Homework 3 Solutions
Fundamental Algorithms, Spring 2025, Professor Yap, Section Leader Dr. Bingwei Zhang

Due: Friday Feb 28, in GradeScope by 11:30pm.
HOMEWORK with SOLUTION

INSTRUCTIONS:

- We have a "NO LATE HOMEWORK" policy.
  Special permission must be obtained *in advance* if you have a valid reason.
- Any submitted solution must be fully your own (you must not look at a fellow student's solution, *even if you have discussed with him or her*). Likewise, you must not show your writeup to anyone. We take the academic integrity policies of NYU and our department seriously. When in doubt, ask.
- The official deadline is 11:30pm, but can resubmit as many times as you like.

(Q1) (4+6+4 Points)
Exercise III.3.2, p. 29.
Rotation of zigzag list.

THE QUESTION Call T_n a **zigzag-list** if it is binary tree comprised of a single path

$$(u_0 — u_1 — \cdots — u_n)$$

from the root u_0 to a unique leaf u_n. Moreover, u_i $(i > 0)$ is a left child if i is odd, and a right child if i is even. Describe the result of performing $\texttt{rotate}^n(u_n)$ (i.e., n repeated rotations at u_n).
(i) Show the transformations on T_n in case $n = 6$.
(ii) For a general n, what are the left- and right-spines of u_n at the end?
(iii) What is the final height of T_n as a function of n?

SOLUTION FIGURE:

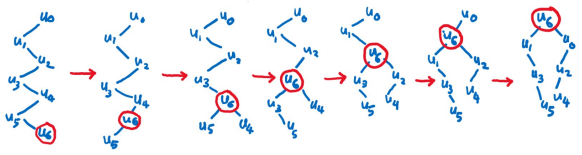

Figure 1: Zigzag-list T_6 and result of $\texttt{rotate}^6(u_6)$

SOLUTION:
(i) See Figure 1:
(ii) The left spine of u_n would be $(u_n — u_1 — u_3 — u_5 — \ldots)$ and the right spline of u_n would be $(u_n — u_0 — u_2 — u_4 — \ldots)$. We can prove this by induction.
(iii) The final height of T_n is $\lceil n/2 \rceil$.

(Q2) (10+10+10 Points)
Exercise III.6.5, p.56.
Choosing a deletion from an AVL tree of size 12 in order to achieve certain effects.

THE QUESTION Draw an AVL tree with 12 nodes such that, by deleting one node, you achieve these effects (respectively):
(a) Rebalancing requires two double-rotations.
(b) Rebalancing requires one single rotation and one double-rotation.
(c) Rebalancing requires two single rotations.

You can use a different tree for parts (a)-(c). You must draw the tree after each single- and double-rotation. HINT: It is unnecessary to assign keys to the nodes: just show the tree shape, and label some nodes to clarify the operations.

Solution Figure:

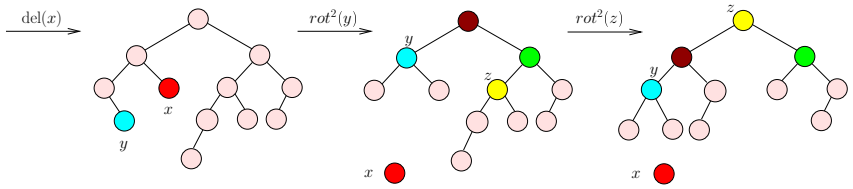

Figure 2: Deleting Node x causes two double-rotations

SOLUTION: The AVL tree for this question is a min-size AVL tree T_4 of height 4. We can view this problem structurally (just draw the tree without the keys). Let u be a node in an AVL tree. Let $Swap(u)$ be the operation where you swap the left- and right-subtrees of u. Clearly, swapping preserves AVL-ness. Parts (b) and (c) can be obtained from the tree in part(a) by such swaps!

For part (a), see Figure 2 for the case of two double-rotations.

