CMSC 641
Spring 2010

26/30

Design and Analysis of Algorithms
Quiz 2, Amortized Analysis

Name: Sandipan Dey

Sorted Array with Sidekick

Storing items in a sorted array has the advantage that binary search can find an item in $\Theta(\log n)$ time. However, insertion into a sorted array is cumbersome and takes $\Theta(n)$ time.

A Sorted Array with Sidekick (SAwS) combines a sorted array with a smaller unsorted array, the sidekick. New items inserted in a SAwS are added to the sidekick array, which takes $O(1)$ actual time since the sidekick is not sorted. When the number of items in the sidekick grows beyond a parameter $t(n)$, the sidekick is sorted and merged into the sorted array. After this merger, the new sidekick is empty. To search in a SAwS simply do a binary search in the main array and a linear search in the sidekick.

Instructions:

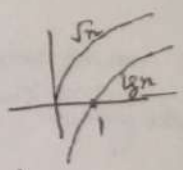
- Throughout this problem, let n be the number of items currently stored in the SAwS. This means n changes if items are added to or removed from the SAwS. You should analyze the running times in terms of n.
- Do not assume anything about the keys stored in the SAwS — i.e., you know nothing about the type, distribution or range of the keys. In particular, do not use counting sort, bucket sort, radix sort or any of the $O(n)$ time sorting algorithms.
- You may assume without further comment that in the worst case sorting m items takes $\Theta(m \log m)$ time, binary search in a sorted array of m items takes $\Theta(\log m)$ time, finding the median of an unsorted array of m items takes $\Theta(m)$ time and merging two sorted arrays of m and ℓ items takes $\Theta(m + \ell)$ time.
- You may assume that your programming environment allows you to store `undefined` in an array element to indicate that no item is stored there.
- Do not write pseudo-code. If you want to sort an array A, just write "Sort the array A."
- Be brief. Start writing only *after* you have thought through your solution. Concise and correct answers are the best answers.

Questions:

a. (5 points) Recall that the parameter $t(n)$ specifies the maximum allowable size of the sidekick array. Suppose that we choose $t(n) = \sqrt{n}$. Briefly describe how the sidekick array can be merged into the sorted array in $\Theta(n)$ actual time.

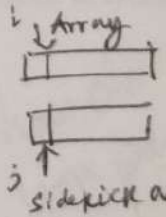
5

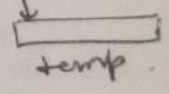
① Sort the sidekick array: will take $\theta(\sqrt{n}\lg\sqrt{n}) = \theta(\sqrt{n}\lg n)$ time.
Now, ~~$\sqrt{n}$~~ $\lg n = O(\sqrt{n}) \Rightarrow$ sorting the array takes $O(n)$ time.



$\sqrt{n}\lg n$
$= O(\sqrt{n}\cdot\sqrt{n})$
$= O(n)$

② Merge the two sorted arrays (using "Merge" of mergesort)
It takes $\theta(n+\sqrt{n}) = \theta(n)$ time ✓





~~and temp~~ → take a temp array of size $n+\sqrt{n}$

→ initially assign 2 pointers to the beginning of the two arrays (i, j) $(i, j, k \leftarrow 0)$

Loop do these steps until one of the arrays exhaust:

→ if array[i] < SidekickArray[j], temp[k] ← array[i], ~~k ← k+1~~ i ← i+1

→ else temp[k] ← SidekickArray[j], j ← j+1

→ k ← k+1

You don't have to explain how merging works

→ When done copy the rest of the remaining array to temp array.

→ Now the temp contains the ~~sorted arr~~ sorted version of the merged arrays.

① + ② takes $O(n) + \theta(n) = \mathbf{O}(n)$ time.

b. (10 points) Using either the accounting method or the potential method, show that the insertion and search operations in a SAwS can be accomplished in $O(\sqrt{n})$ amortized time. Here $t(n) = \sqrt{n}$.

10

Insertion: when size(sidekick array) $< \sqrt{n}$, $O(1)$

(no insert only in the sidekick array)

size(sidekick array) $= \sqrt{n}$, we need to merge.

If while inserting ($\sqrt{n}+1$) we cost every insertion operation a charge of c units, ~~one unit can be used for insertion,~~ another 1 unit can be used for merging of that element & we notice that merging happens only when # insertion instructions is a multiple of $\sqrt{n}$, i.e., when the $k\sqrt{n}$ th element is to be inserted where $k = 1, 2, 3, \cdots$, now we have at $\sqrt{n}$ th instruction total charge stored $= c(\sqrt{n} + \sqrt{n} + \cdots + \sqrt{n}) = c\sqrt{n}$, which is enough to pay for merge. But $c\sqrt{n}+1 = O(\sqrt{n})$, hence amortized time $O(\sqrt{n})$.

($\sqrt{n}$ times)

1 unit is used for inserting

$c \geq 2$, a const.

any arbitrary combination of insert & search will be also $O(\sqrt{n})$

Similarly, charge each search by $c\sqrt{n}$ units, it can always pay for the actual cost $\theta(\sqrt{n} + \lg n)$ since $\lg n = O(\sqrt{n})$

8

c. (10 points) Describe how to implement the delete operation in a SAwS so that search, insert and delete each takes $O(\sqrt{n})$ amortized time. (Again, $t(n) = \sqrt{n}$.) Justify the amortized analysis of the running times using either the accounting method or the potential method. Note that a delete reduces the value of n and of $t(n)$.

Delete operation: the element $x \in A$ (original array).

$\Rightarrow$ search is $\theta(\lg n)$.

the element $x \in SA$ (sidekick array)

$\Rightarrow$ search is $\theta(\sqrt{n})$.

deleting might cause sidekick to be too large.

-1

search x first. If $x \in A$, replace x by any arbitrary element from SA, if SA is non-empty. Else, shift the array to left

-1

very expensive how is this part?

If $x \in SA$, just swap x with the last element of SA and delete the last element : $\theta(1)$.

If we charge each delete by $c\sqrt{n}+1$, we can as above show that this covers every charge

3

d. (5 points) If we choose $t(n) = \log n$, we would reduce the actual time of search to $O(\log n)$. What would happen to the amortized running time of insert and delete?

amortized run time of insert $= O(\lg n)$

" " " delete $= O(\lg n)$

how do you pay for merges?

-2