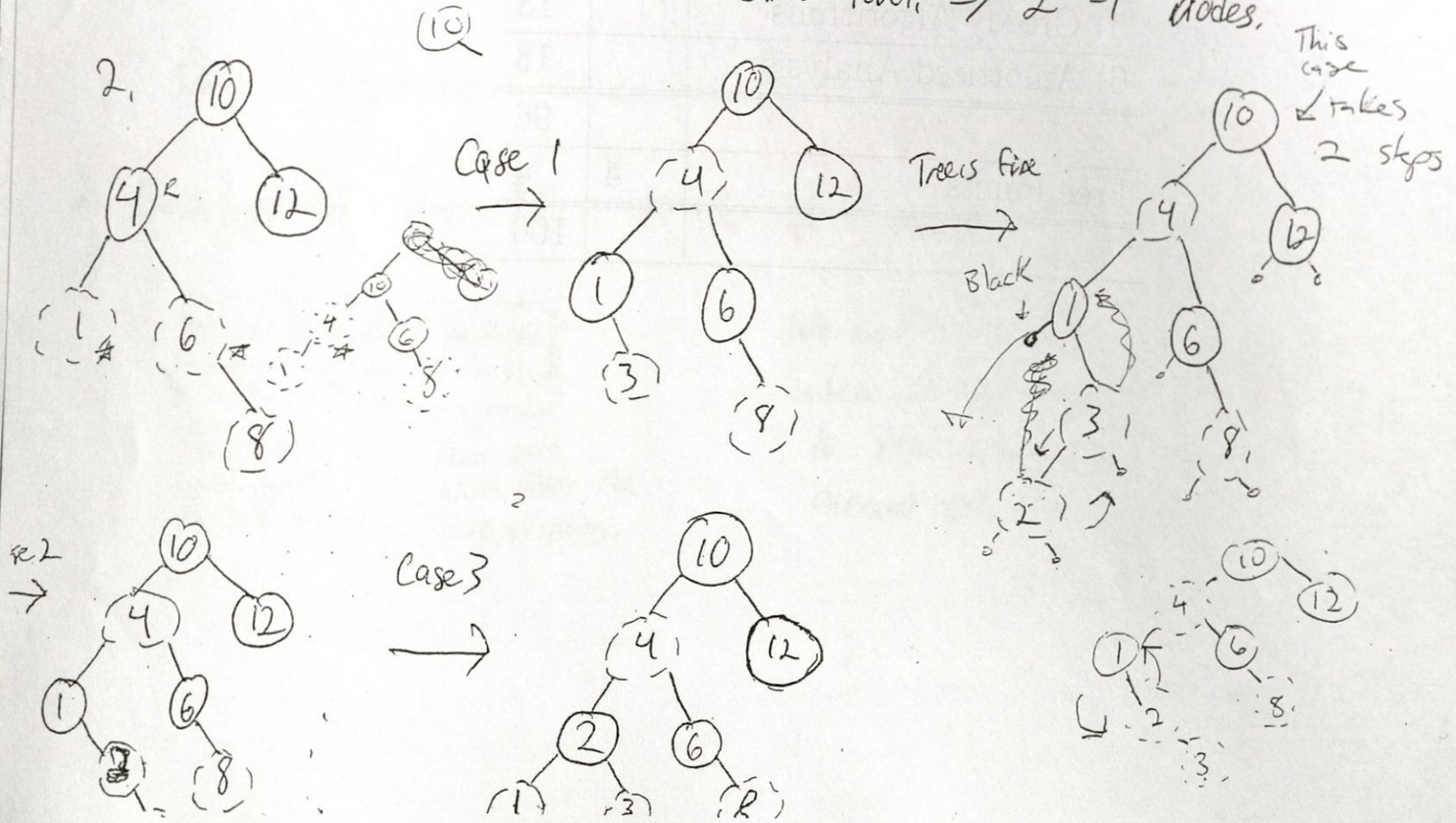
Suppose we insert the following three nodes into the tree in order. Show the resulting tree after each insertion.

(a) Insert 8

(b) Insert 3

(c) Insert 2

1, Yes, if all leaves are at the same level, $\Rightarrow 2^h - 1$ nodes.