Part(b): if you $Swap(parent(y))$ in the original tree, then a deletion of x will cause one single rotation, followed by a double-rotation as before.

Part(c): starting from the tree in part(b), if you $Swap(root)$, then a deletion of x will cause two single rotations. Alternatively, you can also do $Swap(root.right)$.

(Q3) (12+14 Points)
Exercise III.6.7, p. 57.
There are 3 parts, but you only need to do parts (a) and (b). I will slightly reformulate the original question here.

We want to develop an exact formula for $\mu(h)$, the min-size AVL tree of height h. This satisfies the recurrence

$$\mu(h) = 1 + \mu(h-1) + \mu(h-2)$$

for $h \geqslant 2$, with $\mu(0) = 1, \mu(1) = 2$. The presence of "1+" makes $\mu(h)$ non-homogeneous. We want to reduce it to the homogenous case without the "1+" term (i.e., like Fibonacci recurrence).
(a) First consider a general linear recurrence of the form

$$T(n) = C + \sum_{i=1}^{k} a_i T(n-i) \tag{1}$$

where $C, a_1, \ldots, a_k$ are constants. Assume $C \neq 0$, but we want to transform it into the homogeneous linear recurrence,

$$t(n) = \sum_{i=1}^{k} a_i t(n-i). \tag{2}$$

by choosing $t(n) = T(n) + c$ for some constant c. Note that the a_i's in (1) and (2) are identical. What is c in this transformation? Also, state a limitation in this transformation.
(b) Use the method of part(a) to give an exact solution for $\mu(h)$ of the form

$$\mu(h) = c + A\phi^h + B\widehat{\phi}^h, \qquad (h \geqslant 0)$$

where $\phi, \widehat{\phi}$ are the roots of $x^2 - x - 1$ and $A = \frac{2}{\sqrt{5}} + 1 = 1.8944$ and $B = -\frac{2}{\sqrt{5}} + 1 = 0.1056$.
(c) (Yuejie Wang, fall2020) The transformation of part(a) can be generalized so that it has no limitations. Show that the homogeneous linear recurrence (2) can always be achieved by $t(n) = T(n) + cn^s$ for some constant c and some $0 \leqslant s < k$.

SOLUTION: (a) Let $t(n) = T(n) + c$.

$$\begin{aligned} t(n) &= T(n) + c \\ &= C + \sum_i a_i T(n-i) + c \\ &= C + \sum_i a_i (t(n-i) - c) + c \\ &= C + \sum_i a_i t(n-i) - (c \sum_i a_i) + c \\ &= \sum_i a_i t(n-i) \end{aligned}$$

where $C = c\sum_i a_i - c$ or

$$c = C/(-1 + \sum_i a_i). \tag{3}$$

So the limitation is that $\sum_{i=1}^{k} a_i$ must not be equal to 1.
Comments:

SOLUTION: (b) Since

$$\mu(h) = 1 + \mu(h-1) + \mu(h-2)$$

and the limitations of (a) does not apply in this case, we may apply the transformation

$$\overline{\mu}(h) = c + \mu(h) = 1 + \mu(h),$$

from part(a) (with $C = c = 1$) to obtain a homogeneous Fibonacci recurrence

$$\overline{\mu}(h) = \overline{\mu}(h-1) + \overline{\mu}(h-2).$$

Thus $\overline{\mu}(h) = A\phi^h + B\widehat{\phi}^h$. We can determine A and B using initial conditions:

