CSE373 Midterm II

Fall 2009

(Closed book, closed notes)

Name: ....

Problem	Points
Q1 [20 points]	
Q2 [20 points]	
Q3 [15 points]	
Q4 [15 points]	
Q5 [30 points]	
Total [100 points]	

1 [20 points] Heaps

1. Consider a binary heap with $n = 2^k - 1$ elements, stored in an array `a[1], ..., a[n]`, using the implicit representation. The heap supports the operations `insert` and `deleteMin`, that is, the smallest element is `a[1]`. For each of the statements below indicate whether they are true or false.

   (a) The largest element in the heap is `a[n]`.

   True or False ? False

   (b) `a[1]+a[2]+...+a[n/2]` $\leq$ `a[n/2+1]+a[n/2+2]+...+a[n]`

   True or False ? True

   (c) The median element in the heap is either `a[n/2]` or `a[n/2+1]`.

   True or False ? False

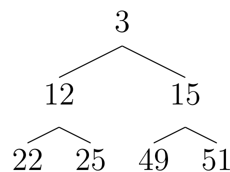
   (d) `a[1]` $\leq$ `a[2]` $\leq$ `a[4]` $\leq$ `a[8]` $\leq \ldots \leq$ `a[`2^{k-1}`]`

   True or False ? True

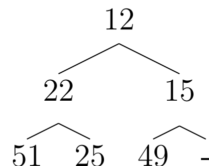
   (e) The second smallest element is either `a[2]` or `a[3]`.

   True or False ? True

2. Show the heap after a `deleteMin` operation:



Answer:



2 [20 points] Hash Tables

1. Consider a hash table with m entries, with the following hash function: $h(x) = x \texttt{ mod } m$. For each input sequence below indicate the aymptotic running time for inserting all elements in the hash table assuming the hash table uses (i) separate chaining, or (ii) linear probing. Your answer should be in big O notation, like O(m log m) (not necessarily a real answer). Please remember that the insert operation must check if the key being inserted is not already in the hash table.

   (a) `0, m, 2*m, ..., (m-1)*m      (m elements)`

      i. Separate chaining: Answer: $O(m^2)$

      ii. Linear probing: Answer: $O(m^2)$

   (b) `0, 1, 2, ..., m-2, m-1         (m elements)`

      i. Separate chaining: Answer: $O(m)$

      ii. Linear probing: Answer: $O(m)$

   (c) `0,    1,    2, ...,  m/2-2,  m/2-1,  m/2,`
   `m,  m+1,  m+2, ..., 3m/2-2, 3m/2-1, 3m/2     (m elements)`

      i. Separate chaining: Answer: $O(m)$

      ii. Linear probing: Answer: $O(m^2)$

2. For each statement below indicate whether it is true or false. Assume the has table has m entries.

   (a) Finding an element in a separate chaining hash table with n elements can be done in $O(1)$ worst case running time.

   Answer: False

   (b) If the fill factor $\lambda = n/m$ is approximatively 0.8 then inserting in a hash table using linear probing may never terminate

   Answer: False

   (c) The advantange of using quadratic probing over linear probing is that quadratic probing tends to avoid primary clustering.

   Answer: True

   (d) If m is a prime number, the second hash function $h2(x)$ never returns 0, and the table has at least one empty slot, then double hashing will always find an empty slot to insert a new element.

   Answer: True

3 [15 points] Bubble Sort

Consider the following variant of bubble sort:

```
void BubbleSort (int a[n]) {
    swapPerformed = true
    while (swapPerformed) {
          swapPerformed = false
          for (i=0; i<n-1; i++) {
                if (a[i+1] <= a[i]) {
                   Swap(a[i],a[i+1])
                   swapPerformed = true
            }
        }
    }
}
```

The function is intended to sort the array in ascending order and to run in time $O(n^2)$, but there is a bug in the function.

1. On the code above, show where the bug is and how to fix it. Your goal is to make a single change in the code such that the function sorts the array in ascending order and runs in time $O(n^2)$; you do not need to try to do other improvements.

   Answer: `a[i+1]` $\leq$ `a[i]` should be `a[i+1]` $<$ `a[i]` .

2. Then, indicate what will go wrong if we don't fix the bug, by choosing one of the following answers. Assume that the input array `a` has at least one duplicate value (a value that occurs two or more times in the array). Circle one answer below.

   - The function runs in time $O(n^3)$
   - The function sorts the array in descending order rather than ascending order.
   - The function never terminates
   - The function stops before the array is sorted.
   - The function is not stable.

Anser: The function never terminates. Once two equal elements are adjacent, $a[i] = a[i+1]$, then the function will swap them forever.

4 [15 points] Properties of Sorting Algorithms

For each of the sorting algorithms below, indicate whether it is stable, and whether it is in place:

1. Insertion sort

   Stable ? Yes In place ? Yes

2. Bubble sort

   Stable ? Yes In place ? Yes

3. Selection sort

   Stable ? Yes In place ? Yes

4. Heap sort

   Stable ? No In place ? Yes

5. Merge sort

   Stable ? Yes In place ? No

6. Quick sort

   Stable ? No In place ? Yes

5 [30 points] Merging

Consider two sorted arrays `a[m]`, `b[n]`. The pseudo-code below is the "merge" function; it computes a new array `c` with all elements in `a` and in `b`, increasing order. Viewed as a set, `c` contains the union of `a` and `b`. The function runs in time $O(m+n)$:

```
void Merge(int a[], m, int b[], n, int c[]) {
/* computes the union of a and b */
  i = 0; j = 0; k = 0;
  while (i < m || j < n) {
    if (i >= m) c[k++] = b[j++];
    else if (j >= m) c[k++] = a[i++];
    else if (a[i] > b[j]) c[k++] = b[j++];
    else c[k++] = a[i++];
  }
}
```

Write two new functions, which compute the intersection, and the difference of the sets a and b respectively. That is, the `Intersection` function computes a new array `c` with all elements that are *both* in `a` and in `b`, while the `Difference` function computes a new array `c` with all elements that are in `a` but are *not* in `b`. Each of your functions must run in time $O(m+n)$. You may assume that neither `a` nor `b` have duplicate elements; that is, `a[0]` $<$ `a[1]` $< \ldots <$ `a[n-1]` and `b[0]` $<$ `b[1]` $< \ldots <$ `b[n-1]`.

1. The `Intersection` function. Answer:

```
void Intersection(int a[], m, int b[], n, int c[]) {
/* computes the intersection of a and b */
 i = 0; j = 0; k = 0;
 while (i < m && j < n) {
 if (a[i]==b[j]) {c[k++] = a[i]; i++; j++}
 else if (a[i] < b[j]) i++;
 else j++;
  }
}
```

2. The `Difference` function. Answer:

```
void Difference(int a[], m, int b[], n, int c[]) {
/* computes the difference of a and b */
 i = 0; j = 0; k = 0;
 while (i < m) {
 if (j >= n || a[i] < b[j]) {c[k++] = a[i++]}
 else if (a[i] == b[j]) {i++; j++;}
 else j++;
  }
}
```