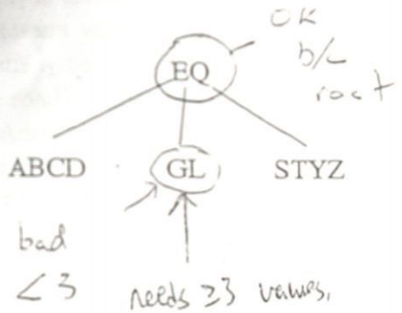


t=4 t=4 2t-1

fewest = 3 most = 7

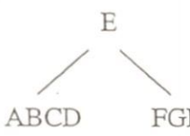
B-Trees (12 Points)

1. Which of the following trees are legal B-trees for $t = 4$? If it is not legal, then give a one or two sentence explanation for why it is not a legal B-tree.



OK b/c root

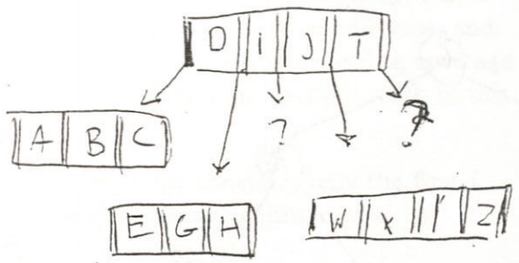
bad < 3 needs ≥ 3 values.



Valid

not enough children.

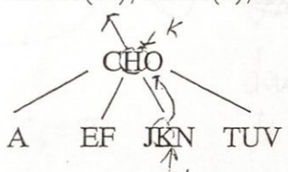
rot 1 ≤ root nodes ≤ 7



because things are added to the root when other nodes split, there can never be empty pointers from the root

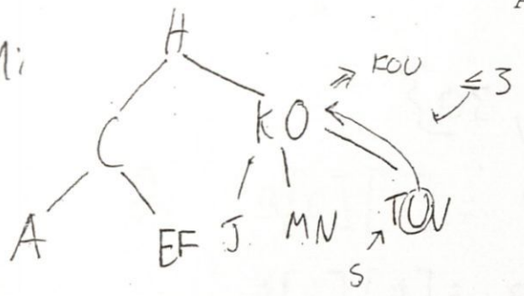
2. Consider the following B-tree with $t = 2$. Show the resulting B-trees after making the following sequence of insertions: Insert(M), Insert(S), Insert(G).

t=2 fewest = 1 most = 3



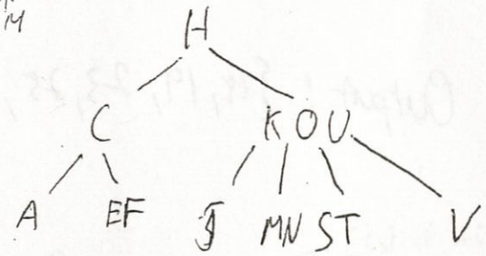
JKMN

M:

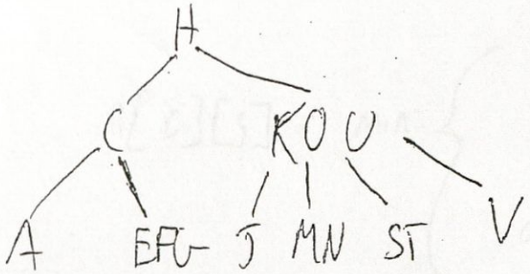


KOU ≤ 3

S:



G:



B trees push nodes up!