$$\begin{aligned}
2 &= \overline{\mu}(0) = A\phi^0 + B\widehat{\phi}^0 = A + B \\
3 &= \overline{\mu}(1) = A\phi^1 + B\widehat{\phi}^1 = \frac{\sqrt{5}}{2}(A - B) + 1 \\
2 &= \overline{\mu}(1) = \frac{\sqrt{5}}{2}(A - B) \\
\begin{bmatrix} 2 \\ 2 \end{bmatrix} &= \begin{bmatrix} 1 & 1 \\ \frac{\sqrt{5}}{2} & -\frac{\sqrt{5}}{2} \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} \\
\begin{bmatrix} A \\ B \end{bmatrix} &= -\frac{1}{\sqrt{5}} \begin{bmatrix} -\frac{\sqrt{5}}{2} & -1 \\ -\frac{\sqrt{5}}{2} & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \end{bmatrix} \\
&= \begin{bmatrix} \frac{2}{\sqrt{5}} + 1 \\ \frac{2}{\sqrt{5}} - 1 \end{bmatrix} = \begin{bmatrix} 1.8944 \\ -0.1056 \end{bmatrix}.
\end{aligned}$$

In conclusion, the exact formula is

$$\mu(h) = -1 + \overline{\mu}(h) = -1 + A\phi^h + B\widehat{\phi}^h$$

Comments: Since $\mu(h)$ is an integer, we may check that the formula $\mu(h) = \lfloor -1 + A\phi^h \rceil$ is true all $h \geqslant 0$. Using a calculator, we can check that $\mu(h) = -1 + 1.8944\phi^h + 0.1055\widehat{\phi}^h$. the correct values of $\mu(5) = 20$ and $\mu(6) = 33$.

SOLUTION: (c) If $1 \neq \sum_{i=1}^{k} a_i$, then then part(a) shows that we can choose $s = 0$ or $t(n) = T(n) + c$. Otherwise, we $s \geqslant 1$ to be the smallest integer such that the value

$$b_s := \sum_{i=1}^{k} a_i (-i)^s \tag{4}$$

is non-zero. Write b for this b_s. Moreover, we can show that s satisfies $1 \leqslant s < k$ by appeal to the non-singularity of the Vandermonde's matrix for the linear system corresponding to (4). Writing $f(n) = cn^s$, we check that

$$f(n) - \sum_{j=1}^{k} a_j f(n-j) = -bc.$$

We therefore obtain

$$\begin{aligned}
t(n) &= f(n) + T(n) \\
&= f(n) + C + \textstyle\sum_{j=1}^{k} a_j T(n-j) \\
&= f(n) + C + \textstyle\sum_{j=1}^{k} a_j \Big(t(n-j) - f(n-j)\Big) \\
&= f(n) + C - \textstyle\sum_{j=1}^{k} a_j f(n-j) + \sum_{j=1}^{k} a_j t(n-j) \\
&= C - bc + \textstyle\sum_{j=1}^{k} a_j t(n-j) \qquad \text{(by previous lemma)} \\
&= \textstyle\sum_{j=1}^{k} a_j t(n-j)
\end{aligned}$$

provided we choose $c = C/b$.

(Q4) (12+12 Points)
Exercise III.6.9 (p. 58).

Figuring out the possible heights of AVL trees of size 100. A helpful fact from the Chapter III is $\mu(h) \leqslant 2\phi^h$ where $\phi = 1.6180$ is the Golden ratio.

QUESTION: My pocket calculator tells me that $\log_\phi 100 = 9.5699\cdots$.
(a) What does this tell you about the maximum height achievable by an AVL tree of size 100?
(b) What is the range of heights for an AVL tree of size 100?
NOTE: we told you the pocket calculator value so that you could solve this problem without consulting your own calculators. Also $\log_\phi 2 = 1.44$ (rounded to 2 digits).

SOLUTION: (a) The maximum achievable height is 8. Let T be an AVL tree of size 100 and of maximum height, denoted by h. It follows that h is the smallest integer such that

$$\mu(h) \leqslant 100 < \mu(h+1).$$

We are only interested in the second inequality.

$$\begin{aligned} 100 &< \mu(h+1) \leqslant 2\phi^{h+1} \\ \log_\phi 100 &< \log_\phi(2) + h + 1 \\ 9.56 &< 1.44 + h + 1 \\ 7.12 &< h \end{aligned}$$

Since h is the smallest integer satisfying this inequality, this means that h must be 8.
(b) The range is 6 to 8. The upper bound of 8 is from part (a). To get the lower bound, we know that the maximum size of a binary tree with height h is $M(h) = 2^{h+1} - 1$, achieved by the perfect binary tree (which is clearly AVL). Since $M(6) = 2^7 - 1 > 100 > M(5) = 2^6 - 1$, we conclude that we can construct an AVL tree with 100 nodes of height 6. Namely, start with the perfect binary tree of height 5. Now fill in the next level of leaves until you reach size 100.
Comments: Wrong answers: Note that the argument of part(a) is a bit subtle. We must NOT use the first inequality, $\mu(h) \leqslant 100$.

