Midterm II Exam
15-122 Principles of Imperative Computation
Anupam Datta, André Platzer, Saquib Razak
April 8, 2014

Name: __________________________________________
Andrew ID: _____________________________________

Instructions

• This exam is closed-book with one sheet of notes permitted.
• You have 80 minutes to complete the exam.
• There are 4 problems on 17 pages, including one extra sheet.
• Read each problem carefully before attempting to solve it.
• Do not spend too much time on any one problem.
• Consider if you might want to skip a problem on a first pass and return to it later.
• You can assume the presence of #use <util> throughout the exam.

<table>
<thead>
<tr>
<th></th>
<th>Max</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amortization</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Hash Tables</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>Heaps and Priority Queues</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>AVL Trees</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>125</td>
<td></td>
</tr>
</tbody>
</table>
1 Amortization (20 points)

Consider a $k$-bit counter for which it costs 1 token to flip a bit. In this problem, you will assess the worst case time complexity of a single increment operation and the amortized time complexity of performing $n$ increment operations of this counter, where $n = 2^k$. State your answers in terms of $k$ and $n$.

6 Task 1 The worst-case complexity of a single increment operation is: $O(k)$. Explain your answer briefly.

2 Task 2 How many tokens are required to flip bit 0 (the rightmost bit) of the counter through $n$ increments?

2 Task 3 How many tokens are required to flip bit 1 (the second bit from the right) of the counter through $n$ increments?
Task 4 How many total tokens are required to perform $n$ increment operations? Justify your answer by showing how you compute the number of tokens.

Therefore, the amortized cost of a single increment operation is: $O(\ 1 \ )$. 
2 Hash Tables (45 points)

In this question we study hash tables as implemented in class, where new elements are inserted at the beginning of the linked list. In the diagrams below, we only show integer keys.

Task 1
Draw the final state of the hash table data structure after inserting the following sequence of elements into it:

4, 2, 8, 14

with the following hash function implementation:

```c
int hash(int x) {
    return x;
}
```

![Diagram of hash table](image)

Task 2
Give a definition of a hash function of complexity $O(1)$ and a sequence of elements to insert into the hash table that results in the following final state:

```
int hash(int x) {
    return _________________________________;
}
```

When inserting the following sequence of elements:
10 **Task 3** Give a definition of a hash function of complexity $O(1)$ and a sequence of elements to insert into the hash table that results in the following final state:

![Hash Table Diagram](image)

```c
int hash(int x) {
    return _____________________________________________________________;
}
```

When inserting the following sequence of elements:

10 **Task 4** Give a definition of a hash function of complexity $O(1)$ and a sequence of elements to insert into the hash table that results in the following final state:

![Hash Table Diagram](image)

```c
int hash(int x) {
    return _____________________________________________________________;
}
```

When inserting the following sequence of elements:

3 **Task 5** What is the load factor of the hash table in Task 4? Express your answer as a fraction.

6 **Task 6** Is there a sequence of 12 elements to insert that results in the final table in Task 4? Provide such a sequence or explain briefly (1–2 sentences) why it cannot exist.
3 Heaps and Priority Queues (40 points)

This problem will involve using min-heaps and priority queues implemented using an array representation as studied in class:

```c
struct heap_header {
    int limit; /* limit = capacity+1 */
    int next; /* 1 <= next && next <= limit */
    int[] data; /* \length(data) == limit */
};
typedef struct heap_header* heap;

heap heap_new(int capacity) /* create new heap of given capacity */
//@requires capacity > 0;

bool is_heap(struct heap_header* H) /* Specification function to check heaps*/

One way of taking an existing array and converting it to a heap is to go to each non-leaf node, and check if the key of at this node is less or equal to all of its childrens’ keys. If this ordering invariant is not met, then fix the heap by sifting down from this node until a leaf node is reached or heap ordering invariant is met. Repeat the above for all nodes starting from the last non-leaf node and working backwards until the root node at index 1.

The code on the next page is given as a sample implementation. Refer to it while answering questions in this section.

The outside for loop iterates through each non-leaf node starting from node at index (n-1)/2 (the last non-leaf node) and working backwards until the root node at index 1. The inner while loop checks if the heap ordering invariant is met at the current node and shifts down if the invariant is not satisfied.

Note: A copy of the same algorithm is on the last page of this exam that you can rip off.
1. heap heapify(int[] elements, int n)
2. //@requires \length(elements) == n;
3. //@ensures is_heap(result);
4. {
5.  heap H = heap_new(n)
6.  //@assert \length(H->data) == n;
7.  for(int k = 0 ; k < n; k++)
8.  //@loop_invariant k >= 0;
9.  {
10.     H->data[k] = elements[k];
11.  }
12.  H->next = n;
13.  for(int i = (n-1)/2; i > 0; i--)
14.      //@loop_invariant i*2 < H->next;
15.      {
16.          int j = i;
17.          int left = 2*j;
18.          int right = left+1;
19.          while(left < n)
20.              //@loop_invariant 1 <= j && j < n;
21.              //@loop_invariant left == 2*j;
22.              //@loop_invariant right == 2*j+1;
23.              {
24.                  if (H->data[j] > H->data[left] &&
25.                      H->data[left] < H->data[right]) {
26.                      swap(H->data, j, left);
27.                      j = left;
28.                  }
29.                  else if (right < n &&
31.                      swap(H->data, j, right);
32.                      j = right;
33.                  }
34.                  else {
35.                      j = n-1;
36.                  }
37.              }
38.          left = 2*j;
39.          right = left+1;
40.      }
41. //Contents of H->data should be shown at this line in task 2
42.  return H;
43.}
Task 1 For the code given for function heapify, specify which of the following array access are safe. If an array access is safe, indicate line numbers that guarantee that safety. If the array access is not protected, write unsafe access.

1. Line 10: H->data[k]  _______________________________


3. Line 24: H->data[left]  _______________________________


Task 2 Let’s say that the heapify function is called with the array shown below. Show the value of variable \( i \) and the contents of \( H->data \) at the end of each iteration of the for loop in the table below (state of \( H->data \) on line 40. Also draw the final heap \( r \) that will be returned by the function.

Note: This function produces a correct min-heap when called with the given array.

