מבוא לתורת המספרים

שם הקורס

רוגרו ברנס

שם התלמיד

317759496 — מספר התלמיד

204078 — מספר הקורס

1.10.13 — תאריך

2013 ב — מחזור לימודים (סמסטר)

[ ] 58870

מלא את הפרטים בכל המקומות הדרושים

וקרא

ה

1002919061

ב2013 20407 86

אסמכתא: 58870

ת"ז: 317759496

ברנס רוברט 242

מותר לצלם שאלון

58870

מבוא

הקפד לרשום א

317759496 — מספר התלמיד

5 — חדר

42 — בחינה

מרכז

מספר המועד (ר

204078 — מספר הקורס

86 — מספר המועד

לשימוש הבודקים

בדיקה ראשונה		שם הבודק	בדיקה שניה
097	ציון הבחינה		ציון הבחינה
025	ציון שאלה 1		
24	ציון שאלה 2		
25	ציון שאלה 3		
--	ציון שאלה 4		
23	ציון שאלה 5		
	ציון שאלה 6		
	ציון שאלה 7		
	ציון שאלה 8		
	ציון שאלה 9		
	ציון שאלה 10		
	ציון שאלה 11		
	ציון שאלה 12		
	ציון שאלה 13		
	ציון שאלה 14		
	ציון שאלה 15		
	ציון שאלה 16		
	ציון שאלה 17		
	ציון שאלה 18		
	ציון שאלה 19		
	ציון שאלה 20		
	ציון שאלה 21		
	ציון שאלה 22		
	ציון שאלה 23		

לשימוש הבודק

$\downarrow \frac{1}{3} \downarrow \frac{2}{3} \downarrow$

$[\quad | \quad | \quad]$ Best Case

שאלה 2 13:06

(A) Worst Case: We choose the two leftmost or right most elements as the pivots.

Best Case: We choose the elements at $\left\lfloor \frac{end-start+1}{3} \right\rfloor + start$ and $\left\lfloor \frac{2(end-start+1)}{3} \right\rfloor + start$ i.e. the elements at positions $\frac{1}{3}$ & $\frac{2}{3}$ in the array.

We can use deterministic Select to choose the pivots, since it takes $O(n)$ time to partion around a given position.

(B) $T(n) = T(n-2) + T(0) + T(0) + \Theta(n)$ We assume $T(0) = \Theta(1)$

$T(n-2) = T(n-4) + T(0) + T(0) + \Theta(n-2)$

$T(n-4) = T(n-6) + 2T(0) + 2T(0) + \Theta(n-4)$

$T(n) = \cancel{T(n-2)} + \cancel{T(n-4)} + T(n-6) + 6T(0) + \Theta(n) + \Theta(n-2) + \Theta(n-4)$

שורה I $T(n) = T(n-2I) + 2I \cdot T(0) + \Theta(n) + \ldots + \Theta(n-2(I-1))$

(*) $n-2+n-4+n-6$

$\frac{n}{2}n - \sum_{i}^{n/2} 2i$

(A.1)

$2 \cdot \sum_{i}^{n/2} 2i = 2 \cdot \frac{1}{2} \frac{n}{2}\left(\frac{n}{2}+1\right)$

$\frac{n^2}{2} - \frac{n^2}{4} - \frac{n}{2}$

$= n^2\left(\frac{1}{2}-\frac{1}{4}\right) - \frac{n}{2}$

The recursion stops when the height of the recursion tree is $\frac{1}{2}n$ and we do $\Theta(n)$ work at each level resulting in an $\boxed{\Theta(n^2)}$ algorithm.