(Q5) (4+(5x5) Points)
Exercise III.7.2 (p. 79).
Insertion/Deletion from RB Tree.
CORRECTION: the tree is found in Figure 17 on page 50. It is a BST of size 16, rooted at 12.
We *also* want you to write down ratio(u) next to each node u.

THE QUESTION Hand-simulation of insertion and deletions for Ratio Balanced Trees (RBT or $RB[1/3]$). Start with the binary search tree in Fig.17 (p.50 in ¶III.47). First, verify that it is an RB Tree by writing down ratio(u) next to each node u. Next, consider the following sequence of RBT insertions/deletions:

$$(a)\,Del(14), \quad (b)\,Del(1), \quad (c)\,Ins(6.5), \quad (d)\,Ins(7.5), \quad (e)\,Del(13).$$

Draw the RBT after each operations. Show the intermediate tree after each rebalancing act. A rebalancing act can be a single- or a double-rotation.
NOTE: starting from the initial RBT, you have 10 distinct trees to draw!

SOLUTION in Figure 3

(Q6) (8+10+10 Points)
Exercise III.7.4, p. 79.
Why do we restrict ρ to $(0, \frac{1}{2})$ in the definition of ratio balanced trees $RB[\rho]$?
HINT: what happens for trees of small sizes? For part(b), think of $\rho = (q-1)/q$ for $q = 1, 2, 3, 4, \ldots$.

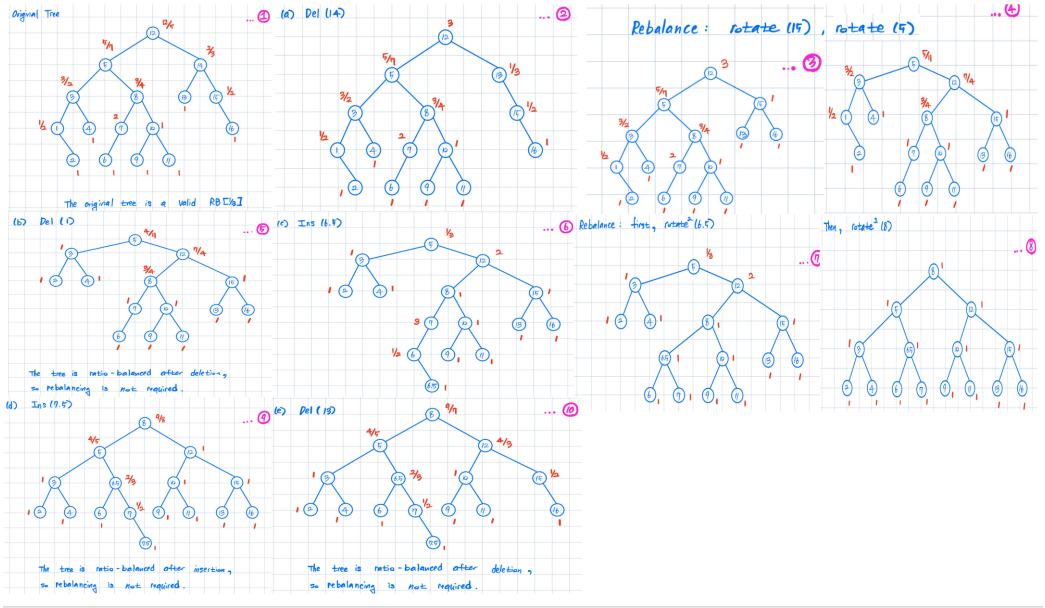

Figure 3: (a) Del(14), (b) Del(1), (c) In(6.5), (d) In(7.5), (e) Del(13).

