Name: _____

ID Number: _____

UNIVERSITY OF VICTORIA
EXAMINATIONS- DECEMBER 2013
CSC 225 - Algorithms and Data Structures: I
Section A01, CRN 10760
Instructor: Wendy Myrvold
Duration: 3 hours

TO BE ANSWERED ON THE PAPER.

Instructions:

This question paper has eleven pages (the last page is blank in case you need extra space) plus the header page.

Students **MUST** count the number of pages in this examination paper before beginning to write, and report any discrepancy immediately to the invigilator.

Use only space provided on exam for answering questions. Closed book. No aids permitted.

Question	Value	Mark
1	15	
2	15	
3	10	
4	10	
5	10	
6	15	
7	15	
8	10	
Total	**100**	

1. Circle true or false for each question and justify your answer. No marks will be given unless there is a correct justification.

(a) [2 marks] The function $f(n) = k\ 2^k$ is in $\Omega((k+1)\,2^{k+1})$ when $k \geq 1$.

True **False**

(b) [2 marks] The function $S(n) = \sum_{i=1}^{n} i^4$ is in $O(n^4)$ for $n \geq 1$.

True **False**

(c) [2 marks] The function $S(n) = \sum_{i=1}^{n} i^4$ is in $O(n^6)$ for $n \geq 1$.

True **False**

(d) [3 marks] When designing a divide and conquer algorithm for items stored in a linked list of size n, an approach which divides the problem in half will always be faster asymptotically in the worst case than one which divides the list into one subproblem of size 1 and one of size $n - 1$.

True **False**

(e) [3 marks] It takes $\Omega(n \log n)$ time to build a heap of size n because this is how long it takes to do n insertions.

True **False**

(f) [3 marks] Hashing takes $O(1)$ time in the worst case to insert an item into a hash table.

True **False**

2.(a) [2 marks] Describe the best case scenario for a quickSort algorithm that is sorting *n* values that are all distinct.

(b) [8 marks] Propose a recurrence relation for the best case time for quickSort when the values are all distinct and solve your recurrence relation using repeated substitution.

(c) [3 marks] Give a simple expression in terms of the problem size *n* for the time complexity you determined from step (b) (express in terms of Θ notation).

(d) [2 marks] It is possible to have better running time in the best case when key values can be repeated. How can you implement quickSort to get a faster running time, and how does it change the best case time complexity?

3. Consider this method:

```
public static void use_space(int n)
{
   // Block A
   int [ ] A;
   int i, j, k, sum;

   if (n <= 1) return;

   // Block B
   sum=0;
   for (i=0; i < 1 * n ; i++)
      for (j=0; j < 2 * n; j++)
         for (k=0; k < 3 * n; k++)
            sum++;

   // Block C
   sum= sum/n;
   for (i=0; i < 4; i++) sum++;

   // Block D
   A= new int[sum];

   // Block E
   use_space(n/2);
   use_space(n/2);
}
```

For full marks For parts (a) and (b), your recurrence should be as simple as possible while still yielding the correct asymptotic (Θ) complexity.

(a) [5 marks] Propose a recurrence relation $T(n)$ that you could solve to get the asymptotic (Θ) time complexity of the use_space method. Justify your answer by explaining the time taken by each of the blocks of the code (A-E).

(b) [5 marks] Propose a recurrence relation $S(n)$ that you could solve to get the asymptotic (Θ) space complexity of the use_space method. Justify your answer.

4. Consider the following program for building a union/find data structure. Fill in the parent array, indicate the height, and draw a picture of the union/find data structure that is created for parts (a), (b) and (c).

```
public class UnionFind
{  int [ ] parent;
   public UnionFind(int n)
   {
      int i;
      parent= new int[n];
      for (i=0; i < n; i++) parent[i]= i;
   }
   public void build_union(int lower, int size)
   {
      int right_lower, left_size, right_size;

      if (size <= 1) return;

      left_size= size/2;
      right_lower = lower + left_size;
      right_size= size - left_size;

      build_union(lower, left_size);
      build_union(right_lower, right_size);
      union(right_lower, lower);
   }
   public void union(int u, int v)
   {
      int pu, pv;

      pu= find(u);
      pv= find(v);
      parent[pu]= pv;
   }
   int find(int u)
   {
       while (parent[u] != u)
       {
         u=parent[u];
       }
       return(u);
   }
}
```

(a) [2 marks] build_union(0, 2)

u	0	1
parent[u]		

The height is:

The picture:

(b) [2 marks] build_union(0, 4)

u	0	1	2	3
parent[u]				

The height is:

The picture:

(c) [6 marks] build_union(0, 8)

u	0	1	2	3	4	5	6	7
parent[u]								

The height is:

The picture:

5. Fill in the blank in the following statement then prove it is correct by induction.
If a call is made to *build_union(lower, n)* for some value of $n = 2^k$ for an integer $k \geq 0$, then the resulting union/find data structure has root node *lower* and has height *h* equal to:

[2 marks] _____ **Your proof [8 marks]:**

```
public class UnionFind
{
  int [ ] parent;
  public UnionFind(int n)
  {   int i;
      parent= new int[n];
      for (i=0; i < n; i++) parent[i]= i;
  }
  public void build_union(int lower, int size)
  {
      int right_lower, left_size, right_size;

B1    if (size <= 1) return;

B2    left_size= size/2;
B3    right_lower = lower + left_size;
B4    right_size= size- left_size;

B5    build_union(lower, left_size);
B6    build_union(right_lower, right_size);
B7    union(right_lower, lower);
  }
  public void union(int u, int v)
  {   int pu, pv;

U1    pu= find(u);
U2    pv= find(v);
U3    parent[pu]= pv;
  }
  int find(int u)
  {
F1     while (parent[u] != u)
F2     {
         u=parent[u];
      }
F3     return(u);
  }
}
```

6. [9, -2 marks for each incorrect answer] Your job is to analyse various data structures for implementing a set. The sets under consideration can contain up to n elements represented by the integers from zero up to $n - 1$. Consider the following data structures for a set S:

   (a) The s elements of set S are stored in sorted order in locations 0 to $s - 1$ of an array of size n.

   (b) The elements of set S are stored in an unsorted linked list **[the linked list should not contain any duplicate values].**

   (c) The elements of S are indicated by a *characteristic vector* which is an array of size n which has position i set to 1 if element i is in the set and 0 if it is not in the set.

Let S be a set which has size s and let T be a set of size t. Give the **worst case** time complexities of algorithms for the following set operations **as functions of** n**,** s**, and** t.

Operation	(a) Sorted array	(b) Unsorted list	(c) Characteristic vector
Is x in S?			
Add x to S (do not count time to check if x in S)			
Create union of S and T	(i)	(ii)	

Justify your time for the set union for the cases where:

(i) [3 marks] the sets are stored in sorted arrays.

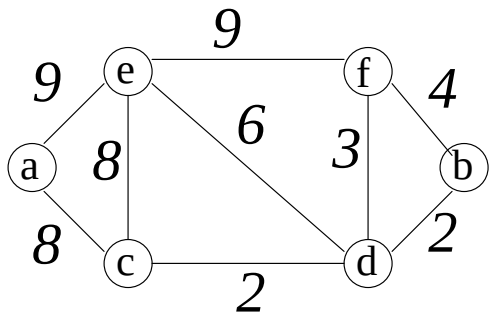
(ii) [3 marks] the sets are stored in unsorted lists.

7. Show the values of the data structures *tree*, *min_wt*, and *closest* (as described in class) after each phase of the Dijkstra/Prim minimum spanning tree algorithm. The *phase* equals the number of vertices in the tree so far.

(a) [2 marks] *tree:*

Phase:	0	1	2	3	4	5	6
a							
b							
c							
d							
e							
f							

start here



(b) [5 marks] *min_wt:*

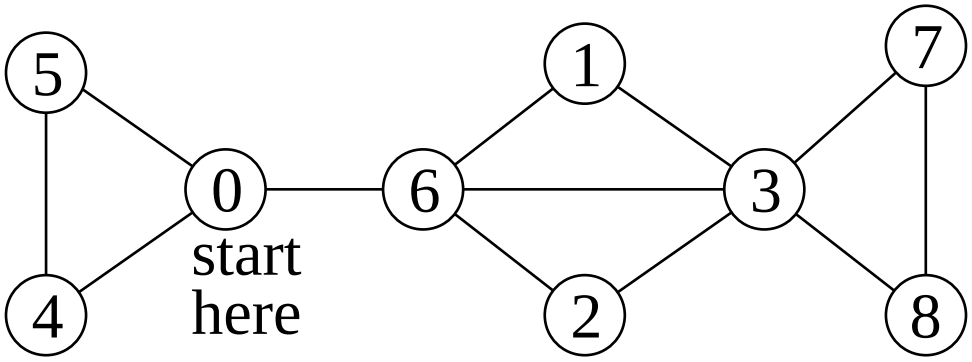
Phase:	0	1	2	3	4	5	6
a							
b							
c							
d							
e							
f							

(c) [5 marks] *closest:*

Phase:	0	1	2	3	4	5	6
a							
b							
c							
d							
e							
f							

(d) [3 mark] Mark the edges in the minimum weight spanning tree.

8.(a) [8 marks] Perform DFS (Depth First Search) on the graph given below. Start at vertex 0. When traversing the neighbours of a vertex, traverse them **in numerical order**. Show all your work including the contents of the stack (at each step), the parent array and the DFI array.



	0	1	2	3	4	5	6	7	8
Parent:									
DFI:									

Stack contents at each step (at step i, the ith edge is added to the DFS tree):

Step 0	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7	Step 8

(b) [2 marks] Mark the edges of the DFS tree on the picture.

Use this page if you need extra space. Clearly indicate the question(s) you are answering.

The End.