$T(n) = n \cdot T(0) + \sum_{i=0}^{n/2} (n-2i)$

$= n \cdot T(0) + \frac{n^2}{2} - \sum_{i}^{n/2} 2i = \Theta(n^2)$ (*)

(C) $T(n) = 3 \cdot T(n/3) + \Theta(n)$ $f(n) = \Theta(n)$

$f(n) = \Theta(n^{\log_3 3})$ and by Case 2 of Master Method

$T(n) = \Theta(n \lg n)$

32:37

Dequeue(S) Remove the element at the head of the list, then use the stored pointer to call RB-Delete. We then call Tree-Minimum and update Min

$O(1) + O(\lg n) + O(\lg n) = O(\lg n)$

Minimum(S) We return the element pointed to by Min. $O(1)$

54:45

לשימוש הבודק

שאלה 4

1:01

1:50

1:57

1. According to the answer I received h is given and is the height of the tree.

לשימוש הבודק

לשימוש הבודק

(1) RB-Tree: Each node in the tree contains a count of key's (objects) and a ~~linked~~ doubly linked list of objects with that key. Maintain head & tail pointers. $\boxed{RB}$ $\boxed{S \text{ אלה}}$

1:01

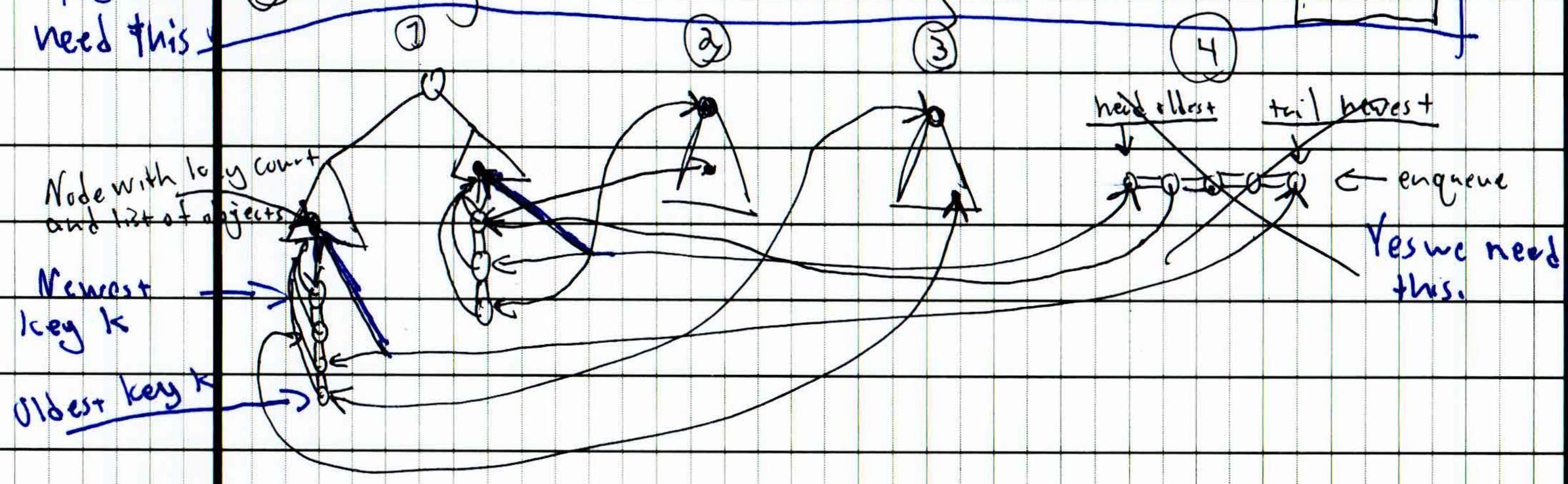
~~(2) Max-Freq Pointer: Points to node with largest number of keys.~~

(2) Max-Heap whose keys are the key counts $\boxed{C}$

(3) Max-Heap whose keys are the time gaps. $\boxed{G}$

Yes we need this

~~(4) Doubly linked list containing insertion times~~ $\boxed{DDQ}$



All nodes in each structure contain pointers to the corresponding node in the other structures. Each node in a List in ~~the~~ RB has a pointer back to the rb node it is part of so counts can be modified on deletion.