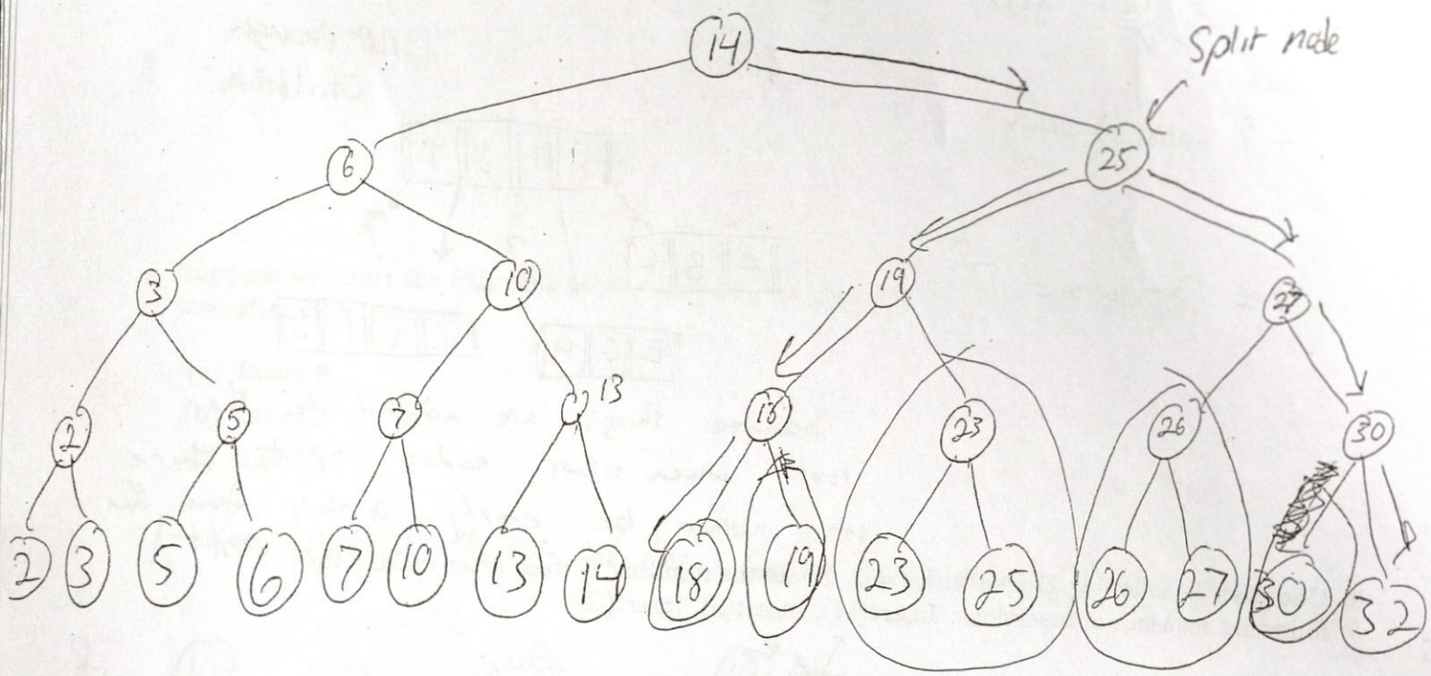
Red black trees add nodes down.

3 Range Trees (15 Points)

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

$$2, 3, 5, 6, 7, 10, 13, 14, 18, 19, 23, 25, 26, 27, 30, 32$$

1. Draw the 1D range tree.
2. Show how the search works on your tree when searching for the points in the interval $[17, 31]$ (show any split nodes and search paths).



Output: {18, 19, 23, 25, 26, 27, 30}

ynamic Programming (18 Points)

sider a problem somewhat similar to the towers problem from the homework. We again ve n houses h_1 to h_n on a straight road which are indexed according to their position on the road (h_1 is the first house on the road, and h_n is the last house on the road). We again need to place towers within 4 miles of each house to provide service, but now there are only m fixed locations l_1 to l_m where we can build towers (as opposed to being able to place them anywhere on the road). Again assume that the locations are indexed according to their position on the road. Additionally the cost of building a tower depends on the location, and now we have a nonnegative cost c_j of building a tower at location l_j. We would like to choose a set of locations to build towers that provides service to all houses at a minimum cost.

1. On the homework problem, a correct greedy algorithm repeatedly identifies the first uncovered house and builds a tower 4 miles past this house. So a natural attempt at a greedy solution for this problem is to repeatedly identify the first uncovered house, and build a tower at the location that is farthest down the road while still providing coverage to this house. Give an example to show that this greedy algorithm does not work in this situation.

2. Let $a[i][j]$ denote the weight of a minimum-weight cover that considers only the first i houses and the first j possible tower locations. Give a recursive definition of $a[i][j]$. Don't forget the base case.

