University of Waterloo
CS240R Fall 2017
Solutions to Review Problems

Reminder: Final on Tuesday, December 12 2017

Note: This is a sample of problems designed to help prepare for the final exam. These problems do *not* encompass the entire coverage of the final exam, and should not be used as a reference for what the final exam contains.

True/False

For each statement below, write true or false. Justify three of them.

a) False. The step size for linear probing is always 1, so a second hash function is not needed.

b) True. A run of 2 bits is "compressed" into `010`, which is 3 bits long. Similarly, a run of 4 bits is encoded into a 5-bit string.

c) False. Depending on the distribution of points, the number of nodes can increase exponentially with height (up to $n/2$ nodes), while all of these nodes are visited. An example is if we had $n/2$ pairs of points, with early partitions separating the different pairs ($n/2$ internal nodes), but the points in each pair are close enough that $\Omega(h)$ partitions are needed to separate them. Range search can be as bad as $\Omega(nh)$ then.

d) False. Suffix trees preprocess the text, not the pattern.

e) False. Deletion of an internal node involves swapping with the successor, and the successor might be a 2-node, so there is no height loss on deletion. In fact, the height will only drop if merging occurs at every single level below the root.

f) False. The bubble-up version runs in $O(n \log n)$ time, so the overall runtime of HeapSort is still in $O(n \log n)$ time.

g) False. The structure of the kd-tree only depends on the relative ordering of the x and y coordinates. Spread factor is only relevant in quadtrees.

h) True. Insertion in Cuckoo Hashing can lead to a loop, since each key has only two positions to map to, and so rehashing may be necessary regardless of the load factor.

i) False. Only a single left rotation is required, since this is a right-right imbalance.

j) True. The dictionary for MTF changes as the characters are being encoded/decoded, since entries are moved to the front.

Multiple Choice

Pick the best answer for each question.

1. d) `E` = 7, `L` = 2, `M` = 6, `S` = 8
2. c) 7.
3. a) CCCCCC
4. c) 4
5. b) RadixSort, HeapSort, QuickSort
6. b) The root of the compressed trie always tests the first bit.
7. c) Boyer-Moore
8. I wonder...

Hashing

a) Many possible answers. Here is one: Insert 13, 7, 8, 6.

$$\begin{aligned} h_1(13) &= 6; \\ h_1(7) &= 0; \\ h_1(8) &= 1; \\ h_1(6) &= 6; h_2(6) = 1. \end{aligned}$$

Index	Key
0	7
1	8
2	6
3	(empty)
4	(empty)
5	(empty)
6	13

Table 1: Hashing part (a)

b) Inserting 13, 7, 8, 6 using $h_2(n)$ with linear probing:

$$h_2(13) = 4;$$
$$h_2(7) = 4;$$
$$h_2(8) = 1;$$
$$h_2(6) = 1.$$

Index	Key
0	(empty)
1	8
2	6
3	(empty)
4	13
5	7
6	(empty)

Table 2: Hashing part (b)

The function $(3n \bmod 6) + 1$ is a poor hash function since 6 is a multiple of 3. Specifically $6 = 3 \times 2$, so even numbers hash to $0 + 1$ while odd numbers hash to $3 + 1$. Having only two possible outcomes despite having a larger hash table size is bad. In general, a hash function that involves $(an \bmod b)$ should have a and b co-prime to each other.

Huffman Compression

a) `pswd: lull`

b) In Huffman compression, characters with greater frequency have shorter or equal-length encodings compared to characters with lower frequency. Here, the letter "ℓ" appears three times while "u", only appears once, but the former has a longer code (010) than the latter (11).

If we swapped the codes for ℓ and u, we would get a shorter string, so the original string was not optimal. Therefore, the string and dictionary combination could not have been generated by Huffman.

Rabin-Karp

a) $h(\texttt{TAGCAT}) = 15$.

T	G	C	C	G	A	T	G	T	A	G	C	T	A	G	C	A	T
T	A																
	T																
		T															
								T	A	G	C	A					
										T							
												T	A	G	C	A	T

Table 3: Table for Rabin-Karp problem.

b) Map each character to its weight $w(\texttt{A}) = 1$, $w(\texttt{C}) = 2$, $w(\texttt{G}) = 3$, $w(\texttt{T}) = 4$. Then $h(T[i+1 \ldots i+m]) = h(T[i \ldots i+m-1]) - w(T[i]) + w(T[i+m])$.

c) One possible answer: $T = \texttt{C}^n$, $P = \texttt{C}^{m-2}\texttt{GA}$.