Search(S, k) We use the regular RB-Search (binary search) on RB. $O(\lg n)$

Insert(S, k) First we search for k in RB. $O(\lg n)$. If we find it, we increment the count and insert at the head of the list. If it's not found, do regular RB-Insert $O(\lg n)$ and set count to 1. In the first case we call Heap-Increase-Key(I) $O(\lg n)$ on the corresponding node in C. Otherwise insert a new node into C. $O(\lg n)$ We now insert the time stamp at the tail of DDQ. We ~~check~~ calculate the difference between this time and the previously inserted time, then insert this into G. We maintain all the relevant pointers between nodes in RB, C, G & DDQ

Insertions into heaps use Max-Heap-Insert $O(\lg n)$

לשימוש הבודק

Max-Freq-Key: Call Maximum on C. $O(1)$

Delete-Old(S, k) Call Search(S, k) $O(\lg n)$ to find the node in RB. Then use the tail pointer to get the oldest. Delete from DDQ. If it's the head or tail there make sure to update those pointers. Otherwise just delink it. $O(1)$ Call Maxheap-Delete on the node in G. $O(\lg n)$ Decrement the objects key in C and call Max-Heapify $O(\lg n)$ or if the count reaches $\emptyset$ Max-Heap-Delete. $O(1)$ We then remove the object from the list in RB node, and decrement the count there. If count reaches $\emptyset$ call RB-Delete on the node $O(\lg n)$ total $O(\lg n)$

? אפשר לעדכן

Delete-New(S, k) Identical to Delete Old except that we take the head of the list at node k in RB.

Max-Time-Gap(S) Call Maximum on G. $O(1)$

1:49

לשימוש הבודק

שאלה 1 2:00

Max-Heap-Increase-Key(A, i, ~~key~~ d)

