CS3343 Analysis of Algorithms Fall 2019

Homework 4

Due 10/14/19 before 11:59pm (Central Time)

1. Arrays and Heaps (6 points)

(1) (3 points) Suppose you wanted to create an algorithm to find the number of elements $\geq k$ in an array, A, of n distinct elements.

Specifically, describe efficient algorithms to solve this problem when A is the following data structures.

(a) Unsorted array — loop through the array and check if element is bigger
(b) Sorted array — Since the array is sorted we can do a binary search and find k, then we can say number on right will be strictly $> k$
(c) Max-heap — Check if root $= k$ if true there are no element $> k$

(2) (3 points) Analyze the runtime of your three algorithms from the previous part:

(a) Unsorted array — $O(n)$ time because we have to loop through each element to see if 43.2k
(b) Sorted array — $O(\log n)$ time because each time we divide array by half
(c) Max-heap — $O(n \log n)$

2. Heaps With Linked Lists (4 points)

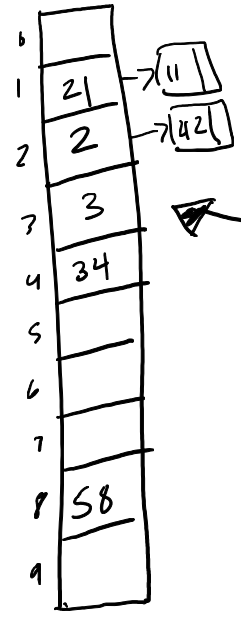
Originally we stored our heap in an array. Consider instead storing our heap as a doubly linked list.

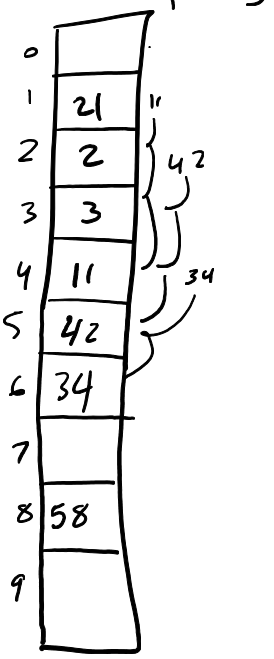
(1) (2 points) For a node i what are the new asymptotic run times for $left(i)$, $right(i)$, and $parent(i)$? Justify your answer.
(2) (2 points) How does this affect the run times of findMax(), insert(key), extractMax()? Justify your answer.

findMax(): $O(1)$ for max-heap because 1st one would be largest one
insert(key): would take $O(n)$ times because we would have to loop n times in worst case.
extractMax(): 3.0

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





(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.

Continued on the back ↪

$h(8) = \lfloor 8(8(0.625) - \lfloor 8(0.625) \rfloor) \rfloor = 0$

$h(5) = \lfloor 8(5(0.625) - \lfloor 5(0.625) \rfloor) \rfloor = 1$

$h(14) = \lfloor 8(14(0.625) - \lfloor 14(0.625) \rfloor) \rfloor = 6$

3.125

(b) (1 point) Now compute the hash function of the key 14 using the alternative algorithm described below.

You can assume we have a word size $w = 4$. Since $m = 8 = 2^3$, $p = 3$. Since $A = 0.625 = 10/2^4 = 10/2^w$, $s = 10$.

Alternative Algorithm: 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.

4. Counting Sort (4 points)

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

```
1: //Precondition: The n values in A are all between 0 and k
2: Let C[0...k] be a new array
3: //We will store our sorted array in the array B.
4: Let B[1...n] be a new array
5: for i = 0 to k do
6:     C[i] = 0;
7: end for
8: for i = 1 to n do
9:     C[A[i]] = C[A[i]] + 1;
10: end for
11: //C[i] now contains the # of elements in A equal to i
12: for i = 1 to k do
13:     C[i] = C[i] + C[i − 1];
14: end for
15: //C[i] now contains the # of elements in A that are ≤ to i
16: for i = n down to 1 do
17:     B[C[A[i]]] = A[i];
18:     C[A[i]] = C[A[i]] − 1;
19: end for
20: return B;
```

(a) (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.

(b) (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)

4.a $A = \{2,1,5,3,1,2,5\}$

$i=7$ $B = \{\_\ \_\ \_\ 5\ \_\}$ $C = \{2,4,5,5,6,7\}$

$i=6$ $B = \{\_\ \_\ \_\ 2,1,5\}$ $C = \{2\ 3\ 5\ 5\ 6\ 7\}$

$i=5$ $B = \{\_\ 1\ \_\ 2\ \_\ \_\ 5\}$ $C = \{1\ 3\ 5\ 5\ 6\ 7\}$

$i=4$ $B = \{\_\ 1\ \_\ \_\ 2\ \_\ 3\ \_\ 5\}$ $C = \{1\ 3\ 4\ 5\ 6\ 7\}$

$i=3$ $B = \{\_\ 1\ \_\ \_\ 2\ \_\ \_\ 3,5,5\}$ $C = \{1\ 3\ 4\ 5\ 5\ 7\ 5$

$i=2$ $B = \{1\ 1\ 2\ 2\ 3\ 5\ 5\}$ $C = \{0\ 3\ 4\ 5\ 5\ 7\}$

$i=1$ $B = \{1\ 1\ 2\ 2\ 3\ 5\ 5\}$ $C = \{0\ 2\ 4\ 5\ 5\ 7\}$

4b Loop through the array and compare if each element is in between the range of a and b and increase the count.