KD-Trees

a) See Figure 1.

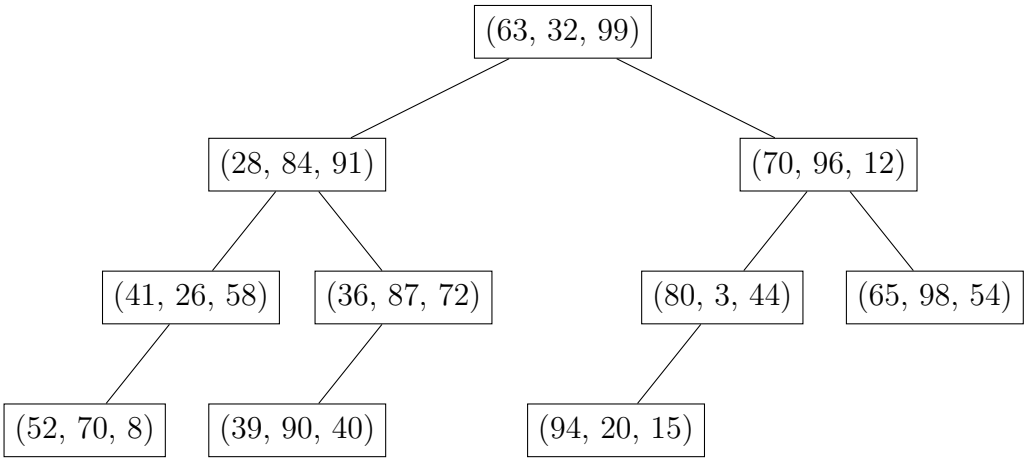


Figure 1: 3D kd-tree

b) See Figure 2. Only (65, 98, 54) is reported.

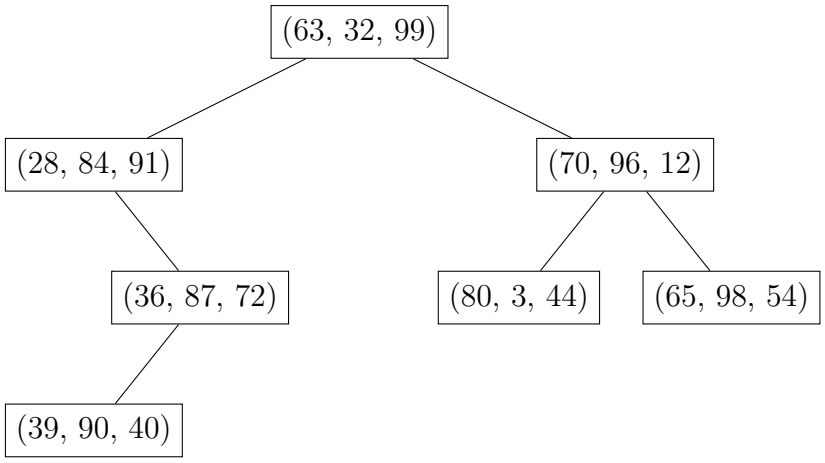


Figure 2: Range Search on the 3D kd-tree

Range Trees

a) The y-BSTs are:

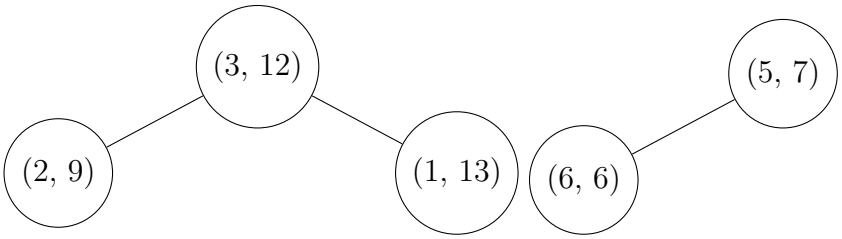


Figure 3: y-BSTs for $(2, 9)$ [left] and $(5, 7)$ [right]

