Design & Analysis of Algorithms - CS 4/56101

Homework 4

Put each of the problems on a separate sheet of paper, and make sure your name is on each sheet. To hand them in, staple them together and bring them to the class on due date. Please make sure to express your algorithms in pseudo-code when directed (see the appropriate section of the course textbook for the proper pseudo-code style), and always provide justification for your answer when asked to give the running time of an algorithm. Be brief and concise, and draw pictures where appropriate.

Problem 1.

You have a hash table of size $m = 11$ and two hash functions $h_1$ and $h_2$:

$h_1(x) = \text{(sum of the values of the first and last letters of } x) \mod m$

$h_2(x) = ((\text{value of the last letter}) \mod (m-1))+1$

where the value of a letter is its position in the alphabet (e.g., value(a)=1, value(b)=2, etc.).

Here are some precomputed hash values:

<table>
<thead>
<tr>
<th>word</th>
<th>ibex</th>
<th>hare</th>
<th>ape</th>
<th>bat</th>
<th>bird</th>
<th>carp</th>
<th>dog</th>
<th>mud</th>
<th>koala</th>
<th>stork</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1$</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>$h_2$</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>10</td>
<td>5</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

1. Draw a picture of the resulting hash table after inserting, in order, the following words:

ibex, hare, ape, bat, koala, mud, dog, carp, stork.

**solution:** It depends on which hash technique you use. You should specify it explicitly in your answer. Please refer to the second question for three hash tables resulting from three different hash techniques.

2. Highlight cells that are looked at when trying to find bird. Do this for each of the following techniques. Chaining with $h_1$ as your hash function. Linear probing with $h_1$ as your hash function. Double hashing with $h_1$ as your first hash function and $h_2$ as your second hash function.
solution:
(1) Chaining with $h_1$ as your hash function

(2) Linear probing with $h_1$ as your hash function

(3) Double hashing with $h_1$ as your first hash function and $h_2$ as your second hash function
Problem 2.

(a) Draw the merge tree for an execution of the merge-sort algorithm on
the input sequence (6, 9, 20, 8, 14, 27, 43, 22, 30, 19). solution:

(b) Draw the quick-sort tree for an execution of the quick-sort algorithm
on the input sequence from (a).

solution:

Suppose we modify the deterministic version of the quick-sort algorithm
so that, instead of selecting the last element in an n-element sequence as the
pivot, we choose the element at rank \( \lfloor n/4 \rfloor \).
(c) Draw the quick-sort tree for an execution of this modified quick-sort algorithm on the input sequence from (a). solution:

(d) What is the running time of this version of quick-sort on a sequence that is already sorted?

solution: On average it is still $O(n \log n)$.

Problem 3.

(a) Suppose the entering freshmen class at some university has N students. The information pertinent to each student in the class (name, sex, identification number, etc.) can be found in some element of A, an array of records indexed from 1 to N. Assume the records are in some random order, and that we wish to rearrange the array so that all the female records precede all the male records. Give a linear ($O(N)$) time algorithm which performs this "partial sort" on A. Assume "sex" is a field in each record, with value "Male" or "Female".

solution: The following algorithm sorts the students in the university freshman class by sex (females first). The algorithm requires $O(N)$ time. I assume that there is at least one male and one female in the array of students (which is denoted by A).

Initialize pointer ‘MalePointer’ to 1
Initialize pointer ‘FemalePointer’ to \( N \)

While MalePointer is less than FemalePointer

Begin

While MalePointer points to a female’s record in \( A \)
  Increment MalePointer by 1
While FemalePointer points to a male’s record in \( A \)
  Decrement FemalePointer by 1
If MalePointer is less than FemalePointer then
  Swap \( A[\text{MalePointer}] \) with \( A[\text{FemalePointer}] \)

End

Because of the nested loops, it may not be obvious that this is an \( O(N) \) algorithm. However, the basic operation of the algorithm is pointer movement. The \( \text{MalePointer} \) moves only from smaller to larger indices, while \( \text{FemalePointer} \) moves from larger to smaller. Since the algorithm terminates when the pointers have crossed, essentially \( N \) pointer changes (decrements or increments) are performed. With each pointer increment or decrement, there is at most a constant number of additional steps (the swap requires only constant time). Therefore, the time complexity of the algorithm is some constant times the number of pointer alterations, which is \( N \). Thus it is an \( O(N) \) algorithm.