THE QUESTION The ratio bound ρ in the ratio balanced class $RB[\rho]$ is normally restricted to $(0, \frac{1}{2})$. Can we extend to $\rho \in [\frac{1}{2}, 1)$?
(a) What is wrong with the class $RB[\frac{2}{3}]$?
(b) Which other values of $\rho \in [\frac{1}{2}, 1)$ pose problem similar to part(a)?
(c) Suppose we *really* want to allow values of ρ that lie in the range $(\frac{1}{2}, 1)$. How can we overcome the above objections?

SOLUTION: There is no BST in $RB[2/3]$ of `size` $= 4$ (i.e., `esize` $= 5 = 2 + 3$). In the "most balanced" situation, the `esize`'s of the left-subtree and right-subtree are 2 and 3. Then the ratio of the root does not lie in the range $(2/3, 3/2)$. Similarly, we had already excluded `size` $= 2$ (i.e., `esize` $= 3 = 1 + 2$) in the text.

SOLUTION: In general, for any integer $q \geqslant 2$, the class $RB[(q-1)/q]$ excludes any tree with `size` $= 2i$ (for $i = 1, 2, \ldots, q-1$). What if ρ is an irrational number in $(\frac{1}{2}, 1)$? In that case, there is a largest integer q such that $\frac{q-1}{q} < \rho$. Then $RB[\rho]$ will exclude the same `size`'s as $RB[(q-1)/q]$.

SOLUTION: From part(b) we know there are only a finite number of sizes are excluded in $RB[(q-1)/q]$. We can simply allow the ratio bounds for these sizes to be treated as "exceptions". Of course the insertion/deletion algorithms might be messy because of the special cases.

(Q7) (22 Points)
Exercise III.8.1, p. 100.
Insertion and deletion into a $(2, 3)$-search tree.

THE QUESTION Begin with the $(2, 3)$-search tree in Figure 38 (p.85). Perform the following sequence of operations:

$$(a) \quad Ins(20), \qquad (b) \quad Ins(19), \qquad (c) \quad Del(10)$$

Assume the standard insert/delete algorithms. A single operation may cause multiple rebalancing acts

(split, merge, borrow) which you must indicate.

SOLUTION: see Figure 4

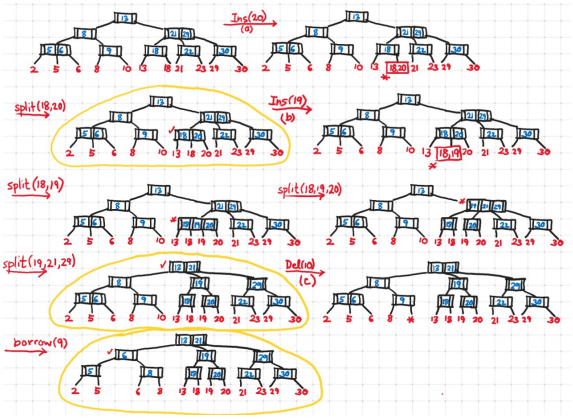

Figure 4: $(a)Ins(20)$, $(b)Ins(19)$, $(c)Del(10)$

(Q8) (2x(10+14) Points)

Exercise III.8.7, p.102.

Internal Organization of (a, b)-tree Nodes. The question has four parts (i)-(iv). BUT we ask you to only consider two of them, namely part(i) ordered array, and part(iv) AVL tree.

THE QUESTION We consider B-trees in this question. Since a B-tree node contains many keys, it must support searching within its nodes. We consider four different **schemes** for this internal organization:

(i) an ordered array,

(ii) an ordered singly-linked list,

(iii) an ordered doubly-linked list,

(iv) an AVL tree.

