CS3343 Analysis of Algorithms Fall 2017

Homework 4

Due 10/15/17 before 11:59pm (Central Time)

1. Heapsort (4 points)

Originally we stored our heap in an array. Consider instead storing our heap as a doubly linked list.

(1) (2 points) For a node i what are the new asymptotic run times for $left(i)$, $right(i)$, and $parent(i)$? Justify your answer.
For each operation you calculate the index you need using the appropriate formulas then return the pointer to the calculated index by traversing there using the difference between i and the calculated index. Thus given i, then each operation is $O(2i+1-i)$, $O(2i+2-i)$, $O(i-(\lfloor(i-1)/2\rfloor))$ respectively. Ignoring constants they are all $O(i)$.

(2) (2 points) How does this affect the run times of findMax(), insert(key), extractMax()? Justify your answer.
findMax() will still be $\Theta(1)$ since the first element will still be the maximum. For insert(key), the list must first be traversed which takes $\Theta(n)$ time, then in the worst case key is greater than all other elements and will take $O(\sum_{i=0}^{logn} \lfloor(i-1)/2\rfloor) \approx O(\frac{log^2 n}{4})$, since you must check each elements parent as you traverse up the list, thus insert(key) will take $\Theta(n)$ worst case run time since you always traverse the entire list to start at the end when inserting. Finally, for extractMax() using similar logic, it is easy to see that it will also be $\Theta(n)$ runtime since you must traverse the entire last to get the element that will replace the extracted element at the top of the heap.

2. Counting Sort (4 points)

(1) (2 points) Illustrate the operation of $countingSort(\{2,1,5,3,1,2,5\},6)$. Specifically, show the changes made to the arrays A, B, and C for each pass through the for loop at line 16.

Algorithm 1 void countingSort(int $A[1 \dots n]$, int k)

```
//Precondition: The $n$ values in $A$ are all between 0 and $k$
Let $C[0 \dots k]$ be a new array
//We will store our sorted array in the array $B$.
Let $B[1 \dots n]$ be a new array
**for** $i = 0$ **to** $k$ **do**
    $C[i] = 0$;
**end for**
**for** $i = 1$ **to** $n$ **do**
    $C[A[i]] = C[A[i]] + 1$;
**end for**
//$C[i]$ now contains the # of elements in $A$ equal to $i$
**for** $i = 1$ **to** $k$ **do**
    $C[i] = C[i] + C[i-1]$;
**end for**
//$C[i]$ now contains the # of elements in $A$ that are $\leq$ to $i$
**for** $i = n$ **down to** 1 **do**
    $B[C[A[i]]] = A[i]$;
    $C[A[i]] = C[A[i]] - 1$;
**end for**
return $B$;
```

A remains the same through the algorithm.

2	1	5	3	1	2	5

Now each line will show B and C at each pass through the loop.

B→	0	0	0	0	0	0	0	C→	0	2	4	5	5	7	7
B→	0	0	0	0	0	0	5	C→	0	2	4	5	5	6	7
B→	0	0	0	2	0	0	5	C→	0	2	3	5	5	6	7
B→	0	1	0	2	0	0	5	C→	0	1	3	5	5	6	7
B→	0	1	0	2	3	0	5	C→	0	1	3	4	5	6	7
B→	0	1	0	2	3	5	5	C→	0	1	3	4	5	5	7
B→	1	1	0	2	3	5	5	C→	0	0	3	4	5	5	7
B→	1	1	2	2	3	5	5	C→	0	0	2	4	5	5	7

(2) (2 points) Describe an algorithm that, given an unsorted array of n integers in the range 0 to k, preprocesses its input and then answers any query about how many of the n integers fall into a range a to b in $O(1)$ time. Your algorithm should use $O(n+k)$ preprocessing time.
(Hint: Look at the array C which is computed by the above code for inspiration)

Algorithm 2 int countInRange(int $A[1 \ldots n]$, int a, int b)

```
//Precondition: The $n$ values in $A$ are all between 0 and $k$
Let $C[0 \ldots k]$ be a new array
for $i = 0$ to $k$ do
    $C[i] = 0$;
end for
for $i = 1$ to $n$ do
    $C[A[i]] = C[A[i]] + 1$;
end for
//$C[i]$ now contains the # of elements in $A$ equal to $i$
for $i = 1$ to $k$ do
    $C[i] = C[i] + C[i-1]$;
end for
//$C[i]$ now contains the # of elements in $A$ that are $\leq$ to $i$
int count;
if $a = 0$ then
    $count = C[b]$;
else
    $count = C[b] - C[a-1]$;
end if
return count;
```

3. Hash Table (7 points)

(1) Consider inserting the keys $2, 21, 3, 58, 11, 42, 34$ into a hash table of length $m = 10$ with the hash function $h(k) = k \bmod 10$.

(a) (2 points) Illustrate the result of inserting these keys using linear probing to resolve collisions.

0	1	2	3	4	5	6	7	8	9
	21	2	3	11	42	34		58	

(b) (2 points) Illustrate the result of inserting these keys using chaining to resolve collisions.

0	1	2	3	4	5	6	7	8	9
	21	2	3	34				58	
	11	42							

(2) Consider inserting the keys $8, 5, 14$ into a hash table of length $m = 8$ with the hash function $h(k) = \lfloor m(kA - \lfloor kA \rfloor) \rfloor$ where $A = 0.625$.

(a) (2 points) Illustrate the result of inserting these keys.

0	1	2	3	4	5	6	7
8	5					14	

(b) (1 point) Now compute the hash function of the key 14 using the implementation we described in our notes.
You can assume we have a word size $w = 4$. Since $m = 8 = 2^3$, $p = 3$. Since $A = 0.625 = 10/2^4 = 10/2^w$, $s = 10$.
(Hint: Compute ks and convert it to a binary number. This number will consist of $\leq 2w$ bits. Look at the rightmost w bits. Of those bits, convert the leftmost p bits back to an integer. This integer is your hash table slot.)

$k \cdot s = 14 \cdot 10 = 140 = 0100\ 1100_2$ with $2w = 8$ bits
the first $p = 4$ bits are 1100
$1100_2 = 6_{10}$ so the hash slot is 6