3. Give a bottom-up dynamic programming algorithm based off your recursive definition. What is the running time of your algorithm?

Feel free to use the back of this page if you need more room.

1.

Greedy will take l_2 as it is farther down the road, but clearly it is better to take l_1.

2. $a[0][j] = 0 \quad \forall j$

$a[i][0] = \infty \quad \forall i > 0$

$$a[i][j] = \min \begin{cases} a[i][j-1] & \text{cost if we do not include tower } j \\ a[k][j-1] + c_j & \text{cost if we do include tower } j \text{, where } k \text{ is index of last house that tower } j \text{ does not cover} \end{cases}$$

3.

```
for (j = 0 to m)
    a[0][j] = 0

for (i = 1 to n)
    a[i][0] = ∞.

for (i = 1 to n) {

    for (j = 1 to m) {

        let K denote the index of the last house s.t. K ≤ i and
            location l_j does not cover h_K.

        costToInclude = c_j + a[K][j-1]

        costToLeaveOut = a[i][j-1]

        a[i][j] = min(costToInclude, costToLeaveOut);

    }

}

return a[n][m];
```

Running time $\Theta(n^2)$

5 Greedy Algorithms (18 Points)

Suppose we are given a set $A = \{a_1, \ldots a_n\}$ of n intervals on the real line, where each interval $a_i \in A$ is denoted $[s_i, f_i]$. A subset of the intervals B forms a *tiling* of A if the intervals in B "cover" the intervals of A. That is, for any real number r such that r is in some interval of A, there is an interval in B that contains r.

- Example: Let $a_1 = [1, 10], a_2 = [4, 15], a_3 = [9, 14]$, and $a_4 = [13, 20]$. Then the subset $B_1 = \{a_1, a_3, a_4\}$ forms a tiling as only a_2 is not in B_1 and every $r \in [4, 15]$ is contained in some interval in B_1. The subset $B_2 = \{a_1, a_4\}$ is not a tiling as 12 is in a_2 (and a_3), but is not in either a_1 or a_4.

Give a greedy algorithm that computes a tiling of A with the minimum number of intervals for any set A of n intervals. Briefly argue why your algorithm is correct (5 or 6 sentences should suffice).

$T = \emptyset$.

While (there is a number r in an interval of A that is not covered by T) {

Let x denote the smallest such number.

Let A_x denote the subset of intervals in A that contain x.

Let $I \in A_x$ be the interval in A_x with the largest right endpoint.

$T = T \cup \{I\}$

}

Return T.

Note that some interval in a solution must cover x. Suppose there is an optimal solution that doesn't choose I to cover x, but instead uses I'. We can remove I' from the solution and include I and still have a tiling with the same number of intervals, and thus this solution is

6 Amortized Analysis (18 Points)

Recall the dynamic array data structure that we described in class. Initially it was an array of size 1, and we can insert elements into the array by "appending" it to the end of the array. If an insertion is performed when the array is full, we doubled the size of the array and copied the previous elements into the new array and could then insert the new element into the larger array.
Now suppose we have the same data structure, but instead of doubling the array we triple the size of the array. So the first array will be of size 1, when it fills we get an array of size 3, then 9, then 27, etc.

1. Use an aggregate analysis to get an upper bound on the cost of n insertions.
2. Use the accounting method to get an upper bound on the amortized cost of a single operation.

1.

i	1	2	3	4	5	6	7	8	9	10	11
Size	1	3	3	9	9	9	9	9	9	27	27
c_i	1+0	1+1	1+0	1+3	1+0	1+0	1+0	1+0	1+0	1+9	1+0 ...

$$\sum_{i=1}^{n} c_i \leq n + \sum_{j=1}^{\log_3 n} 3^j \leq n + 2n = 3n$$

Cost of n insertions is $\leq 3n$.

2. Each operation pays \$3. \$1 for now and \$2 for later.

\$0	\$0	\$0	\$2	\$2	\$2	\$2	\$2	\$2

These were added before the last tripling. These were added after the last tripling.

We will have enough money stored when we need to copy elements to new array, and the bank account will never go below 0.