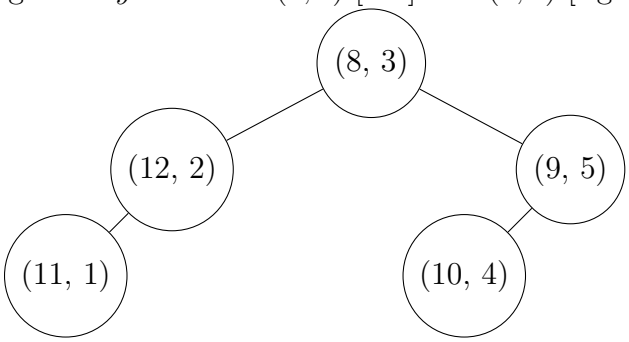


Figure 4: y-BST for $(9, 5)$

b) (It's okay if all boundary nodes are listed together)

- Common Boundary Nodes: $(7, 10)$.
- Left Boundary Nodes: $(4, 8), (2, 9), (1, 13)$.
- Right Boundary Nodes: $(9, 5), (11, 1), (10, 4)$.
- Inside Nodes: $(5, 7), (6, 6), (3, 12), (8, 3)$.
- Outside Nodes: $(12, 2)$.

Order Notation

a) Many examples. A simple one is $f(n) = n + 1$, $g(n) = n$. We can take $c = 1$, since $n \leq n + 1$ for all $n \geq 0$.

b) This claim is false. Many counterexamples are applicable, but a simple one is $f(n) = n - 1$ and $g(n) = n$. Both functions map positive integers n to non-negative reals, and $g(n) \in O(f(n))$. But if $g(n) \in Onion(f(n))$, then $n \leq c(n-1)$ for all $n \geq 0$, which implies $1 \leq 0$ for $n = 1$, which is impossible for all values of c. Therefore, the statement is false.

(For most applications of order notations, however, the functions $f(n)$ and $g(n)$ would map positive integers to *positive* reals, in which case the claim would have been true.)

Run-Length Encoding

a) Either 0^n or 1^n works. For instance, 1^n compresses to $10^{\lfloor \log n \rfloor}(n)_2$, where $(n)_2$ is the binary representation of n. The compression ratio is $\frac{1+\lfloor \log n \rfloor + \lfloor \log n \rfloor + 1}{n} = \frac{2\lfloor \log n \rfloor + 2}{n}$.

b) Either $(0011)^{n/4}$ or $(1100)^{n/4}$ works. Each run compresses to 010. The compression ratio is $\frac{1+6(n/4)}{n} = 1.5n + 1/n$.

Tries

Several possible algorithms. Here are some:

- Find the lowest common ancestor of b_1 and b_2, let's call it a. Then try to find the predecessor of a (denoted as $p(a)$) and the successor of a (denoted as $s(a)$) using at most $\max(|b_1|, |b_2|)$ downward steps for each. Return true if $\{a, p(a)\} = \{b_1, b_2\}$ or $\{a, s(a)\} = \{b_1, b_2\}$. Otherwise, return true if a stores no key while $\{p(a), s(a)\} = \{b_1, b_2\}$. If all attempts fail, return false.

- Search for b_1. Then try to find the predecessor and successor of b_1, taking at most $|b_2|$ steps downward for each. If either of them is b_2, then return true. Otherwise, return false.

- Search for both b_1 and b_2 to form left and right boundaries (not necessarily in that order). Return true if there are no inside nodes (the right child of a node which is in the left boundary but not right boundary, and vice versa), and false otherwise.

Finding the predecessor/successor: To find the predecessor of k, first check if k has a left child. If so, then check the subtree of the left child of k and return its rightmost element, i.e., keep going right until you're unable to.

If k does not have a left child, then check its ancestors from bottom to top until we find a node a such that k is in the right subtree of a, and return a. If no such a is not found, then k has no predecessor.

Lempel-Ziv-Welch Encoding

Encoded string:

$68-65-82-75-95-128-78-95-66-129-75-83-132-65-78-75$

Added dictionary entries:

Code	Substring
128	DA
129	AR
130	RK
131	K␣
132	␣D
133	DAN
134	N␣
135	␣B
136	BA
137	ARK
138	KS
139	S␣
140	␣DA
141	AN
142	NK

Notice that `DAN` was already in the dictionary before `DANK` was encoded. Even though `DAN` would have only needed one codeword, we ended up using two codewords for just the `AN` part, since the `D` was already encoded from a previous substring.