(b) Let \( A \) and \( B \) be two sequences of \( n \) integers each. Given an integer \( m \), describe an \( O(n \log n) \) time algorithm for determining if there is an integer \( a \) in \( A \) and an integer \( b \) in \( B \) such that \( m = a - b \).

solution: This algorithm is going to use a sorting algorithm — any sorting algorithm that runs in \( O(n \log n) \) worst-case time would do (say, merge-sort).

The following is a general outline of the algorithm followed by pseudocode.

- First we compute a new sequence \( C \) of size \( n \) such that for each \( i \) between 0 and \( n - 1 \), \( C[i] = m + B[i] \). This takes \( O(n) \) time.
- We sort \( C \) (this takes \( O(n \log n) \) time)
- We sort \( A \) (this also takes \( O(n \log n) \) time)
- We scan arrays \( C \) and \( A \) from the beginning each looking for similar elements. Now that the arrays are sorted, we’d have to do at worst \( 2n \) comparisons. Therefore this step takes \( O(n) \) time.
• if $C$ and $A$ have an element in common (say, $a$), then there’s an element $b$ in $B$, such that $a = m + b$. Note that we are not required to find such element by the statement of the problem — just to confirm that such element exists.

So the running time of this algorithm is going to be $O(n + n \log n + n \log n + n) = O(n \log n)$.

Algorithm IsThereASum $(A, B, n, m)$

Input: array $A$; array $B$; size $n$ of $A$ and $B$; number $m$

Output: whether there is exists $a \in A$ and $b \in B$ such that $a - b = m$

$C$ is a new array of size $n$

for $i \leftarrow 0$ to $n$ do

$C[i] \leftarrow m + B[i]$

Sort($C$)

Sort($A$)

$cind \leftarrow 0$

$aind \leftarrow 0$

while $(cind \neq n$ or $aind \neq n)$ do

if $(C[cind] = A[aind])$ then

return true

if $(C[cind] \leq A[aind])$ then

$cind \leftarrow cind + 1$

else

$aind \leftarrow aind + 1$

return false

Problem 4. The scientists of Space Lab want to sort their remote sensing data to reconstruct the 3-D images of the surface of Mars. They have one million 64-bit integers to sort.

1. If they use quicksort, estimate the number of key value comparisons performed. (Hint: consider the average time complexity of quicksort.)

solution: Quicksort takes average time $O(n \log n)$ time. Hence, an estimate of the number of comparisons performed is $1,000,000 \log_2 1,000,000 \simeq 20,000,000$.

2. If they treat these integers as four-digit, radix-$2^{16}$ numbers and use straight radix sort based on the radix-$2^{16}$ notation, how many sorting passes are required to do the sorting?
solution: Since there are four digits in this notation, four sorting passes are needed.

3. Unfortunately, the computer in the Space Lab has only 9 megabytes of available memory for running the sorting task. Is it practical to choose bucket sort to be the stable sorting method in the implementation of the above straight radix sort? Explain your answer.

solution: The available memory space is 9 megabytes. Since about 8 megabytes are needed to store one million 64-bit integers (64 bits = 8 bytes), only one megabyte of memory is left. Since bucket sort takes additional storage space to perform its task, it is not suitable for this case. The scientists should find a stable sorting method that sorts numbers in place!

Problem 5. Superfast Software Inc. was founded last year by three young programmers. They all dreamed their company would become a really big one and would distribute a large number of software products all over the world. Thus, they decided to use 64-bit integers to represent their inventory codes. Since it is just a one-year-old company, the inventory database now contains only 2000 distinct product codes, in the range from 1 to 3000. At this time they need to sort these codes and one of the co-founders suggests using a general comparison-based $O(n \log n)$-time sorting algorithm such as heap-sort. But another co-founder disagrees and suggests using radix-exchange sort because it is a so-called "linear time" (that is, $O(n)$) algorithm. Do you think radix exchange sort is good for this case? Explain your answer.

solution:

It’s dangerous to evaluate the practical performance of an algorithm solely by the big-Oh notation.

In this problem, where we use 64-bit integers to represent inventory codes in the range 1 to 3000, radix exchange sort may be inferior to quicksort, even though its time complexity is $O(64n) = O(n)$. Namely, radix-exchange sort performs about $b \times n = 64 \times 2000$ bit comparisons. Instead, in quicksort about $n \log n = 2000 \times 20$ integer comparisons are performed.

In most computers, integer comparisons take the same time as bit comparisons, so that quicksort is faster.

Note that in this problem, all the leftmost 52 bits are zeros, and only the rightmost 12 bits are different. This means that radix exchange sort performs 52 useless phases over the leftmost 52 bits.