```c
int size = 10;
int[] array = alloc_array(int, size);
array[1] = 20;
array[2] = 5;
array[3] = 11;
array[4] = 15;
array[5] = 16;
array[6] = 9;
array[7] = 7;
array[8] = 8;
array[9] = 10;
heap r = heapify(array, size);
```

<table>
<thead>
<tr>
<th>( i )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>( H-&gt;data )</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Draw your heap here:

```
5
/  \
8 7
/  \
10 16 9 11
/  \
15 20
```
Task 3 In the given code, the else statement on line 34 allows us to exit the while loop if the current node is already less than both of its children. Hence, in the best case, for each iteration of the for loop on line 13, the while loop executes exactly one time. This says that the best case complexity of this code is $O((n-1)/2)$. Give an example of an array below that will achieve this best cases complexity.

```c
int size = 10;
int[] array = alloc_array(int,size);

array[1] = __________;
array[2] = __________;
array[3] = __________;
array[4] = __________;
array[5] = __________;
array[6] = __________;
array[7] = __________;
array[8] = __________;
array[9] = __________;
heap r = heapify(array,size);
```

Task 4 In order to measure the time complexity of the above algorithm experimentally, I measured the time it took to complete heapify on several different input sizes with randomly generated values. The following table shows the times measured for each input size. Based on this data what would be the time complexity of the given algorithm?

<table>
<thead>
<tr>
<th>n</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10,000</td>
<td>9.429 ms</td>
</tr>
<tr>
<td>20,000</td>
<td>18.769 ms</td>
</tr>
<tr>
<td>40,000</td>
<td>37.666 ms</td>
</tr>
<tr>
<td>80,000</td>
<td>75.151 ms</td>
</tr>
<tr>
<td>160,000</td>
<td>150.687 ms</td>
</tr>
</tbody>
</table>

```
Complexity = O(__________)
```
4 AVL Trees (20 points)

In this problem, we will look at AVL trees that are balanced Binary Search Trees as studied in class.

5 Task 1 For each of the following trees, indicate if the tree meets all specifications of an AVL tree.

Is the above tree an AVL tree? Justify your answer:

Is the above tree an AVL tree? Justify your answer:
Task 2 Draw the AVL tree, that will result after the given elements are inserted in the tree below:

Insert 12:

Insert 15 in the tree resulting from inserting 12 in the original tree:
Task 3 Give an example of element(s), that when inserted in the following AVL tree, will cause a double rotation.

Write your answer here:
COPY OF ALGORITHM FROM QUESTION 3

1. heap heapify(int[] elements, int n)
2. //@requires \length(elements) == n;
3. //@ensures is_heap(result);
4. {
5.   heap H = heap_new(n)
6.   //@assert \length(H->data) == n;
7.   for(int k = 0 ; k < n; k++)
8.   //@loop_invariant k >= 0;
9.   {
10.      H->data[k] = elements[k];
11.   }
12.   H->next = n;
13.   for(int i = (n-1)/2; i > 0; i--)
14.   //@loop_invariant i*2 < H->next;
15.   {
16.      int j = i;
17.      int left = 2*j;
18.      int right = left+1;
19.      while(left < n)
20.      //@loop_invariant 1 <= j && j < n;
21.      //@loop_invariant left == 2*j;
22.      //@loop_invariant right == 2*j+1;
23.      {
24.         if (H->data[j] > H->data[left] &&
25.            H->data[left] < H->data[right]){
26.            swap(H->data, j, left);
27.            j = left;
28.         }
29.         else if (right < n &&
31.            swap(H->data, j, right);
32.            j = right;
33.         }
34.         else{
35.            j = n-1;
36.         }
37.         left = 2*j;
38.         right = left+1;
39.      }
40.      //Contents of H->data should be shown at this line in task 2
41.   }
42.   return H;
43.}