String Matching Automata

a) The DFA:

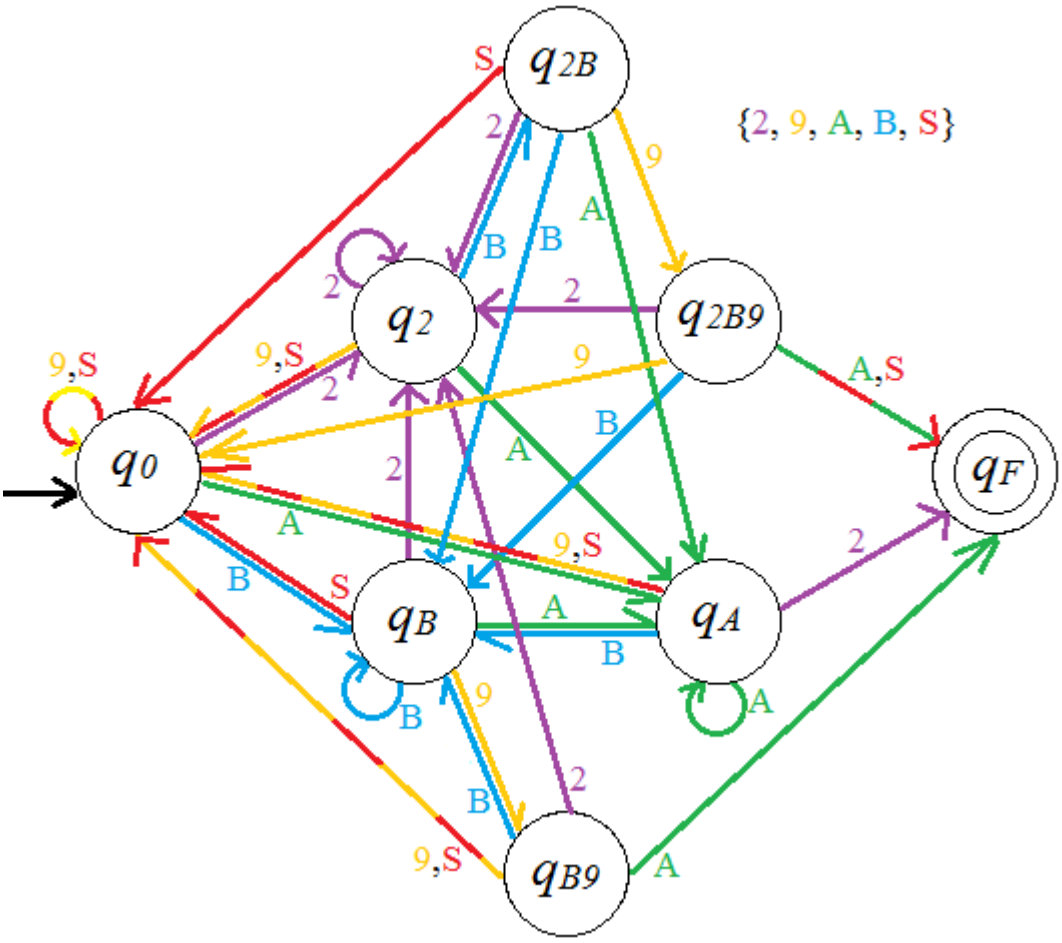


Figure 5: DFA for part (a)

b) $q_0 \to q_2 \to q_{2B} \to q_{2B9} \to q_2 \to q_0 \to q_A \to q_B \to q_{B9} \to q_F$.

c)

i	0	1	2	3	5	6	7
$P[i]$	9	2	B	A	S	2	B
$S[i]$	-7	-6	-5	-4	0	-2	6

Suffix Trees

a) Suffix Tree:

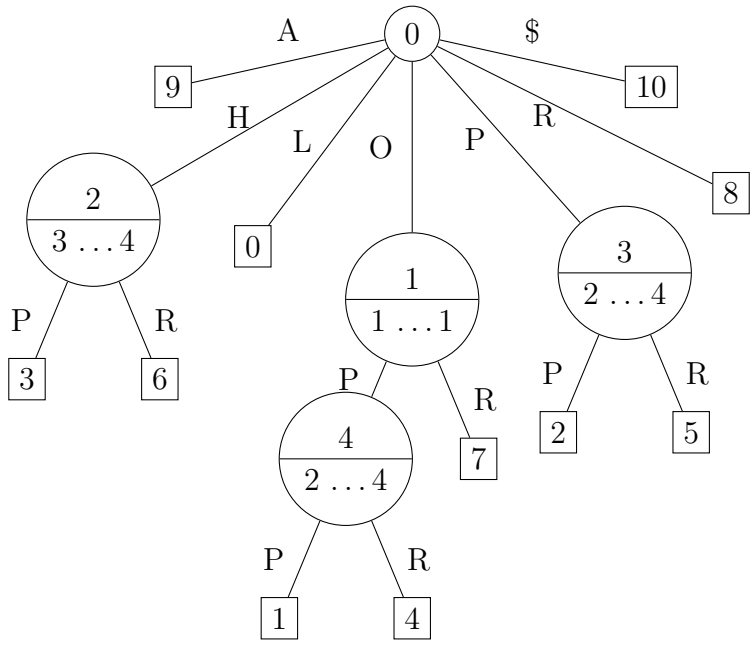


Figure 6: Suffix Tree for `LOPHOPHORA`

b) Traverse the suffix tree in any order to find the internal node such that the index of the next character to be tested is the highest from all internal nodes in the tree. This is achieved in linear time. Let this index be ℓ. From there, we can identify the length-ℓ prefix that corresponds to this internal node by dropping to any leaf node and reading the first ℓ characters from the corresponding suffix. This length-ℓ prefix is the longest repeated substring returned by the algorithm.

   To see why this works, observe that every internal node of a suffix tree represents a common prefix between multiple different suffixes, and therefore represents a repeated substring in the text. The index of the next character to be tested represents the length of the common prefix, and therefore the length of the repeated substring. So the internal node with

the highest such index has the longest repeated substring as the prefix represented by the node.

Burrows-Wheeler Transform

i	0	1	2	3	5	6	7	8	9	10	11	12	13
A	E	P	E	S	L	P	P	\$	A	S	E	A	R
$sort(A)$	\$	A	A	E	E	E	L	P	P	P	R	S	S
i	8	9	12	0	2	11	5	1	6	7	13	3	10

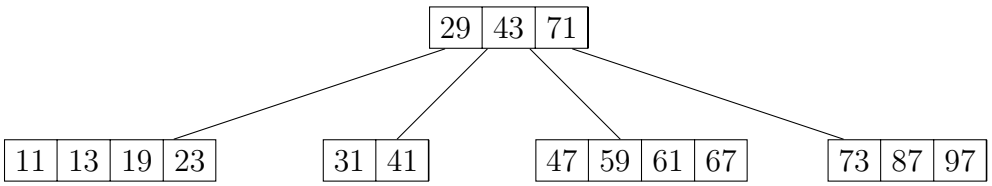
Therefore, we have

j	1	9	6	11	13	10	7	5	2	12	3	0	8
Original Text	P	A	P	E	R	S	P	L	E	A	S	E	\$

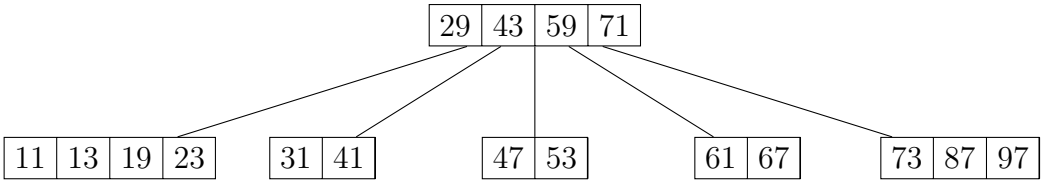
The original string was `PAPERSPLEASE$`.