Consider a specific numerical example: each B-tree node is stored in a **block** whose size is $4096 = 2^{12}$ bytes. An address within the block is called a **local pointer**, and it needs only 12 bits. But for simplicity, we assume local pointers use two bytes. The address of each block in the secondary storage (i.e., disc) is specified by a **block pointer** that uses 4 bytes. Assume that each key take 6 bytes. Now answer these two questions for the schemes (i)-(iv):

(a) What is the maximum value of the parameter b under the scheme? Be sure to show your calculations.

(b) What is the worst case time to search for a key in a B-tree with two million items? Give the answer in terms of the number of CPU cycles ("CPUC") for each of the four schemes. State explicit numbers, not expressions. REMARK: in the text, we only count the number of I/O operations. But for part(b), you must also count the cost of searching within each block.

Use the following additional ASSUMPTIONS:

(A1) Each internal node must store a parent pointer as well as its degree (i.e., number of children). Use 2 bytes for the degree. For schemes (ii)-(iv), we also reserve two bytes for book-keeping purposes.

(A2) Since this is a B-tree, once you know b, you can set $a \leftarrow \lfloor (b+1)/2 \rfloor$.

(A3) Each disk I/O takes 1000 CPU cycles.

(A4) Assume that $(a', b') = (a, b)$. This determines the number of items in a leaf. Is this reasonable? Well, assume that an item represented by the pair $(key, data)$ uses 6 bytes for key and 4 bytes for $data$. Intuitively, the $data$ is only a pointer to the actual data in disc that may use much more space!

Therefore, the space requirement for an external node is comparable that of an internal node. We assume the search structure in an external node uses the same scheme (i)-(iv), but you may assume it has the same performance as the internal nodes.

(A5) If searching for a key takes $O(\lg n)$ or $O(n)$ CPU time, always assume that "4" is the constant in big-Oh notation. E.g., searching for a key in an AVL tree with $n = 100$ keys takes $4 \lg n = 4 \times 7 = 28$ CPUC (taking $\lg 100 = 7$). E.g., Searching in a linked list of length $n = 100$ takes $4n = 400$ CPUC.

(A6) The root of the search tree is always in main memory, so you never need to read or write the root.

(A7) You may use calculators, but we encourage simplified estimates whenever possible (e.g., you can assume $\lg 10^6 \simeq 20$).

SOLUTION

(a) Since each block must keep track of its degree (2 bytes) and the parent (4 bytes), we have $4096 - 6 = 4090$ bytes to organize the data structure of (i)-(iv). Remark: knowing the degree of a node is useful for checking if a node is overfull or underfull.

(a-i) Array representation. This is the simplest. With b children, we need $10b$ bytes. So $10b \leqslant 4090$. So we can choose $b = 409$.

Before providing the solution for (ii)-(iv), let us explain why we ask you to reserve 2 bytes for "book-keeping". A common feature of (ii)-(iv) is that we want to divide the block into a set of **local nodes** (or **L-nodes** for short) of some fixed size. Each L-node is either used or free. When you insert into your data structure (linked list, doubly-linked list or BST), you need to get a free L-node. When you delete, some L-node becomes free. Thus, you must keep track of the set of free L-nodes. The classic way to keep track these nodes is to singly-link the free L-nodes together into a FREELIST. To get a free L-node, you just return the head of FREELIST, and update the head to the next node in FREELIST. Therefore, the management of FREELIST has no impact on the size of your nodes. But you need to allocate 2 bytes in the block to point to the head the FREELIST. THEREFORE each block now has $4096 - 8 = 4088$ bytes available.

(a-ii) Singly-linked list. Each L-node of the linked list needs 12 bytes: 4 bytes for a block pointer, 6 bytes for the key, and and 2 bytes to point to the next L-node. Thus the number of $12b \leqslant 4088$. So the largest possible choice is $b = 340$.

(a-iii) Doubly-linked list. Each node needs 14 bytes, because we need an extra L-node when compared to (ii). Thus the maximum value of b is $\lfloor 4088/14 \rfloor = 292$.

