CSE 5311 Section 003 Fall 2004

Quiz 2 Solution
September 22, 2004

This is a closed-book, closed-notes, closed-computer, closed-neighbor quiz.

1. Consider the problem of sorting, in worst-case linear time, an array $A$ of 10,000 9-digit social security numbers in increasing order. For each of the sorting algorithms below, indicate whether or not the algorithm will achieve worst-case, linear-time performance, and briefly explain why or why not.

(a) (2 points) CountingSort
No. CountingSort does not achieve worst-case linear time for this problem. CountingSort achieves worst-case linear time only when the largest element $k$ in the array is proportional to the length of the array $n$, i.e., $k = O(n)$. However, $k = \Theta(10^9)$ and $n = 10^5$, so $k \neq O(n)$ and CountingSort does not achieve worst-case linear time.

(b) (2 points) RadixSort
Yes. RadixSort achieves worst-case linear time, because the number of digits $d = 9$ is a constant, regardless of the length of the array $n$. Therefore, the worst case running time is $\Theta(dn) = \Theta(n)$.

(c) (2 points) BucketSort
No. BucketSort does not achieve worst-case linear time for two reasons. First, BucketSort only guarantees average-case linear time, not worst-case. Second, the numbers to be sorted are not between 0 and 1.

(d) (2 points) QuickSort
No. QuickSort is a comparison sort, which we have proven to be $\Omega(n \lg n)$ in the worst-case, regardless of the input.

2. (3 points) Explain how we can use the linear-time Select2 algorithm to implement a version of QuickSort whose worst-case running time is $\Theta(n \lg n)$.

Since Select2 can find the $i^{th}$ smallest element in an array of size $n$ in $O(n)$ time, we can use Select2 within QuickSort to find the median ($i = n/2$) of the array to be sorted. If we then use this median as the pivot in the Partition algorithm, we are guaranteed to achieve balanced partitioning, which leads to QuickSort’s optimal $\Theta(n \lg n)$ running time.
3. Consider a hash table of size $m = 12$ that uses collision-resolution by open addressing and the quadratic probing hash function $h(k, i) = ((k \mod m) + i + i^2) \mod m$.

(a) (4 points) Show the hash table resulting from inserting the keys 10, 22, 34 and 16, in this order.

```
<table>
<thead>
<tr>
<th>0</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>34</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>16</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>
```

- $h(10, 0) = 10$
- $h(22, 0) = 10$ collision
- $h(22, 1) = 0$
- $h(34, 0) = 10$ collision
- $h(34, 1) = 0$ collision
- $h(34, 2) = 4$
- $h(16, 0) = 4$ collision
- $h(16, 1) = 6$

(b) (4 points) The hash function for this hash table does not generate a valid probe sequence. Explain why and give a key that cannot be inserted into the table you produced in part (a) above.

In general, a quadratic probing hash function will yield an invalid probe sequence whenever $(c_1 i + c_2 i^2) = m$ for some value of $i < m - 1$. Any values computed at this choice for $i$ will be equal to the values computed for $i = 0$. In the specific case of this hash function, $i + i^2 = m$ for $i = 3$; therefore, the probe sequence will try at most three different locations before wrapping back to the $i = 0$ value.

So, once we fill up the $i = 0, 1, 2$ positions of a probe sequence starting at $h(k, 0)$, then attempting to insert a fourth key $k'$, such that $h(k', 0) = h(k, 0)$, the probe sequence will only visit the three positions already filled over and over again, and the new key cannot be inserted. For example, inserting key 46 into the above hash table will result in cycling among the hash values of 10, 0, and 4, which are all full. In fact, any key of the form $12x + 10$ will yield the same result.
4. Consider the following valid red-black tree, where “R” indicates a red node, and “B” indicates a black node. Note that the black dummy sentinel leaf nodes are not shown.

(a) (3 points) Show the resulting red-black tree after using RB-INSERT to insert key 3 into the above tree.
This is Case 3 of RB-INSERT.

(b) (3 points) Show the resulting red-black tree after using RB-DELETE to delete key 15 from the original tree above.
This is Case 4 of RB-DELETE-FIXUP.