CSE2011 SUMMER 2014

MID-TERM EXAM

Student # ……..

Name ……..

	Value	Score
Q1	8	
Q2	12	
Q3	15	
Q4	18	
Q5	16	
Q6	15	
Q7	16	
Total	100	
Bonus	15	

Bethune College Student Ombuds Services (BC SOS) provides **free** programs to help students succeed in their courses, either through ***facilitated*** regularly scheduled study groups (PASS, bethune.yorku.ca/pass) or one-on-one peer tutoring (bethune.yorku.ca/tutoring). Students who consistently take advantage of these resources get higher grades. The Learning Skills Workshops, e.g. "Study Secrets" and "Time Management Skills", can make all the difference as students deal with

Q1 (8 points) What size of a problem can a program handle in given time? The program can use various algorithms with different running time. Fill in the table. (1 ms = 0.001s, 1 μs = 0.000,001s)

algorithm	program running time	1 second	1 hour	1 day	1 week
log n	10 ms for n=1000				
n	1 μs for n=1000				
n log n	10 μs for n=1000				
n^2	1 ms for n=1000				
2^n	0.5 s for n=10				

Q2 (12 points) Mark the column for which the statement at the top is true.

f(n)	g(n)	f(n) is **Ω**(g(n))	f(n) is **O**(g(n))	f(n) is **Θ** (g(n))
$\sqrt{n^7}$	n^3			
n log n	$n (\log n)^2$			
3 n $\log_3$ n	10 n $\log_{10}$ n			
2^n	$2^{n+0.001}$			
2^n	$(2+0.001)^n$			
$(\log n)^2$	n log log n			

Q3 (15 points) Provide the worst-case running time for listed data structures and operations (here p represents a position, e an element, and r the rank in the sequence):

a) Sequence with n elements implemented using an array of Positions where each position stores an element and an array index

addAfter(p,e)	**Θ**( )
prev(p)	**Θ**( )
remove(p)	**Θ**( )
indexOf(p)	**Θ**( )
remove(r)	**Θ**( )

b) Sequence with n elements implemented using a doubly-linked list of Positions where each position stores an element

addAfter(p,e)	**Θ**( )
prev(p)	**Θ**( )
remove(p)	**Θ**( )
indexOf(p)	**Θ**( )
remove(r)	**Θ**( )

Q4 (18 points) What is the time complexity of the algorithms used in following programs?

```
public static int f1(int n) {
      int x = 0;
      for(int i = 0; i < n*n; i += 2)
            x++;
      return x;
}
```

f1 is Θ(……..)

```
public static int f2(int n) {
      int x = 0;
      for(int i = 1; i < n*n; i *= 2)
            x++;
      return x;
}
```

f2 is Θ(……..)

```
public static int f3(int n) {
      int x = 0;
      for(int i = 0; i < n*n*n; i++)
            for(int j = 1; j < n; j *= 2)
            x++;
      return x;
}
```

f3 is Θ(……..)

```
public static int f4(int n) {
      int x = 0;
      for(int i = 0; i < n*n*n; i++)
            for(int j = 1; j < i; j += 2)
            x++;
      return x;
}
```

f4 is Θ(……..)

```
public static int f5(int n) {
      int x = 0;
      for(int i = 0; i < n*n*n; i++)
            for(int j = 1; j < i; j *= 2)
            x++;
      return x;
}
```

f5 is Θ(……..)

```
public static int f6(int n) {
      int x = 0;
      for(int i = n; i > 1; i /= 2)
            for(int j = 1; j < n; j *= 2)
            x++;
      return x;
}
```

f6 is Θ(……..)

Q5 (16 points) We have a binary tree with elements 1, 2, 3, 4, 5, 6, 7, 8, 9 (not necessarily in this order). Postorder traversal visits the nodes in the order 8, 7, 1, 9, 3, 2, 6, 4, and 5. Inorder traversal visits the nodes in the order 8, 1, 7, 5, 2, 3, 9, 4, and 6.

a) Describe a method/strategy to re-construct the tree, preferably as pseudo-code.

b) Draw the tree (follow the method described above if you have any)

Q6 (15 points) Provide short answers:

What are the 3 elements of successful recursion?

What are iterators and what are the good reasons to use them?

What is a comparator and what are the good reasons to use them?

Can you build a priority queue and adaptable priority queue using an array?

What are the advantages (give a real example of the benefits) and disadvantages of position-aware entries?

Q7 (16 points)

The following array contains a heap with integer keys. Give the final version of the array as it would look after 2 deleteMin() operations:

0	1	2	3	4	5	6	7	8	9	10	11	12
x	3	7	8	40	20	9	10	50	45			

0	1	2	3	4	5	6	7	8	9	10	11	12
x												

The following array contains a heap with integer keys. Give the final version of the array as it would look after insert(5) followed by insert(2) operations:

0	1	2	3	4	5	6	7	8	9	10	11	12
x	3	7	8	40	20	9	10	50	45	25		

0	1	2	3	4	5	6	7	8	9	10	11	12
x												

Consider a heap implemented as an array A. The heap has 10000 elements stored in A[1] … A[10000]

True / False There are 5000 leaves

True / False A[4000] > A[3999]

True / False A[4000] > A[125]

True / False In pre-order traversal (as a binary tree) the last visited node contains the largest value

True / False In post-order traversal (as a binary tree) the last visited node contains the smallest value

True / False Is it possible to determine if key 99999 is stored in the heap using no more than 14 comparisons?

BONUS (15 points)

a) Write pseudo code to output a singly-linked list in reverse order when you are NOT allowed to allocate memory dynamically. What is the running time of the algorithm?
b) Write pseudo code to output a singly-linked list in reverse order when you are ALLOWED to allocate memory dynamically. What is the running time of the algorithm?
c) You have an increasingly-sorted circular list (using an array) of n elements that is full. The front and rear pointers have been lost. Explain how you can find the smallest element in better then **O**(n).

ADDITIONAL SPACE FOR ANSWERS