B-Trees

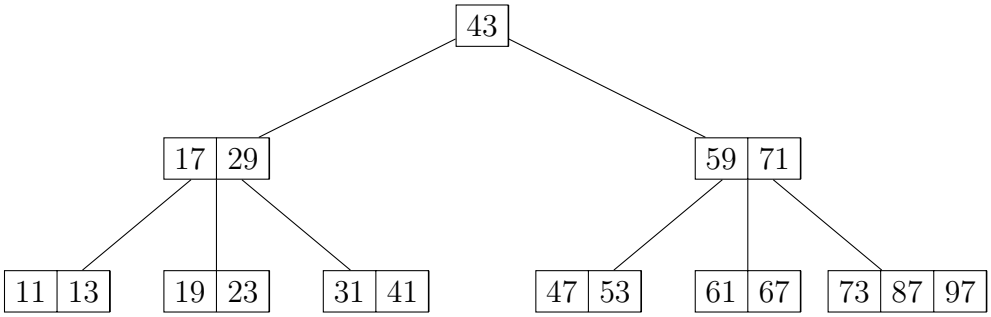
a) Insert (13):



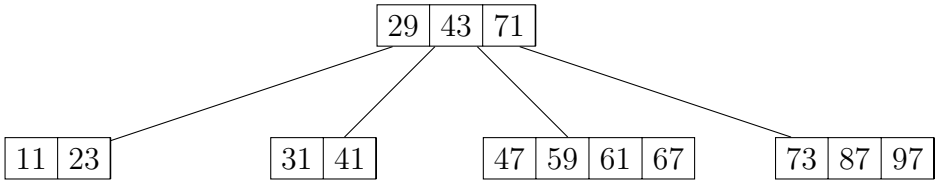
Insert(53):



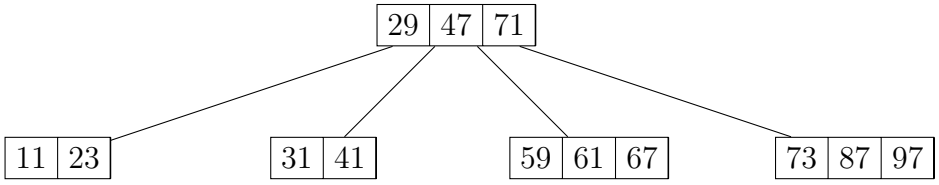
Insert(17):



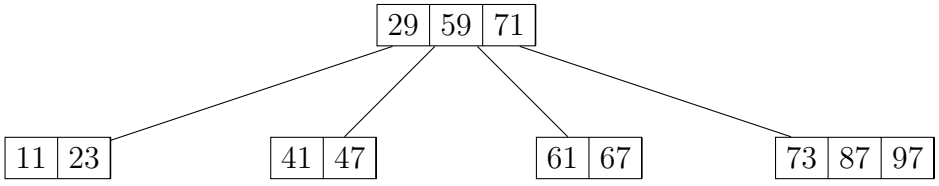
b) Delete(19):



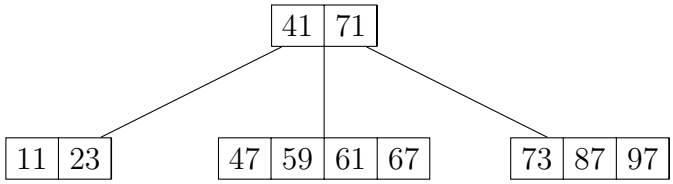
Delete(43):



Delete(31):



Delete(29):



Range Query

The idea is to use some structure similar to A4P3b, where we have a range tree such that each node is augmented with some data about its entire subtree. A relatively simple solution is as follows:

Preprocessing: Construct a balanced BST based on the indices, i.e., the keys encompass the range 0 to $n - 1$. Then augment the following data to each node:

- `val`: the value $A[k]$, where k is the key of the node;
- `max`: the maximum value from all nodes within its subtree;
- `min`: the minimum value from all nodes within its subtree.

There are n nodes, and each node stores a constant quantity of data, so the overall space complexity is still in $O(n)$.

Range Query: Given the arguments i and j, we perform a BST search for i and j to form P_1 and P_2 respectively. This takes $O(\log n)$ time.

Similar to range search on range trees, every node in either P_1 or P_2 are boundary nodes. For each node in P_1 (including i) but not P_2, its right child is a "top" inside node. Likewise for each node in P_2 (including j) but not P_1, its left child is also a "top" inside node. There are $O(\log n)$ boundary nodes and $O(\log n)$ "top" inside nodes.