~~if key < A[i]~~

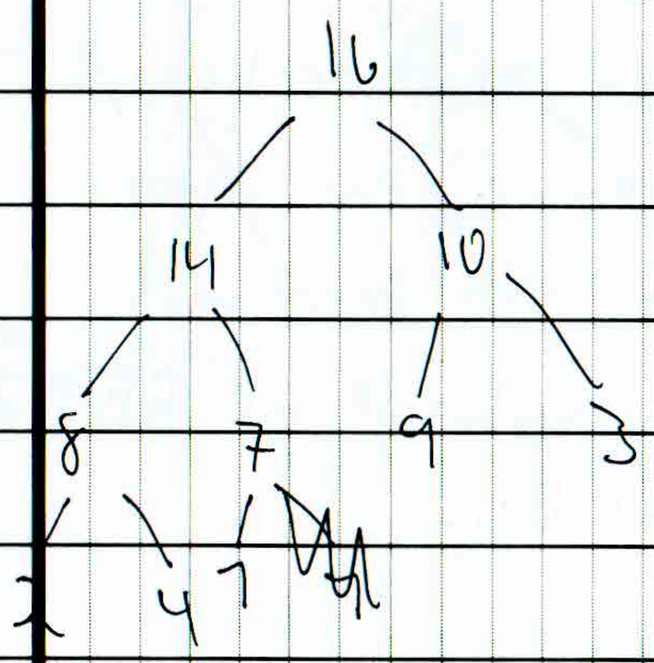
~~error "new~~

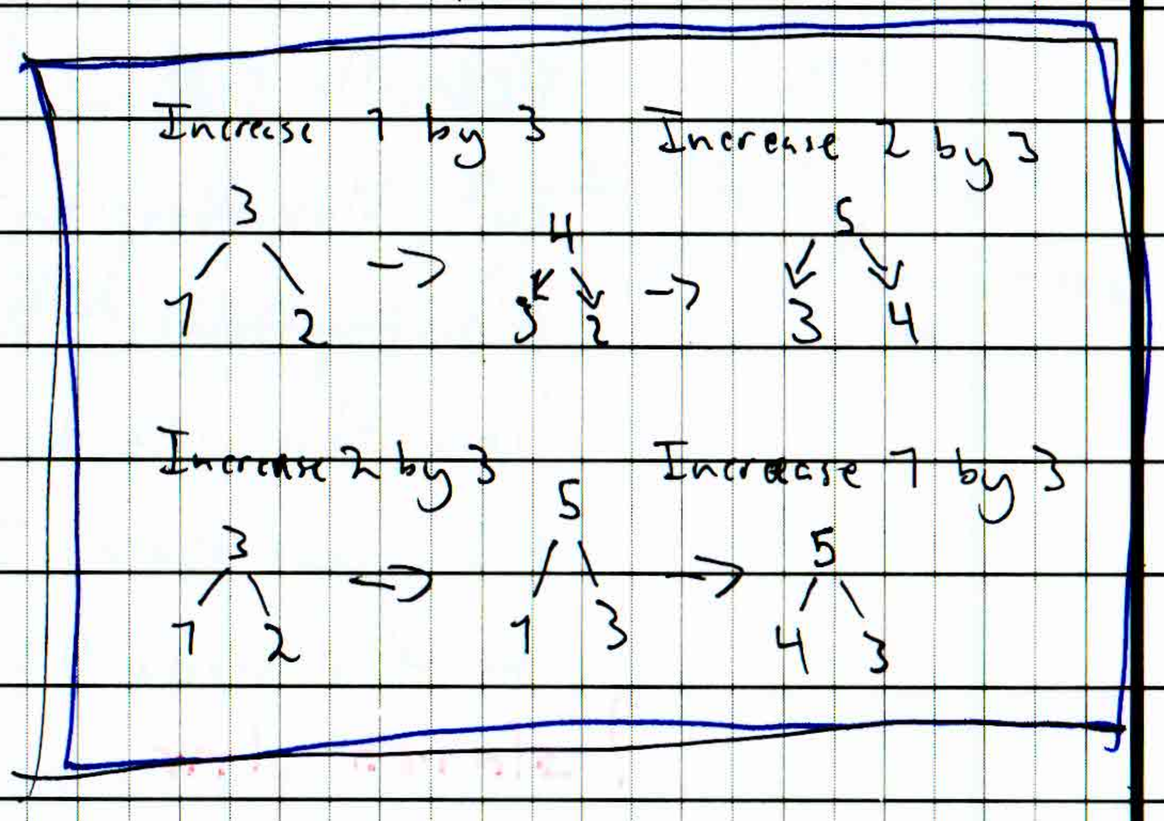
A[i] = A[i] + d

while i > 1 and A[Parent(i)] < A[i]

swap(A[i], A[Parent(i)])

i = Parent(i)





① As can be seen from this example different orders of calls can result in different trees.

② When do diffrent orders of calls result in the same heap?

① When the ~~two~~ calls result in neither element "passing" the same common ancestor node on it's way up the heap, then the order of calls won't matter. ~~But when the two elements~~ Passing may be too strong of a word. The two element must not do a comparison against any common ancestor node in the heap. If they do, then the order of calls might matter, depending on ~~the~~ if either of the element exchange places with the common ancestor.

2:24

לשימוש הבודק

שאלה 3 32:48

(A) We can use a doubly linked list with pointers to the tail and head of the list.

~~For Enqueue and Dequeue we use the standard functions from the book~~

Enqueue(S, z) We append a new element to the node pointed to by tail. $O(1)$

Dequeue(S) We remove the element pointed to by head. $O(1)$

Minimum(S) We do a linear scan of the list keeping a pointer to the minimum element encountered so far. ~~We then use the next, prev pointers of the element~~ We then return a pointer to that element. $O(n)$

(B) ~~We can use a Red-Black tree whose keys are the insertion times (assumed unique) and a pointer~~

We use a doubly linked list as above, plus a Red-Black tree. We maintain pointers between corresponding elements in the two structures, and maintain a pointer to the minimum element.

Enqueue(S, z) We insert z at the end (tail) of the list and then use RB-Insert to insert into the RB-Tree. We then call Tree-Minimum to update Min.

$O(1) + O(\lg n) + O(\lg n) = O(\lg n)$

(Making sure to maintain pointers between corresponding nodes)