SOLUTION

(a-iv) AVL tree. Each L-node needs 16 bytes: this is two bytes more than (iii) because the L-node needs a left-, right- and parent pointer. So a simple answer is to choose

$$b = \lfloor 4088/16 \rfloor = 255.$$

But this is too optimistic. Let us see why. Each AVL node needs to maintain two bits of balance information (for simplicity, we allocate one byte to this). See ¶III.48, p.50, for details. A more subtle issue is this: when you lookup a key K, you get back the closest key K_i in the BST. We may have $K < K_i$ or $K \geqslant K_i$. Thus, you need to get to either P_i or P_{i-1} where P_j's are the pointers to children nodes of the B-tree. Suppose you store P_i with K_i. But how to you get to P_{i-1}? The simplest solution is to assume that your AVL nodes also has a predecessor pointer (these are local pointers). But in order to maintain predecessor pointers, we also need to maintain successor pointers. That means that each AVL node needs, not 16 bytes, but 21 bytes (1 byte for balance info, four bytes for succ/pred pointers). So $b = \lfloor 4088/21 \rfloor = 194$. *BUT don't worry, we ask the graders to be gentle if you miss some of these subtleties.*

(b) Generalities about height. For worst case height, the degree of the root is 2 and the degree of each internal node is a in an B-tree. A tree of height $h \geqslant 1$ must have at least $2a^h$ items. It follows that $2 \times 10^6 \geqslant 2a^h$. Taking logs, the height h of our search tree satisfies $h \leqslant \lg(10^6)/\lg(a)$. Using our lemma on Mixed Integer Inequalities, this means

$$h \leqslant \left\lfloor \lg(10^6)/\lg(a) \right\rfloor .$$

You may assume $\lg(10^6) = 20$. IMPORTANT: if you wrongly use the inferior bound of "$h \leqslant \lceil \lg(10^6)/\lg(a) \rceil$", your answers below will be quite a bit off!

(b-i) Array representation: Since $b = 409$, we have $a = \lfloor (1+b)/2 \rfloor = 205$ and height is at most $\lfloor 20/\lg(205) \rfloor = \lfloor 2.59 \rfloor = 2$. Since the root is always in main memory, we need to read in read 2 nodes, taking 2000 CPUC. In each node, we need to search the keys in an array to determine the child pointer. We can use binary search in an array, and time is $4\lceil \lg b \rceil = 4 \times 9 = 36$ CPUC. We need to do this search on three nodes in the search path, so this takes $3 \times 36 = 108$ CPUC. So answer is $2000 + 108 = 2108$ CPUC.

(b-ii) Singly-linked list. Since $b = 340$, we have $a = 170$, and height is $\lfloor 20/\lg(170) \rfloor = \lfloor 2.69 \rfloor = 2$. In each node, we need to search the keys in the linked list to determine the child pointer, and time is linear, $4b = 1360$ CPUC. We need to do this search on three nodes, so this takes $3 \times 1360 = 4080$ CPUC. So answer is $2000 + 4080 = 6080$ CPUC.

(b-iii) Doubly-linked list. Since $b = 292$, we have $a = 146$, and height is $\lfloor 20/\lg(146) \rfloor = \lfloor 2.77 \rfloor = 2$. So the answer is the same in (i).

(b-iv) AVL tree. Since $b = 194$, we have $a = 97$, and height is $\lfloor 20/\lg(97) \rfloor = \lfloor 3.02 \rfloor = 3$. Now, searching is an AVL of height $\lceil \lg(97) \rceil$) takes $4 \times \lceil \lg(97) \rceil) = 28$ CPUC. Doing this on four nodes takes time $4 \times 28 = 112$ CPUC. So answer is $3000 + 112 = 3112$ CPUC.

CONCLUSION: The array solution is the fastest. AVL tree comes next. Of course, changing some of our assumptions may lead to completely different answers!