Now, every key between i and j inclusive is either a boundary node or an inside node. The inside nodes are guaranteed to be within the range. So the maximum value in $A[i \ldots j]$ is the highest value that is either the `max` entry of a "top" inside node, or the `val` of a boundary node whose key is in the range.

Thus, we only need to check $O(\log n)$ nodes to find the maximum value in $A[i \dots j]$. Similarly, we can find the minimum value in $O(\log n)$ time, and then compute the maximum difference in $O(1)$ time, so the total runtime is still $O(\log n)$.

For example, with the array presented in the question, our special BST can look something like this:

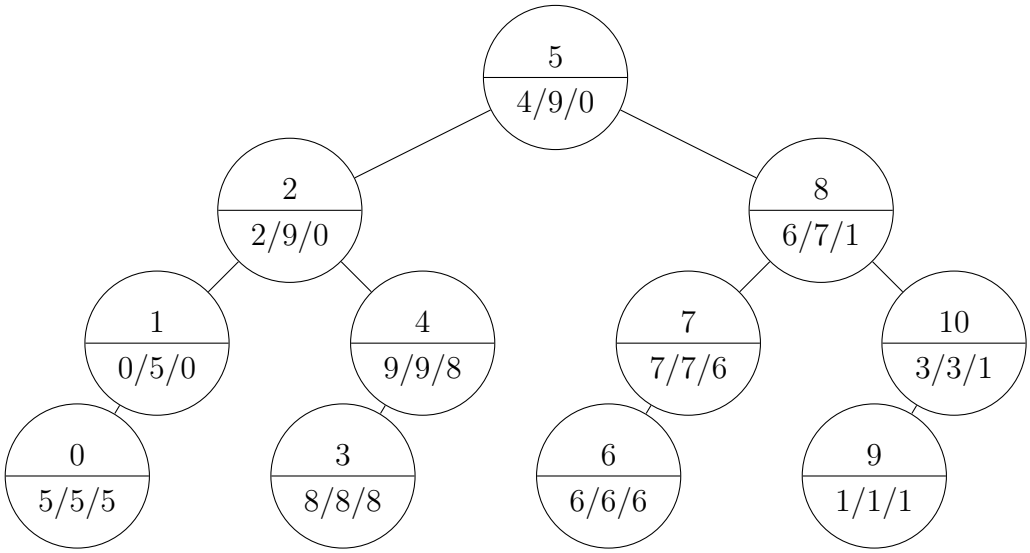


Figure 7: BST for Range Query

For each node, the number on top is the index or key, while the numbers below are `val`/`max`/`min`.

Running $MaxDiff(3, 7)$ results in $P_1 = (5, 2, 4, 3)$ and $P_2 = (5, 8, 7)$, with 6 being the sole inside node. Indices 2 and 8 are out of the range, so

$$\begin{aligned} \text{Maximum} &= \max(5.\texttt{val}, 4.\texttt{val}, 3.\texttt{val}, 7.\texttt{val}, 6.\texttt{max}) = 4.\texttt{val} = 9; \\ \text{Minimum} &= \max(5.\texttt{val}, 4.\texttt{val}, 3.\texttt{val}, 7.\texttt{val}, 6.\texttt{min}) = 5.\texttt{val} = 4; \\ MaxDiff(3, 7) &= 9 - 4 = 5. \end{aligned}$$

Similarly, running $MaxDiff(5, 10)$ yields $P_1 = (5)$, and $P_2 = (5, 8, 10)$ with 7 and 9 being "top" inside nodes (6 is also an inside node, but it's part of

7's subtree). All boundary points are in the range, so

$$\begin{aligned}
\text{Maximum} &= \max(5.\texttt{val}, 8.\texttt{val}, 10.\texttt{val}, 7.\texttt{max}, 9.\texttt{max}) = 7.\texttt{max} = 7; \\
\text{Minimum} &= \max(5.\texttt{val}, 8.\texttt{val}, 10.\texttt{val}, 7.\texttt{min}, 9.\texttt{min}) = 9.\texttt{min} = 1; \\
MaxDiff(5, 10) &= 7 - 1 = 6.
\end{aligned}$$